A

Exploring Dynamic Redundancy to Resuscitate Faulty PCM Blocks

JIE CHEN, George Washington University
GURU VENKATARAMANI, George Washington University
H. HOWIE HUANG, George Washington University

DRAM technology challenges have increased the necessity to adapt to the emerging memory technologies like Phase-Change Memory (PCM or PRAM). While such emerging technologies provide benefits like storage density, non-volatility, and low energy consumption, they are constrained by limited write endurance that becomes more pronounced with process variation. In this article, we explore a novel PRAM-based main memory system which resuscitates a group of faulty pages in a cost-effective manner to significantly extend the PCM main memory lifetime while minimizing the performance impact. In particular, we explore three different dimensions of dynamic redundancy levels and group sizes, and design low-cost hardware and software support for our proposed schemes. We aim to have minimal hardware modifications (that have less than 1% on-chip and off-chip area overheads). Also, our schemes can improve the PRAM lifetime by up to 105× (times) over a chip with no error correction capabilities, and outperform prior schemes such as DRM and ECP at a small fraction of the hardware cost. The performance overhead resulting from our scheme is less than 8% on average across 21 applications from SPEC2006, Splash-2, and PARSEC benchmark suites.

CCS Concepts:•**Computer systems organization** → **Processors and memory architectures; Redundancy;**

General Terms: Design, Redundancy, Performance

Additional Key Words and Phrases: Phase Change Memory, Write Endurance, Lifetime

ACM Reference Format:
Jie Chen, Guru Venkataramani, and H. Howie Huang, 2013. Exploring Dynamic Redundancy to Resuscitate Faulty PCM Blocks. *ACM J. Emerg. Technol. Comput. Syst.* V, N, Article A (January YYYY), 23 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

As multi-core systems begin to demand increasingly higher performance from main memory, the necessity to retain the working sets of all the cores in memory becomes significant. This calls for increased main memory capacity and density, such that the processors can achieve scalable performance and operate within specified power budgets. Current DRAM-based systems have scalability issues due to technology limitations beyond 32 nm [Process Integration and Structures 2007]. Therefore, it becomes necessary to explore alternative emerging memory technologies like Phase Change Memory (PCM or PRAM) to substitute or work alongside DRAM systems.

A preliminary version of this article appeared in the Proceedings of the 42^{nd} Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), in June 2012.
This work is supported, in part, by the National Science Foundation, under CAREER Award CCF-1149557, and OCI-0937875 grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
Authors' addresses: J. Chen, G. Venkataramani, and H. H. Huang, Department of Electrical and Computer Engineering, George Washington University, 607 Phillips Hall, The Academic Center, 801 22nd Street, NW, Washington, DC, 20052; e-mail: {jiec,guruv,howie}@gwu.edu.

DOI: http://dx.doi.org/10.1145/0000000.0000000

PCM, in particular, has been shown to exhibit enormous potential as a viable alternative to DRAM systems because it can offer up to 4× more density [Raoux et al. 2008]. PCM is made of chalcogenide glass, which has crystalline and amorphous states corresponding to low (binary 1) and high (binary 0) resistances to electric currents. There have been recent proposals that look at using PCM-based hybrid memory for future generation main memory [Qureshi et al. 2009b]. However, a drawback when using PCM arises from its limited write endurance. PCM-based devices are expected to sustain an average of 10^8 writes per cell, when the cell's programming element breaks and the write operations can no longer change the values. Prior solutions have studied wear-leveling [Qureshi et al. 2009a] and reducing the number of writes to PCM [Lee et al. 2009; Zhou et al. 2009], as well as reviving the faulty pages that were normally discarded as unusable by the memory controller [Ipek et al. 2010; Chen et al. 2011; Qureshi 2011; Schechter et al. 2010; Seong et al. 2010b; Yoon et al. 2011]. In this article, we adopt a similar goal of continuing to reuse faulty PCM blocks beyond the initial bit failures.

Our motivation behind resuscitating faulty PCM blocks is to put these blocks (that would otherwise be discarded by the memory controller) back into use, i.e., not having to retire them prematurely. As PCM-based memory begins to find widespread acceptance in the market, memory manufacturers will need to take system engineering costs into account. Frequent replacement of failed memory modules can be cost prohibitive, especially for large scale data centers.

In this article, we propose that instead of completely discarding a PCM page as soon as it becomes faulty, a group of faulty pages could be resuscitated in a cost-effective manner such that we significantly extend the PCM lifetime without adversely affecting the application performance. To this end, we explore advanced dynamic redundancy techniques that minimizes the complexity of finding compatible faulty pages to store redundant information. We explore varying levels of redundancy as bits begin to fail. Specifically, we adopt two broad classes of redundancy mechanisms- (1) Parity-based Dynamic Redundancy (PDR), where we add N redundancy (parity) bits for a given number of data bits. (2) Mirroring-based Dynamic Redundancy (MDR), where we have one redundant (mirror) bit for every data bit. Using these redundancy mechanisms, we explore several dimensions that include varying the group sizes (defined as the number of faulty PCM pages that form a group together for parity).

Our main motivation behind exploring dynamic redundancy-based techniques to improving the lifetime of PCM is driven by three factors:

— *Increase the space efficiency in the usage of faulty pages*: In MDR configuration, we mirror identical data on two compatible faulty PCM pages that effectively replicates data across both pages. As a result, both the PCM pages together store a single page worth of data, i.e., the storage density is 50%. In PDR configuration, a group G of n faulty pages have dedicated p parity blocks, that store the parity values for all of the n pages. Therefore, the storage density for $n + p$ (including the parity) pages is $\frac{n}{n+p}$. For example, with a group size of 4 that has 2 additional parity pages, the storage efficiency is 67% (17% more efficient than MDR), and for a group size of 3, the storage efficiency is 60% (10% higher than MDR). At higher values of n, we get better efficiency in terms of storage density, although finding compatible pages for larger groups becomes increasingly difficult.

— *Utilize off-the-shelf memory components without extensive hardware redesign*: Prior techniques such as ECP [Schechter et al. 2010] have to custom design the PCM chip to integrate their lifetime-enhancing techniques. Such designs increase the hardware cost, and more importantly, reduce the flexibility of switching to other lifetime-enhancing techniques in the future. To counter such drawbacks, our goal is

to maximize the use of off-the-shelf components and include techniques that would enhance lifetime with minimum changes to hardware design.
— *Explore design choices that will offer flexibility to the user*: The end-users can make an informed choice that is most suitable to their needs under a given cost budget.

To summarize, the main contributions of this article are:

(1) We explore various combinations of *dynamic redundancy techniques to resuscitate faulty PCM pages* toward improving the lifetime of PCM-based main memory. Specifically, we study additional redundancy techniques such as double parity and higher group sizes (of up to 4 pages per group) to further extend PCM lifetime over our prior work [Chen et al. 2012b].
(2) We propose low-cost hardware that can be combined with off-the-shelf PCM memory and show that only *minimal hardware modifications* are needed to implement our proposed schemes.
(3) We evaluate our design and show that we can improve the PCM lifetime by up to 105× over raw PCM without any Error Correction Capabilities at less than 1% area overhead both on-chip and off-chip. Also, we incur less than 8% performance overhead on average across SPEC2006 [Standard Performance Evaluation Corporation 2006], PARSEC-1.0 [Bienia et al. 2008] and Splash-2 [Woo et al. 1995] applications.

In this article, we make the following significant extensions to our previous work:

(1) We explore multiple parity blocks to resuscitate a group of faulty pages that can tolerate more than one fault in the same byte position across faulty pages, and quantify the associated matching complexity.
(2) We design and implement double parity redundancy scheme and the resulting hardware/software changes.
(3) We conduct additional lifetime, performance and sensitivity studies.

The rest of paper is organized as follows. Section 2 presents the background and state-of-the-art techniques in extending PCM lifetime. Section 3 presents the architectural design and implementation. We describe the evaluation setup and results in Section 4. The conclusion is presented in Section 5.

2. BACKGROUND

In this section, we present a quick review of PCM main memory or PRAM. We present related work on extending PCM lifetime, and brief overview of redundancy based techniques to tolerate faults.

2.1. PCM based Main Memory and Wear-out Problem

The conventional DRAM main memory is beginning to face scalability issues from technology limitations that prevent scaling cell feature sizes beyond 32nm [Process Integration and Structures 2007]. DRAM is also confronting power-related issues due to high leakage caused by shrinking transistor sizes. All of these have led to considerations of alternative memory technologies such as Phase-change Random Access Memory (PRAM).

A caveat in deploying PCM is its requirement of high operating temperatures for set and reset operations that directly affect the lifetime of PRAM devices. In particular, repeated reset operations at very high temperatures cause the programming circuit of phase change material to break, and permanently reset the PCM cell into a state of high resistance. This introduces limited write endurance to PCM that significantly dampens the chances of using PCM as a replacement for DRAM. Additional complica-

tions arise in PCM due to process variation effects that could further decrease write endurance in these devices.

In order to overcome PCM's limited endurance, prior works have looked at better wear-leveling algorithms [Qureshi et al. 2009a], reducing write traffic to PCM memory through partial writes [Lee et al. 2009], writing select bits [Cho and Lee 2009; Zhang and Li 2009], randomizing data placement [Seong et al. 2010a], exploring intelligent writes [Chen et al. 2012a], and using DRAM buffers [Qureshi et al. 2009b]. All of these techniques focus on applying their optimizations prior to the first bit failure. We note that these techniques are complementary to our proposed PCM resuscitation techniques, and can contribute to even higher lifetime when used in conjunction with our schemes.

Dynamically Replicated Memory (DRM) [Ipek et al. 2010] forms pairs of faulty PCM pages that do not have faults in the same byte position so that paired pages can serve as replicas of one another. Redundant data is stored in both pages to make sure that the system can read a non-faulty version of the byte from at least one of the pages. The idea behind pairing is that there is a high probability of finding two compatible faulty pages and hence, one could eventually reclaim what would otherwise be a decommissioned memory space. While this is useful, simply replicating the data in both pages can rapidly degrade the effective capacity of the memory system. We note that, by using a replication scheme, DRM merely explores MDR or mirroring. First of all, by using MDR for PCM pages that have too few errors initially, DRM can waste a lot of non-faulty bytes in the paired pages and unnecessarily replicate the entire block. Secondly, writes need to update both the mirror copies, which means that both pages will endure increased wear, and subsequently result in expedited aging of the pages contributing to their failures. In this work, we overcome the above disadvantages by exploring more dynamic redundancy techniques like PDR, which reduces unnecessary additional writes to PCM blocks. Further, we combine several PDR configurations to explore design points that will be most suitable to users' needs.

Error Correcting Pointers (ECP) [Schechter et al. 2010] is another technique that handles the errors by encoding the locations of failed cells in a table and by assigning new cells to replace them. The main disadvantage with this approach is the complexity of changing PCM chip to accommodate the dedicated pointers, as well as, the costs involved in hardware redesign. ECP incurs a static storage overhead of 12% to store these pointers. SAFER [Seong et al. 2010b] dynamically partitions the blocks into multiple groups, exploiting the fact that failed cells can still be read and reused to store data. LLS [Jiang et al. 2011] is a line-level mapping technique that dynamically partitions the overall PCM space into a main space and a backup space. By mapping failed lines to the backup, LLS manages to maintain a contiguous memory space that provides easy integration with wear leveling. When accessing a failed line, the request will be redirected to access the mapped backup line through a special address translation logic, which requires PCM chip redesign efforts and also incurs extra latency and energy. LLS also relies on intra-line salvaging, such as ECP, to correct initial cell failures. RDIS [Melhem et al. 2012] incorporates a recursive error correction scheme in hardware to track memory cell locations that have faults during write. In this work, we use off-the-shelf PCM storage and perform minimal changes to existing hardware, which lowers the cost of redesign and helps us to minimize the performance overheads.

FREE-p [Yoon et al. 2011] performs fine-grained remapping of worn-out PCM blocks without requiring dedicated storage for storing these mappings. Pointers to remapped blocks are stored within the memory block and the memory controller adds extra requests for memory to read these blocks, which results in additional bandwidth and latency overheads. As the number of faults per block increases, this approach incurs higher performance overheads due to sequential memory reads and increased mem-

ory bandwidth demands. Whereas, our proposed techniques incur significantly lower performance overheads as the memory requests to group blocks can be overlapped and performed simultaneously (as shown in Section 4).

PAYG [Qureshi 2011] is a hardware error correction scheme that allocates ECP-based error correction entries on demand. Every PCM line (64 bytes) is equipped with a local error correction pointer, ECP-1, to correct up to one error. Additional errors in a line are protected with error correction entries from global pools that are implemented as set associative structures. This requires non-trivial modification of PCM memory arrays to support multiple tag checks in those set associate structures. At a process variation of 0.2, PAYG can provide 13% more lifetime compared to ECP while incurs a storage overhead of 3.8% of memory capacity. Whereas our scheme achieves 51% longer lifetime than ECP while only requiring slightly higher storage overhead than PAYG.

2.2. Redundancy-based Techniques

Storing redundant data to achieve high availability was explored by Patterson, Gibson, and Katz for Redundant Arrays of Inexpensive Disks (RAID) [Patterson et al. 1988] in 1987, where the original five RAID levels were presented as a low-cost storage system. Since then, RAID has become a multi-billion dollar industry [Katz 2010], and inspired many similar concepts such as Redundant Arrays of Independent Memory (RAIM) [Hayslett 2011], Redundant Arrays of Independent Filesystems (RAIF) [Joukov et al. 2007], and so on. While our work leverages a number of redundancy techniques, we actually aim to reuse faulty PCM pages, whereas RAID was designed to recover the data on a failed hard disk, which statistically has a higher availability compared to average 10^8 writes per PCM cell. In the context of PCM-based memory, we face a set of new problems, such as mapping management, asymmetric read/write performance, and limited write cycles. On the very high level, our idea has similar spirit as HP AutoRAID hierarchical storage system [Wilkes et al. 1996], where RAID 1 and 5 are deployed at two different levels, with the former used for high performance data access, and the latter for high storage efficiency. In AutoRAID, the decision is made upon active monitoring of workload access patterns and data blocks are migrated automatically in the background. In contrast, our scheme employs a unique redundancy level at different stages of the PCM lifetime, which is selected to maximize its usability and performance.

3. DESIGN OVERVIEW

In this section, we first describe the overview of our hardware design and later show the software support needed.

3.1. Incorporating Dynamic Redundancy Techniques into PCM

We assume that the PCM-based main memory starts without any faults and has wear-leveling algorithms in place to uniformly distribute the writes throughout the pages. For each write to a page, a read-after-write is performed to determine if the write succeeded. For cases where Error Correcting Code (ECC) exists onchip, the first few bit faults can be tolerated using these pre-built schemes. In particular, SECDED (single-error correction, double-error detection) ECC can correct one bit error, while Hi-ECC [Wilkerson et al. 2010] can correct up to four bit errors, both of which can be utilized to tolerate errors before beginning to use our redundancy mechanisms. After the point when the error correction capabilities cannot tolerate any more faults, the PCM main memory would be forced to discard the faulty pages in the absence of resuscitation techniques, that leads to quickly diminishing capacity of the memory. To

maximize PCM lifetime with minimal overheads and low cost, we propose redundancy mechanisms that recycle a group of faulty PCM pages in a cost-effective manner.

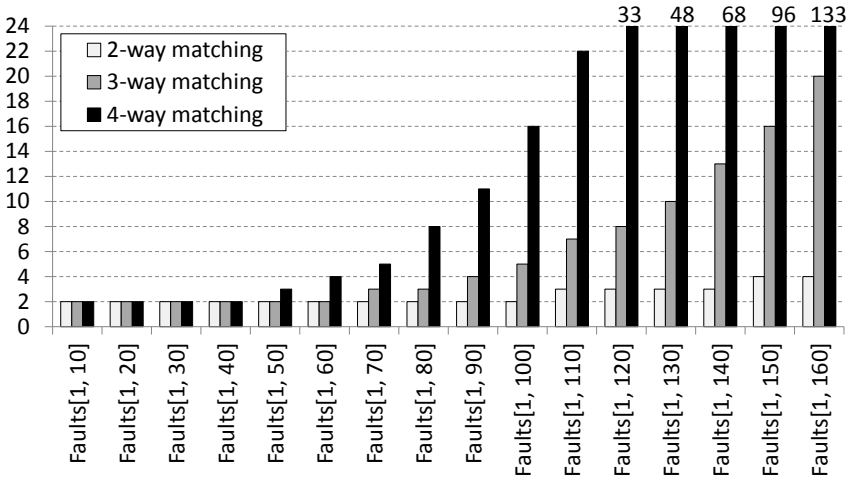
In this work, we adopt two redundancy mechanisms: Mirroring-based Dynamic Redundancy (MDR), where we have one redundant (mirror) bit for every data bit, and Parity-based Dynamic Redundancy (PDR), where we add N redundancy bits for a given number of data bits. For MDR, the selection of two matching PCM pages is relatively easy - as long as they do not have the fault in the same bit location. However, the matching is more complicated for PDR. In the following, we will describe this process in detail.

When the first fault in the PCM page k occurs (beyond the tolerance limit of already incorporated error correction schemes), we temporarily decommission this page and place it in a separate pool of PCM pages $POOL_{PDR}$, where the faulty pages are waiting to be matched with other compatible faulty pages. At this point, we disable the ECC computation, an operation supported by most ECC-based memory controllers [Qin et al. 2005]. We reuse the parity bits to store the faulty byte vector within the already built-in parity support, i.e., for each byte we have a bit to indicate whether it is faulty or not. We start with the PDR techniques, in which one or two dedicated parity pages are associated with a group of data pages. Note that the parity pages are stored separately in a DRAM buffer, which we will discuss later and assume to have correct bits at all times. The number of data pages per group (n) is determined based on the matching complexity. For higher values of n, it becomes exceedingly difficult to find compatible faulty pages.

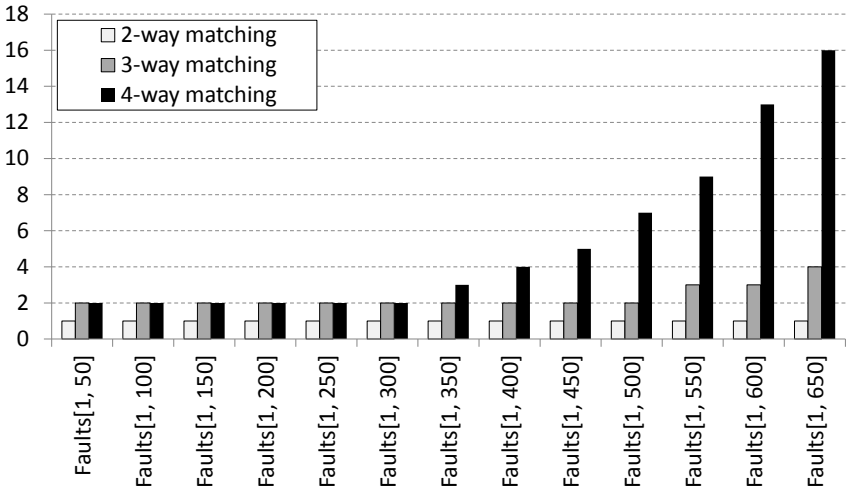
When we decommission the PCM page k, we copy its contents into one of the reserve PCM pages, which serve to store the data in the faulty pages until the matching is done. By default, we assume that there are 10,000 such reserve PCM pages. To start, we randomly pick a group G of compatible faulty pages from $POOL_{PDR}$. Alternatively, a low-cost approximate pairing algorithm similar to one in DRM [Ipek et al. 2010] can be used to assemble the group G of compatible faulty pages from $POOL_{PDR}$. The mapping information is stored as tuples $MAP_{PDR} = \{k_1, k_2, ..., k_n, p_1, p_2, ..., p_m\}$, where n is the group size and m is the number of parity pages (typically one or two), which are managed by the Operating System (OS) (described in 3.4). For example, in a PDR group size of three with one parity page, a tuple of the form, $\{k_1, k_2, k_3, P\}$ is stored, where k_1, k_2, k_3 are the compatible faulty pages and P is the parity page.

Figure 1 shows the average number of random trials needed to find a group of compatible faulty pages. With a single parity page for a group of faulty pages, we define compatibility between faulty pages as *no two pages* having fault in the same byte position (Figure 1(a)). For two parity pages, we define compatibility between faulty pages as *no more than two pages* having fault in the same byte position (shown in Figure 1(b)). In these experiments, we randomly pick a group of n faulty pages and check if they are compatible, where n is determined by the $n - way$ matching shown in our experiments. If they are found to be not compatible, we repeat with another new randomly picked group of pages until we find them to be compatible. The average number of tries needed is recorded. We repeat our experiments by bounding our faults in various ranges as shown in Figure 1. These experiments were done by assuming a pool of one million 4KB pages and we averaged over 10,000 samples. We also tried a pool of 10,000 4KB pages with over 1,000 samples and observed similar results.

We define *n-way, m-parity threshold* as the maximum number of errors beyond which more than three trials are needed on average to find a compatible group of pages. We chose three trials to define our thresholds because the average number of trials needed to find compatible pages increases sharply beyond three trials and the complexity of the matching algorithm increases due to repeated invocations in finding a possible group of compatible pages. As an instance, in Figure 1(a) under 1-parity, beyond 80



(a) Up to one fault in any given byte position

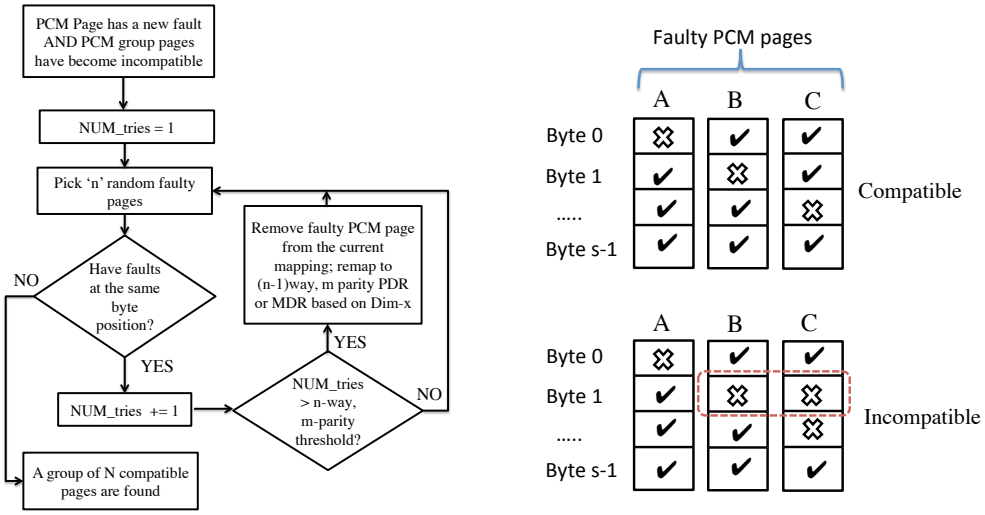


(b) Up to two faults in any given byte position

Fig. 1. Matching compatible faulty pages– Average number of random trials needed for the two-way, three-way and four-way matching. X-axis indicates the bounds on total number of randomly distributed faults in each 4 KB PCM page.

faults, 3-way matching needs more than three trials on average to find a compatible group of 3 pages. Also, the number of trials under 3-way matching sharply rises for more than 80 faults per page. Therefore, 80 is defined as 3-way, 1-parity threshold. By similar reasoning, in Figure 1(b), 350 can be defined as 4-way, 2-parity threshold.

Under 1-parity scheme, we define 160 faults as *1-parity limit threshold* because the matching complexity is too high for any reasonable cost redundancy mechanism to find compatible pages. Similarly, under 2-parity, we define 2,048 faults as *2-parity limit threshold* because of very high matching complexity. We note that these thresholds are defined for our 4 KB sized PCM pages and could be easily computed for other page sizes through similar experiments.



(a) Faulty PCM page matching process (b) Compatible vs. incompatible faulty pages

Fig. 2. Faulty page matching algorithm

We note that we are able to determine these thresholds beforehand (say, at machine installation time) empirically using the size of the PCM block, maximum number of matching trials, and the type of user-desired parity configurations. These thresholds are application-independent and are not runtime-specific, because the writes issued by the applications (and the subsequent wear on the device) are already uniformly distributed in the PCM device through advanced wear-leveling algorithms.

3.2. Exploring Three Design Dimensions

In our design, we define three dimensions, Dim-1, Dim-2 and Dim-3 based on the matching complexity of finding compatible faulty PCM pages. In these dimensions, we explore various combinations of switching between group sizes, number of parity pages and parity/mirroring in these three dimensions. We choose to explore the above three dimensions primarily to provide more user-friendly options and allow the users to decide what is for their best interests. Our experiments quantify the resulting lifetime and the associated performance impact. We note that the end users should be able to choose the configuration that suits their needs. Figure 2 depicts an overview of our faulty page matching algorithm.

Dim-1: We begin with 3-way, 1-parity based PDR. We switch to MDR mode when one of the newly occurring faults in an already matched PCM page (in PDR) surpasses the 3-way, 1-parity threshold and renders the group pages incompatible. This new fault will be detected by the read-after-write operations in PCM-based memory [Yang et al. 2007]. When this happens, we reassign the faulty PCM page to a new pool $POOL_{MDR}$, where pages are waiting to be matched with other compatible faulty pages in the mirroring fashion. We use the MDR technique to tolerate bit faults for the remaining portion of the pages' lifetime (until 1-parity limit threshold of 160 faults) and their corresponding mappings are stored in MAP_{MDR}, managed by the OS. Similarly, the mappings can be represented as tuples of $\{k_1, k_2\}$ where k_1 and k_2 are compatible faulty pages that mirror the content of each other. It is worthy to point out that even after MDR mapping is adopted for some PCM pages, other pages that are currently in

MAP_{PDR} will continue to be in PDR mode until there is a need for remapping these faulty pages.

Dim-2: In this case, throughout the lifetime of the device, we continue to operate in the PDR mode where a group of PCM pages have an associated parity page. We begin with 3-way, 1-parity based PDR, and upon surpassing the 3-way, 1-parity threshold, we downgrade the group size to two and continue in 2-way, 1-parity based PDR mode until we reach the 1-parity limit threshold of 160. This results in the need for configuring the memory controller to monitor for the group sizes corresponding to faulty PCM page and MAP_{PDR} tables to handle the relevant group size information.

Dim-3: Unlike Dim-1 and Dim-2, a group of faulty PCM pages are protected by two parity pages that lets up to two group pages to have faults in the same byte position. We begin with 4-way, 2-parity based PDR, and keep operating under that mode until a newly occurring fault exceeds the 4-way, 2-parity threshold of 350 faults. After this, we downgrade the group size from four to three, and operate under 3-way, 2-parity based PDR until we reach the 3-way, 2-parity threshold of 600 faults. At this time, we downgrade the group size further to two, and operate under 2-way, 2-parity PDR mode until we reach the 2-parity limit threshold of 2048 faults. Again, the memory controller monitors for different group sizes of faulty PCM pages and MAP_{PDR} tables.

On a read to a main memory page k_1, we first determine whether the target page is faulty or not. If k_1 is not faulty, the memory request proceeds as usual. If it is faulty, we determine whether k_1 is in MAP_{PDR} or in MAP_{MDR}. The accesses to MAP_{PDR} and MAP_{MDR} can be performed in parallel. As a result, we obtain the corresponding information on this group of blocks, k_i (where i is determined by the number of group pages in the current PDR mode) and p_j (where j is determined by the number of parity pages in PDR). Alternatively, the information about the group block, k_2 is obtained for MDR. Extra memory requests are issued by the memory controller for the corresponding group and parity blocks depending on the PDR/MDR mapping. Upon reading the blocks, we use the faulty byte vector (reused ECC parity bits) to determine the locations of faulty bytes.

Once the memory requests are satisfied, the data block for k_1 is reconstructed based on the PDR parity or the MDR mirror. We note that these extra requests could potentially lead to a performance bottleneck by saturating the bus bandwidth and delaying the reads that are on the critical path. In order to reduce the performance overheads, additional optimizations such as a temporary read buffer that stores reconstructed memory blocks shall be used. Such optimizations can offer faster read accesses to otherwise faulty memory blocks, however care should be taken to ensure that write accesses properly update the actual PCM and parity blocks, keeping them consistent at all times. To ensure data consistency, every write operation should make sure to invalidate or update the appropriate read buffer entries. Since writes are off the performance-critical path, lazy invalidate or update schemes can be used for read buffer entries.

On a write to a main memory page k_1, we also determine whether the target page is faulty or not. If faulty, we find the corresponding group blocks from the mapping tables. In case of MDR, we issue two memory requests to k_1 and k_2. For PDR, we need to update only k_1 and corresponding parity pages, p_j. Therefore, the number of memory requests needed for a write is always two (three for Dim-3), instead of $n + 1$ ($n + 2$ for Dim-3) memory requests needed for a read.

3.3. Hardware Implementation

One of our primary goals is to lower hardware implementation cost and complexity. Also, the performance impact resulting from our proposed hardware changes should

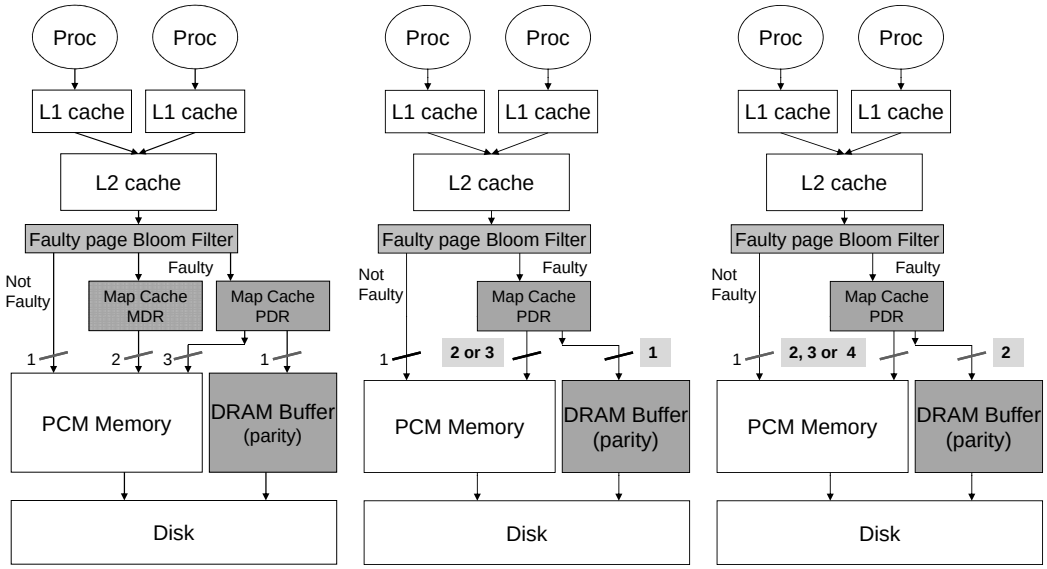


Fig. 3. Hardware modifications and structures (shown in gray boxes) needed to incorporate our PDR/MDR schemes into the processor hardware. The number of memory requests issued to the memory structure is shown next to the arrows (e.g., PDR always issues 3 PCM requests in Dim-1 for a group size of 3; in Dim-2, the number of requests (2 or 3) depends on the current group size), and in Dim-3, the number of requests (2, 3, or 4) also depends on the current group size.

not adversely affect the normal processor performance. Figure 3 shows the hardware structures that are needed to implement our design in a cost-effective manner with low performance overheads.

The first task is to find out whether a page is pristine or has faults. A naive approach to detecting faulty pages is to make memory controller perform additional read-back after a write to determine whether the write succeeded [Ipek et al. 2010]. However, performing repeated read-after-write requests during every write operation can be time consuming and hence, we use compact structures such as Bloom Filters [Bloom 1970] to record information about the faulty pages. Once we determine the faulty/pristine status of the page, we can allow the pristine (fault-free) pages to directly access the memory as usual, while the faulty pages can be directed to lookup one of the MAP structures. One caveat with using structures such as Bloom Filters is that they have a potential for false positives, i.e., a page may be reported to be faulty when it actually does not have faults. Fortunately, our experiments show a negligible rate of false alarms, and usually such cases can be addressed by checking if the page has an entry in one of the MAP tables or by directly reading the faulty byte vector associated with the PCM page.

Once the pages are determined to be faulty, the next step is to check the associated mapping under MAP_{PDR} (and MAP_{MDR} too for Dim-1). This is to reconstruct the original data page for a read or update the necessary parity information on a write corresponding to the faulty PCM page. The mapping information is managed by the OS and frequent invocation of the OS for mapping lookups can result in expensive overheads. Hence, to speedup this process, we use limited-entry caches, $CACHE_{PDR}$ and $CACHE_{MDR}$ to temporarily store MAP_{PDR} and MAP_{MDR} respectively. In our current implementation, we use two small 1024-entry buffer caches to store the map-

pings. This organization is similar to Translation Look-aside Buffers (TLB). Upon a miss in both the mapping caches, the OS service routine is invoked and a mapping lookup is performed. An entry is created in the corresponding mapping cache depending on the dynamic redundancy scheme under which the requested page is mapped. We may remove an entry from the mapping caches for the following reasons: (1) when one of the PCM pages has suffered permanent failure (exceeded parity limit thresholds) and needs decommissioning, or (2) when PDR to MDR remapping needs to be done due to increased matching complexity associated with PDR. Note that, under Dim-2 and Dim-3, we would continue to remain in PDR mode, but with different group sizes. However, due to decreasing group sizes, previously formed groups may need to be reorganized and would require us to flush the corresponding entries from $CACHE_{PDR}$.

We use a separate DRAM buffer to store the parity for faster access by the multi-core processor, which we believe is more cost-effective than simply investing on additional PCM capacity for parity. This is mainly motivated by two facts: (1) DRAM provides fast data access and does not suffer from write endurance problem. Note that the parity information needs to be accurate and cannot have errors in order to recover the data from faulty PCM block. DRAM buffer offers a much better alternative to PCM in this regard. (2) The parity information is much smaller than data itself - for a group of n data pages resident in the main memory, there is only a maximum of two corresponding parity pages. So, DRAM buffer (to store the parity corresponding to the resident RAM pages) can be a lot smaller than the actual PCM-based RAM. Note that this DRAM buffer is simply meant to be a *fast access, cache-like storage for parity information*. The parity pages and the mapping information are permanently *stored* in the lower level disk along with the data pages. As we will observe in our evaluation (Section 4), fast accesses to parity offered by DRAM buffers can easily outweigh some of the demerits such as their inability to retain data during power loss. If really needed, we could avoid the DRAM warmup penalty (after power-on) by investing in solutions like low cost, Flash RAM backup to store the contents of DRAM [Brant et al. 1998]

Finally, we would need to discard pages that have more than parity limit thresholds and decommission these pages for permanent failure. To do so, during rematching of pages (when pages become incompatible upon errors in the corresponding same byte position), we determine if the candidate pages have more than parity limit thresholds. We permanently mark such pages for removal, and invoke the remapping algorithm to reconstitute the other group pages associated with the failed page under MDR or PDR.

3.4. Software Support

The OS manages the mapping of faulty PCM pages, as well as parity pages. It maintains two lists, the *free list* for the DRAM parity buffer, and the *candidate list* for faulty PCM pages. In the beginning, the free list contains all the pages in the DRAM buffer. Upon forming a new group of compatible faulty PCM pages, one entry from DRAM buffer free list is allocated for storing parity and this information is maintained in the OS. When the group is disbanded due to incompatibility among the pages, the OS will put the corresponding parity page back into the free list. When the free list is empty, the OS will stop the matching process.

The OS invokes the matching algorithm every time when the memory controller detects a new faulty page or if a page becomes incompatible with its group pages. This new faulty PCM page is inserted into the candidate list where compatible pages can be paired. The OS performs random selection from the candidate list to quickly form a group of the compatible pages. After a page is successfully matched, it will be removed from the candidate list. Our experiments show that not every fault results in incompatibility between the group pages (see Section 4.5). Remapping happens far

less frequently than the number of faults occurring in pages, which greatly reduces the OS overhead for page remapping.

The OS manages the mapping between faulty PCM pages using a hash table that is stored in a reserved area of the memory address space. Assuming that we need to maintain the mapping information for every page, the worst case storage overhead for a 4GB PRAM with 4KB pages would only be 2.5MB (20 bits per page for 1M pages). Also, that the OS interventions for accessing and updating the mapping information should be kept minimal in order to avoid high performance overheads. As we show in Section 4, with the help of mapping caches and dynamic redundancy levels, we are able to minimize performance impacts on a wide range of the benchmarks.

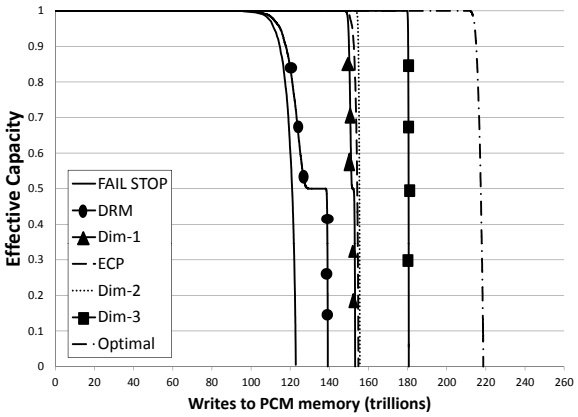
4. EVALUATION

In this section, we first analyze the extended lifetime achievable through our techniques, and then show the performance impact because of our proposed hardware changes. Finally, we present the sensitivity studies that show the performance of our system under various configurations. To present fair comparison results with prior schemes, our configuration parameters are similar to, and derived from earlier works such as DRM [Ipek et al. 2010] and ECP [Schechter et al. 2010].

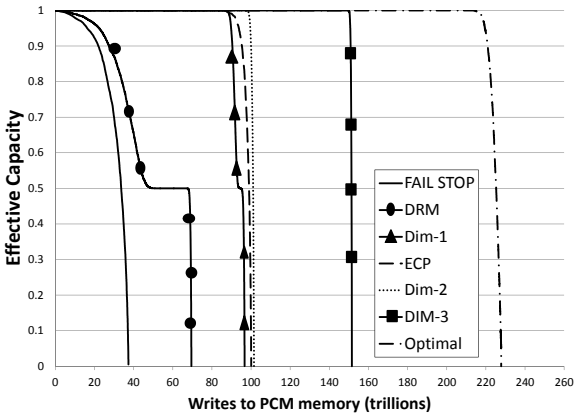
4.1. Lifetime

Figure 4 compares the *lifetime (measured as the total number of writes that can be performed to the PCM)* along with the diminishing effective capacity left in the PCM device. We show the results corresponding to Dim-1, Dim-2, Dim-3 design choices, and contrast them with prior schemes such as Fail_Stop (that does not have any Error Correction capabilities and discards a PCM block as soon as the first fault occurs), DRM and ECP schemes. We also show the Optimal case under Dim-3 configuration, where we continue to use the faulty PCM page until all of the bytes have faults. We assume a baseline of 4GB PCM memory with 4KB page size, and write operations happen on a granularity of 64 Byte blocks. In addition, we assume a 50% probability that any single write operation would flip a particular bit. *We model the PCM cell lifetime to follow a normal distribution with a mean of* 10^8 *and three different process variation factors of 0.1, 0.2, and 0.3* (shown in Figure 4). We assume that hard or permanent faults occurs in PCM memory cells independently, which means the occurrence of one fault does not influence the spatial location of the next fault [Ipek et al. 2010]. Note that there are other possible failure modes such as column and row failures. In such cases, we would simply take the failed column and row offline and decommission them from further use. As part of future work, we plan to extend our proposed techniques to handle a wide range of failures including how to salvage failed rows and columns.

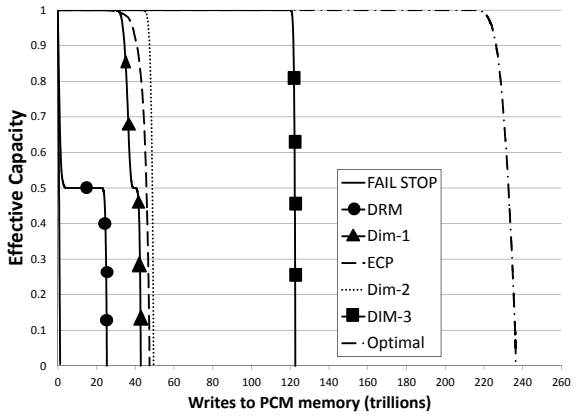
Our experiments show that with a 4KB page size, there is a high probability of at least one byte having a lifetime at the tail-end of the lifetime distribution. This makes the Fail_Stop scheme quickly decommission all of the pages within a short window of writes before the PCM's effective capacity drops to zero. This effect is especially more pronounced at higher levels of process variation, as shown in our experimental results. Although DRM is able to tolerate up to 160 faults in each page before decommissioning the page for permanent failure, the main disadvantage with DRM is that replicated writes to two different pages speeds up the aging process of both pages. DRM offers an extended lifetime of approximately 0.13$\times$ to 22$\times$ over Fail_Stop scheme depending on the process variation factor. The third scheme, ECP, can tolerate up to 6 faults in each 64 Byte block and a 4KB PCM page is decommissioned as soon as the seventh fault occurs in one of its constituent 64 Byte blocks. This lets ECP achieve a lifetime improvement of 0.26$\times$ (variance=0.1) to 41$\times$ (variance=0.3) over Fail_Stop.



(a) variance=0.1



(b) variance=0.2



(c) variance=0.3

Fig. 4. Effective capacity versus the total number of writes issued to the PCM main memory.

Our Dim-x designs achieve the lifetime results that are comparable to, or better than, the ECP scheme at a small fraction of the ECP's hardware cost. In both Dim-1 and Dim-2, we first deploy 3-way, 1-parity PDR that stores parity separately from PCM data pages. Under Dim-1, PRAM capacity begins to shrink after reaching the 3-way, 1-parity threshold when pages are mapped under MDR and mirrored onto each other. Dim-1 design yields the lifetime results that are slightly worse than ECP (1% when the variance is 0.1 to 9.5% when the variance is 0.3). Under Dim-2, PRAM capacity does not shrink after 3-way, 1-parity threshold because the faulty PCM pages still continue to operate in PDR mode albeit with smaller group sizes of two. Therefore, PRAM capacity shrinks only after PCM pages are permanently discarded for failure (after reaching 1-parity limit threshold of 160 faults). This lets the PRAM capacity to degrade more gracefully in Dim-2. Dim-2 exhibits better lifetime than ECP (0.5% for the variance=0.1 to 4.4% when the variance=0.3). Under Dim-3, the faulty PCM page groups have double parity pages that can tolerate up to two faults in the same byte position. The PRAM capacity begins to shrink only after PCM pages are permanently discarded (reaching 2-parity limit threshold of 2,048 faults). We note that the faulty PCM pages continue to operate in PDR mode– first in 4-way, 2-parity mode and subsequently reducing the group size to three and two after reaching 4-way, 2-parity and 3-way, 2-parity thresholds respectively. We find that Dim-3 exhibits much better lifetime than ECP (0.18$\times$ for the variance=0.1 to 2.6$\times$ when the variance=0.3).

This shows a good trade-off from the hardware cost perspective, as our design involves using mostly off-the-shelf components versus the extensive modifications to main memory design needed by ECP. Overall, our proposed redundancy designs achieve good lifetime improvements for PCM, ranging from 0.26$\times$ (when the variance is 0.1 in Dim-1) to 105$\times$ (for the variance of 0.3 in Dim-2) over Fail_Stop.

4.2. Performance Impact

We evaluate the performance impact resulting from our design using SESC [Renau et al. 2006], a cycle-accurate, execution-driven simulator. Our baseline system models an Intel Nehalem-like four-core processor [Intel Corporation 2010] running at 3GHz, 4-way, out-of-order core, each with a private 32KB, 8-way set-associative L1 and a shared 4MB, 16-way set-associative L2. The L1 caches are kept coherent using the MESI protocol. The block size is 64Bytes in all caches. We model 4GB PCM main memory with 4KB pages with read access latency of 50ns and write access latency of 1μs [Numonyx 2009]. For storing parity, we include an additional 16MB DRAM that can accessed simultaneously during the PRAM accesses. When the DRAM buffer is full, we model a 96000 cycle (32μs) latency to write parity information to disk.

Every L2 access first queries a bloom filter, that has a three-cycle latency to determine whether the page is faulty or not. We use 1024-entry $CACHE_{MDR}$ and $CACHE_{PDR}$ in our experiments, where we observe an average miss rate of 3% (and 8% worst case). Note that we do not assume fixed latencies for memory lookups corresponding to MDR or PDR. Our configuration setup has a common memory bus between cores to fully model memory read/write latencies and bandwidth.

For faulty pages, in Dim-1, we access $CACHE_{MDR}$ and $CACHE_{PDR}$ simultaneously each of which has a 2 cycle latency. In Dim-2 and Dim-3, we just access $CACHE_{PDR}$. On a mapping cache miss, we model an additional latency of 500 cycles to perform mapping lookup from lower levels of memory hierarchy and store it in the mapping cache for future use. Upon receiving all the pages in a group, we have a 20 cycle penalty for parity reconstruction and recovering faulty bytes from mirror copies.

We show performance overheads on two sets of scenarios for Dim-1, Dim-2 and Dim-3 configurations:

— *average case*: In this scenario, we assume that a randomly picked page has 50% probability that it is faulty. Therefore, a randomly picked 50% of the PCM pages are considered to be faulty. 50% of faulty pages (25% of the total pages) are mapped under 3-way, 1-parity PDR, and 50% (25% of total pages) are under MDR in Dim-1 and under 2-way, 1-parity PDR in Dim-2. In Dim-3, of the 50% faulty pages, 33.33% (16.67% of total pages) faulty pages are mapped under 4-way, 2-parity PDR, 33.33% (16.67% of total pages) faulty pages are mapped under 3-way, 2-parity PDR and 33.33% (16.67% of total pages) faulty pages are mapped 2-way, 2-parity. The remaining 50% of total pages are assumed to be non-faulty.
— *stress case*: In this scenario, we assume that a randomly picked page is always faulty. A randomly picked 50% of the PCM pages are mapped under 3-way, 1-parity PDR and the remaining 50% pages are under MDR in Dim-1, and under 2-way, 1-parity PDR in Dim-2. Under Dim-3, 33.33% of the faulty pages are under 4-way, 2-parity PDR, 33.33% faulty pages are under 3-way, 2-parity PDR and 33.33% faulty pages are under 2-way, 2-parity PDR.

We use 21 different memory-intensive and CPU-intensive application mix from SPEC2006 [Standard Performance Evaluation Corporation 2006], PARSEC-1.0 [Bienia et al. 2008] and Splash-2 [Woo et al. 1995] benchmark suites with reference input sets. For SPEC2006 and PARSEC applications, we fast forward the first five billion instructions and simulate the next one billion in detail. For Splash2 applications, we simulate them from start to end. In addition, we run SPEC2006 benchmarks as single-core applications individually one at a time. For PARSEC and Splash2 benchmarks, we spawn four parallel threads, one each on every core that share L2 and lower level memory substructures.

Figure 5(a) shows the performance impact of our designs in the average case scenario. We see that all 21 of our applications show the overheads less than 8%, and the highest performance overhead (7.9%) in gcc, followed by 7.3% in cactusADM under Dim-3. Under Dim-1, we notice an average of 1.87%, 0.8%, and 1.2% across SPEC2006, PARSEC and Splash2 respectively, whereas the corresponding averages under Dim-2 are 2.2%, 0.8%, and 1.4%, and under Dim-3 are 2.7%, 1.7%, and 1.6%. We notice an increased memory access rate in Dim-3 (due to continuous use of 2-parity PDR configuration) along with a higher rate of misses in mapping caches contributes to increased performance impact than Dim-2 and Dim-1. This effect is especially pronounced in benchmarks such as gcc, cactusADM, and fluidanimate. Also, issuing multiple requests for group pages creates contention for memory bus bandwidth and degrades performance in benchmarks such as mcf, streamcluster and ocean.

Figure 5(b) shows the performance impact of our designs in the stress case scenario. We see that, all 21 of our applications show the overheads less than 17%, and the highest overhead (16.7%) occurs in gcc, followed by 14.3% for cactusADM in the Dim-3 design. In Dim-1, we notice an average of 3.78%, 1.64%, and 2.14% across SPEC2006, PARSEC and Splash2 respectively, whereas the corresponding averages in Dim-2 are 4.66%, 2.71%, and 2.58%, and that in Dim-3 are 5.40%, 3.24%, and 2.97%. Due to multiple memory accesses (depending on the group size) and demand for mapping cache entries, we notice an increased average performance impact in Dim-3, Dim-2 than Dim-1 across various benchmark suites. Particularly, this effect is seen in benchmarks such as gcc, mcf, and fluidanimate. Similarly, issuing multiple requests incurs performance impact in benchmarks such as cactusADM, mcf, and ocean.

4.3. Area Overheads

We use Cacti 5.3 [Hewlett-Packard 2010], an integrated model for cache and memory access time, cycle time, area, leakage and dynamic power. We use this tool to model

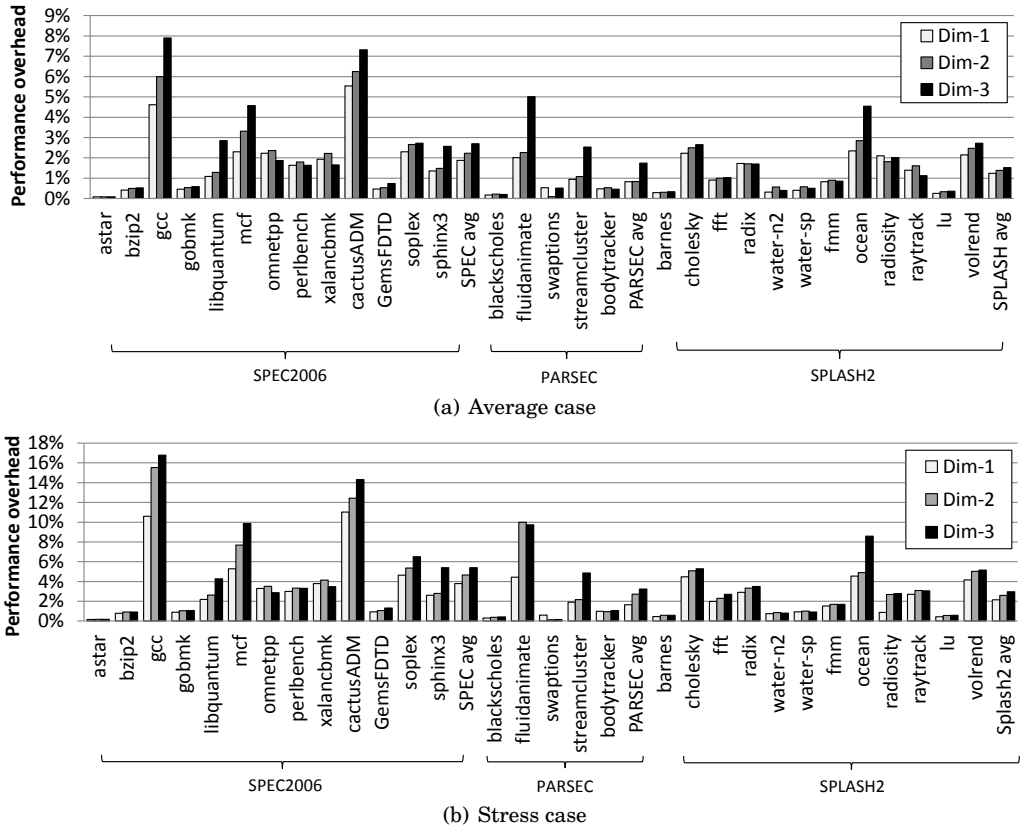


Fig. 5. Performance Overheads of our schemes on Spec2006, PARSEC-1.0 and Splash-2 applications.

Table I. Area overheads of on-chip mapping caches, bloom filter and off-chip DRAM buffer

Processor Die	263 mm^2 [Intel Corporation 2010]
Bloom Filter	2.25 mm^2
$CACHE_{PDR}$	0.16 mm^2
$CACHE_{MDR}$	0.08 mm^2
On-chip area overhead	0.94%
16 MB DRAM Buffer overhead (compared to 4GB PRAM)	$<$0.5%

our two 1024-entry mapping caches, and a compact bloom filter that has space to store 1 million PCM page entries inside the processor chip. We assume that these hardware structures would be integrated with an on-chip memory controller. We use 45 nm technology node in our experiments.

Table I shows the area estimates of our proposed hardware. We find that our area overheads both on-chip and off-chip are less than 1% and our proposed hardware can be easily integrated into the existing processors with minimal changes. We note that this is much cheaper than prior works such as ECP [Schechter et al. 2010] that incur substantial area overheads to store replacement bits along with additional circuitry like row-decoders in order to store the error correcting pointers.

Table II. Average number of extra memory accesses per PCM memory access

	Percentage of faulty pages in PCM RAM				
	20%	40%	60%	80%	100%
Dim-1	0.6	1.1	1.8	2.4	(1, 3)
Dim-2	0.6	1.2	1.8	2.4	(2, 3)
Dim-3	1.0	2.0	2.9	3.9	(3, 5)

4.4. Additional Memory Accesses

One of the key factors behind energy and power consumption by our redundancy schemes is the extra memory requests (mirror page, or group pages and parity) on top of the PCM memory request. We conduct experiments to measure the average number of additional memory accesses per every memory access request sent to the PCM under our Dim-x designs. Table II shows the results of our experiments. The rows represent the Dim-x and the columns represent the percentage of faulty PCM pages. The values in the individual cells denote the average number of extra memory requests (including the parity requests sent to the DRAM) on top of the memory access issued to the PCM.

Dim-1 begins to operate under 3-way, 1-parity PDR and later switches to MDR. Under PDR, we need three extra memory requests (two for PCM and one for DRAM). MDR needs one extra memory request to the PCM mirror copy. The number of extra memory requests rise from 0.6 to 2.4 as we see increasing percentage of faulty pages. When all of the PCM pages have faults, they are mapped either under PDR or MDR. Depending on the redundancy configuration under which the faulty pages are currently mapped, the average number of extra accesses ranges from 1 to 3.

Dim-2 begins to operate under 3-way, 1-parity PDR and later switches to 2-way, 1-parity PDR. 2-way PDR needs two extra memory requests (one for PCM and one for DRAM). The number of extra memory requests rise from 0.6 to 2.4 as we see increasing percentage of faulty pages. When all of the PCM pages have faults, they are mapped either under 3-way or 2-way PDR. Depending on the redundancy configuration under which the faulty pages are currently mapped, the average number of extra accesses ranges from 2 to 3.

Dim-3 begins to operate under 4-way, 2-parity PDR and later switches to lower group sizes. Due to higher group and parity size compared to other Dim-x schemes, the average number of extra memory requests rise from 1.0 to 3.9 as we see increasing percentage of faulty pages. When all of the PCM pages have faults, they are mapped under one of 4-way, 3-way or 2-way PDR configurations. Depending on the group size of the PDR under which the faulty pages are currently mapped, the average number of extra accesses ranges from 3 to 5.

We note that when the PCM pages are non-faulty, the energy and power overheads on main memory accesses are *zero*, i.e, there are no extra memory requests and hence no additional energy needed. As the percentage of faulty pages grows and reaches 100%, the memory subsystem consumes extra energy for additional memory requests. In the worst case for Dim-3 and if the pages are mapped under 4-way, 2-parity PDR, the main memory accesses could consume $5\times$ extra energy when compared with non-faulty memory accesses. Fortunately, given the relatively small number of main memory accesses in most applications (that are typically satisfied by caches) and that extra memory requests are needed only for faulty pages, the energy overheads are unlikely to dominate the overall system energy.

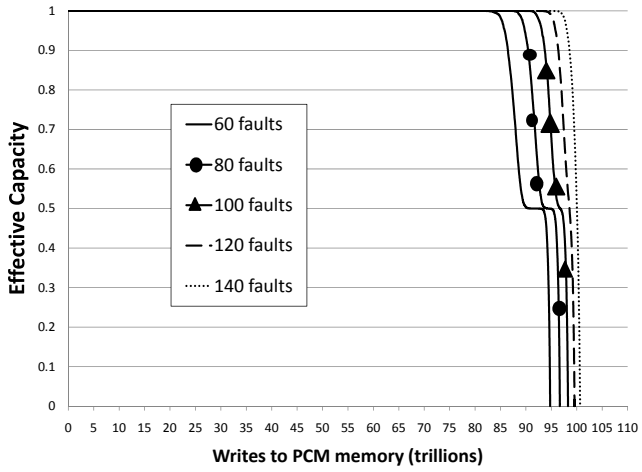


Fig. 6. Lifetime comparison for various configurations of Dim-1 (at variance=0.2). For each configuration, we use PDR until the specified number of faults, and later we switch over to MDR for the rest of the PCM block's lifetime.

4.5. Sensitivity Experiments

In this subsection, we present further analysis to show how our designs behave under various parameters and possible configurations. We show that such analyses present key insights to the user along with experimental justification for choosing user-desirable set of parameters toward our implementation.

4.5.1. Lifetime vs. Redundancy Levels. In Dim-2 and Dim-3, PDR is used throughout the lifetime of PCM device. An advantage of using just PDR is that every PCM block incurs only the write operations directly intended for them, i.e., the use of PDR configuration itself does not impose additional writes to the PCM blocks. However, in Dim-1, we switch to MDR beyond a specific point in time and this may likely accelerate the wear-out of PCM blocks. Since mirroring duplicates the writes to two different PCM blocks, every write ages two separate PCM blocks, contributing to the diminishing capacity of the PCM device. Therefore, we seek to investigate when to switch to MDR during the lifetime of the PCM block.

Figure 6 presents comparison of lifetime for various configurations of Dim-1. In each case, we begin with PDR with group size of 3. After we cross a "specific number of faults" shown in each configuration, we switch to MDR where data is mirrored onto two PCM blocks. In other words, the 3-way, 1-parity threshold is varied for each configuration as shown in the chart. It is important to note that, as we begin to operate in PDR mode with a higher number of faults, the cost associated with three-way mapping increases sharply (Section 3.1). For example, the average number of random trials to complete a three-way matching for PCM blocks with up to 80 faults is one more than three-way matching for up to 60 faults and the corresponding lifetime improvement is 2.1%; whereas, the number of trials to do three-way matching for PCM blocks with up to 140 faults is ten-fold compared to the blocks with up to 60 faults and the corresponding lifetime improvement is 5.2%. Our experiments clearly show that there are diminishing returns in PCM lifetime improvement if we try to remain longer under PDR with higher order group sizes.

4.5.2. Sensitivity of PDR to DRAM Buffer Size. A factor that could be critical to minimizing the performance impact in PDR is the size of the DRAM buffer (that we use for parity

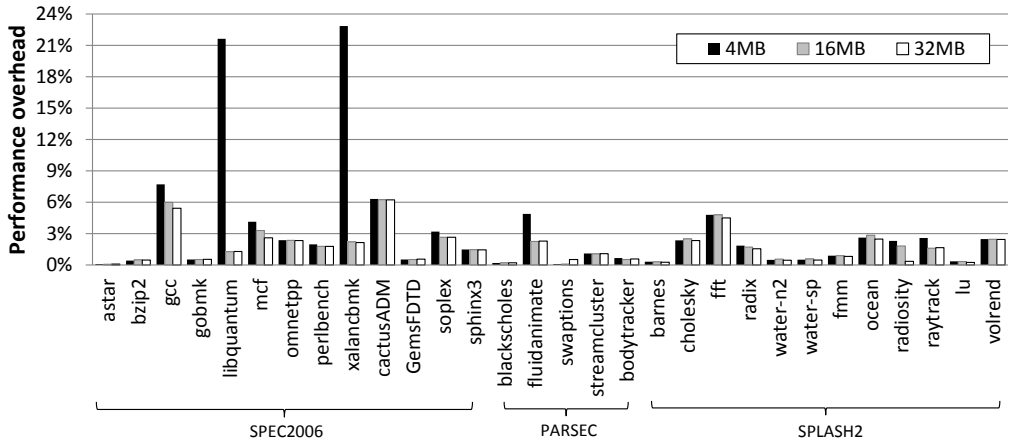


Fig. 7. Performance overheads for different sizes of DRAM buffer in 1-parity PDR (average case).

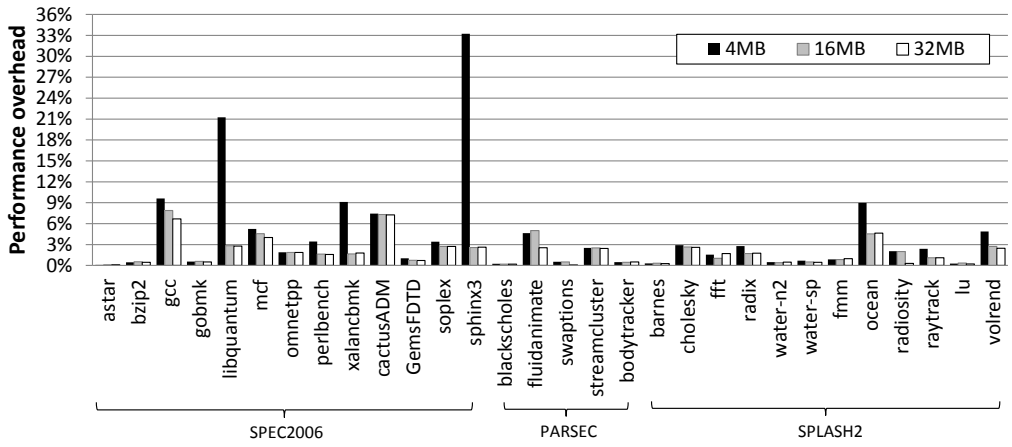


Fig. 8. Performance overheads for different sizes of DRAM buffer in 2-parity PDR (average case).

lookup). To investigate this effect, we present additional experiments to determine the size of DRAM buffers needed for storing the parity in our benchmarks. Due to smaller ratio of parity to PCM data pages, we find that relatively smaller DRAM buffer sizes (compared to PCM main memory) work very well toward minimizing the overall performance impact. Figure 7 shows the overheads experienced by 1-parity PDR when we experiment with 3 different DRAM buffer sizes, viz., 4MB, 16MB and 32MB. Our results show that 16MB is sufficient to keep the performance overheads at less than 7% (average case), while the 4MB DRAM buffer incurs high performance overheads of up to 23% in libquantum and xalancbmk benchmarks. A 32MB DRAM buffer shows negligible benefit over 16MB for performance overheads in almost every benchmark, indicating that DRAM capacity (above 16MB) is no longer a bottleneck beyond the initial compulsory misses to parity information. Figure 8 shows the overheads experienced by 2-parity PDR when we experiment with 3 different DRAM buffer sizes, viz., 4MB, 16MB and 32MB. Our results again show that 16MB is sufficient to keep the performance overheads at less than 7% (average case), while the 4MB DRAM buffer incurs high performance overheads of up to 33% in sphinx3 and 21% in libquantum

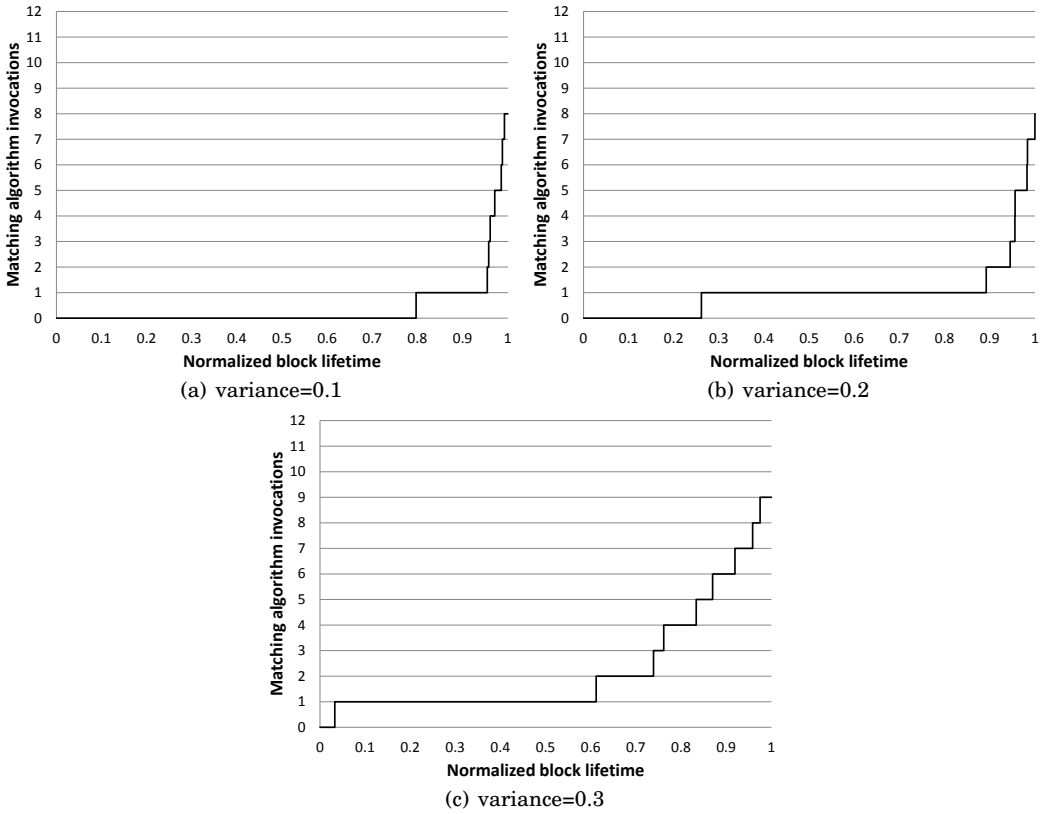
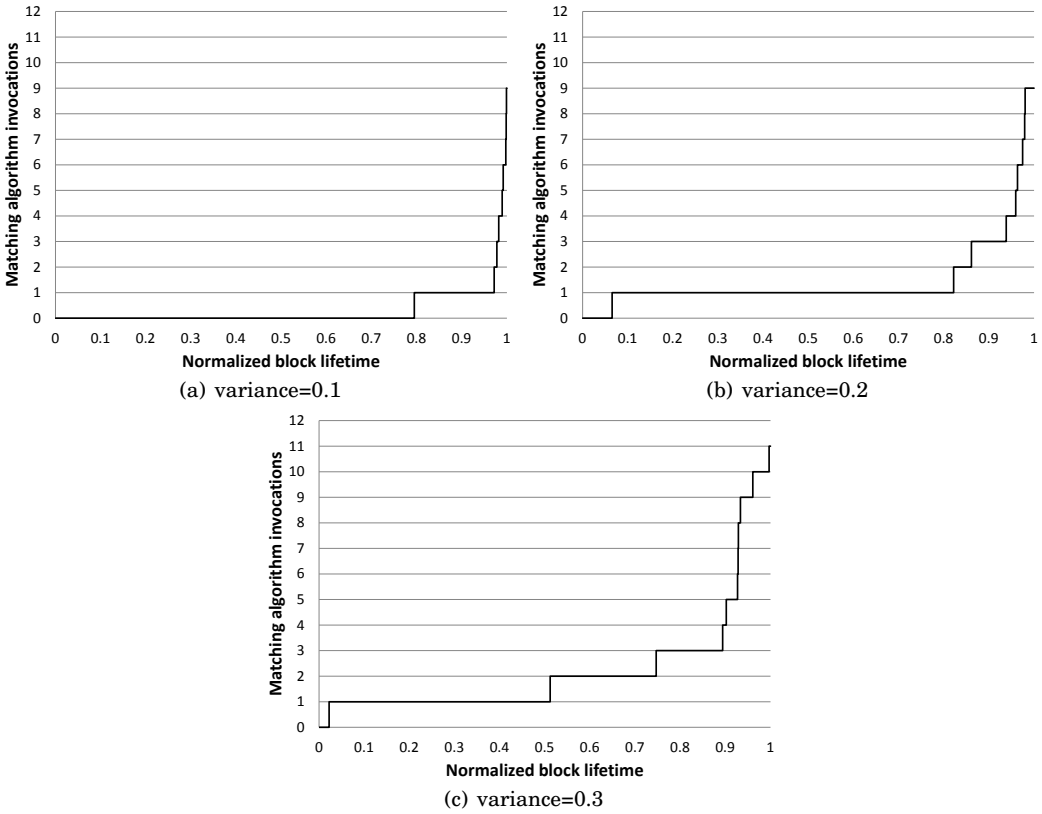


Fig. 9. Frequency of invocation of matching algorithm through PRAM block's lifetime under Dim-1 and Dim-2 (1-parity PDR).

benchmarks. A 32MB DRAM buffer shows very small benefit over 16MB for performance overheads in almost every benchmark, indicating that DRAM capacity (above 16MB) is no longer a bottleneck beyond the initial compulsory misses to parity information.

This result shows that the amount of DRAM buffer needed to maintain low performance overheads across the three benchmark suites is just $\frac{1}{256}^{th}$ of the capacity invested in PCM main memory. Furthermore, this 16MB DRAM buffer has shown less than 17% overheads even for the stress case (seen in Figure 5(b)).

4.5.3. Frequency of Matching Algorithm Invocations. To understand the OS and software overhead, we perform the experiments to quantify the average number of times that matching algorithm needs to be invoked during a PRAM block's lifetime. Figure 9 presents the results for process variation factors of 0.1, 0.2 and 0.3. In this test, we count the number of writes (presented as normalized lifetime), that has been performed to a PRAM block, before the block encounters the first bit fault. Now that the block is no longer pristine, the matching algorithm is invoked to find a group of compatible blocks. As additional writes are performed to this block and its group pages, they incur more bit faults, and eventually become incompatible due to faults in the same byte positions. At this point, the matching algorithm needs to be invoked again to find a new set of compatible pages for this faulty PCM block. We repeat this match-



(a) variance=0.1 (b) variance=0.2

(c) variance=0.3

Fig. 10. Frequency of invocation of matching algorithm through PRAM block's lifetime under Dim-3 (2-parity PDR).

ing process until the PRAM block has exceeded 1-parity (or 2-parity for Dim-3) limit threshold (when we discard the block permanently). Figures 9 and 10 show the results of our experiments. We can see that matching algorithm invocations are often clustered towards the end of PRAM block's lifetime for process variation of 0.1, while it is more spaced out for process variation of 0.3. In all of the cases, we find that average number of matching algorithm invocations is less than 11 throughout the PRAM block's lifetime. Clearly, the matching algorithm accounts for a small fraction of performance overhead in comparison to the actual writes performed on the PRAM block.

5. CONCLUSIONS AND FUTURE WORK

In this article, we explore a number of dynamic redundancy techniques to resuscitate faulty PCM pages and improve the lifetime of PCM-based main memory systems. We explore different design choices along three design dimensions that, (1) use different numbers of faulty pages per group, (2) tolerate faults with different number of parity pages, and (3) have parity and mirroring based redundancy techniques. We show that, by intelligently combining various design choices, the lifetime of PRAM can be improved by up to 105$\times$ over Fail_Stop.

As future work, we will study ways to incorporate our mechanisms in large scale systems and heterogeneous memory types. We will gather the intrinsic properties that affect memory-related behavior in applications including energy and power, and re-

search ways to adapt our mechanisms to the changing demands. As emerging memory types are used pervasively in future generation computer systems, such investigations will hold the key to successfully integrating them.

REFERENCES

C. Bienia, S. Kumar, J.P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. *Princeton University Technical Report TR-811-08* (January 2008).

Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13 (July 1970), 422–426. Issue 7.

William A. Brant, Michael E. Nielson, and Edde Tin-Shek Tang. 1998. Power failure responsive apparatus and method having a shadow dram, a flash ROM, an auxiliary battery, and a controller. In *US Patent 5,799,200*.

Jie Chen, R. C. Chiang, H. Howie Huang, and Guru Venkataramani. 2012a. Energy-aware writes to non-volatile main memory. *SIGOPS Oper. Syst. Rev.* 45, 3 (Jan. 2012), 48–52.

Jie Chen, Guru Venkataramani, and H. Howie Huang. 2012b. RePRAM: Re-cycling PRAM faulty blocks for extended lifetime. In *Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (DSN '12)*. IEEE Computer Society, Washington, DC, USA, 1–12. http://dl.acm.org/citation.cfm?id=2354410.2355179

Jie Chen, Zachary Winter, Guru Venkataramani, and H. Howie Huang. 2011. rPRAM: Exploring Redundancy Techniques to Improve Lifetime of PCM-based Main Memory. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*.

Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance. In *MICRO*.

Dave Hayslett. 2011. System z Redundant Array of Independent Memory. In *IBM SWG Competitive Project Office*.

Hewlett-Packard. 2010. CACTI 5.3. *http://quid.hpl.hp.com:9081/cacti/* (2010).

Intel Corporation. 2010. Intel Core I7-920 Processor. *http://ark.intel.com/Product.aspx?id=37147* (2010).

Engin Ipek, Jeremy Condit, Edmund B. Nightingale, Doug Burger, and Thomas Moscibroda. 2010. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. In *ASPLOS*.

Lei Jiang, Yu Du, Youtao Zhang, B.R. Childers, and Jun Yang. 2011. LLS: Cooperative integration of wear-leveling and salvaging for PCM main memory. In *DSN*. 221 –232. DOI:http://dx.doi.org/10.1109/DSN.2011.5958221

Nikolai Joukov, Arun M. Krishnakumar, Chaitanya Patti, Abhishek Rai, Sunil Satnur, Avishay Traeger, and Erez Zadok. 2007. RAIF: Redundant Array of Independent Filesystems. *MSST* 0 (2007), 199–214.

Randy H. Katz. 2010. RAID: A Personal Recollection of How Storage Became a System. *Annals of the History of Computing, IEEE* 32, 4 (2010).

Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *ISCA*.

Rami Melhem, Rakan Maddah, and Sangyeun Cho. 2012. RDIS: a recursively defined invertible set scheme to tolerate multiple stuck-at faults in resistive memory. In *Proceedings of the International Conference on Dependable Systems and Networks*.

Numonyx. 2009. Phase Change Memory: A new memory to enable new memory usage models. *White Paper http://www.numonyx.com/* (2009).

David A. Patterson, Garth Gibson, and Randy H. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). In *SIGMOD*. 109–116.

Devices Process Integration and Structures. 2007. International Technology Roadmap for Semiconductors. *http://www.itrs.net* (2007).

Feng Qin, Shan Lu, and Yuanyuan Zhou. 2005. SafeMem: Exploiting ECC-Memory for Detecting Memory Leaks and Memory Corruption During Production Runs. In *HPCA*.

Moinuddin K. Qureshi. 2011. 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 (MICRO-44 '11)*.

Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009a. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO*.

Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009b. Scalable high performance main memory system using phase-change memory technology. In *ISCA*.

S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. 2008. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.* 52 (July 2008). Issue 4.

Jose Renau and others. 2006. SESC. *http://sesc.sourceforge.net* (2006).

Stuart Schechter, Gabriel H. Loh, Karin Straus, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA*.

Nak Hee Seong, Dong Hyuk Woo, and Hsien-Hsin S. Lee. 2010a. Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of the 37th annual international symposium on Computer architecture*.

Nak Hee Seong, Dong Hyuk Woo, Vijayalakshmi Srinivasan, Jude A. Rivers, and Hsien-Hsin S. Lee. 2010b. SAFER: Stuck-At-Fault Error Recovery for Memories. In *MICRO*.

Standard Performance Evaluation Corporation. 2006. SPEC Benchmarks. *http://www.spec.org* (2006).

Chris Wilkerson, Alaa R. Alameldeen, Zeshan Chishti, Wei Wu, Dinesh Somasekhar, and Shih-lien Lu. 2010. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA*.

John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 1996. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.* 14 (February 1996). Issue 1.

S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. 1995. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *ISCA*.

B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu. 2007. A Low Power Phase Change Random Access Memory using a Data Comparison Write Scheme. In *ISCAS*.

Doe Hyun Yoon, Naveen Muralimanohar, Jichuan Chang, Parthasarathy Ranganathan, Norman P. Jouppi, and Mattan Erez. 2011. FREE-p: Protecting Non-Volatile Memory against both Hard and Soft Errors. In *HPCA*.

Wangyuan Zhang and Tao Li. 2009. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO*.

Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *ISCA*.