Increasing PCM Main Memory Lifetime

Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem and Daniel Mossé

Department of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania USA
{apf75,miaozhou,sab104,childers,melhem,mosse}@cs.pitt.edu

Abstract

The introduction of Phase-Change Memory (PCM) as a main memory technology has great potential to achieve a large energy reduction. PCM has desirable energy and scalability properties, but its use for main memory also poses challenges such as limited write endurance with at most 10^7 writes per bit cell before failure. This paper describes techniques to enhance the lifetime of PCM when used for main memory. Our techniques are (a) writeback minimization with new cache replacement policies, (b) avoidance of unnecessary writes, which write only the bit cells that are actually changed, and (c) endurance management with a novel PCM-aware swap algorithm for wear-leveling. A failure detection algorithm is also incorporated to improve the reliability of PCM. With these approaches, the lifetime of a PCM main memory is increased from just a few days to over 8 years.

I. Introduction

Current system designs have low power and energy consumption as major requirements. As main memory has become a primary energy consumer [1], many low-power memory systems have been proposed that use new technologies as replacements for DRAM [2], [3], [4]. One technology that has potential is Phase-Change Memory (PCM) due to its low energy, good read performance, non-volatility and bit addressability [5], [4], [3], [2], [6]. However, PCM poses challenges that have to be addressed for it to be used as a main memory replacement. Specifically, PCM suffers from limited endurance (i.e., it wears out due to write operations) and expensive write operations (i.e., high latency and energy). Indeed, too many writes to a PCM main memory will lead to a short device lifetime, poor performance and high energy consumption.

Other memory technologies also suffer from limited endurance and expensive writes – Flash [7], [8] is the most common example. Much attention has been given to improve Flash memory lifetime and performance [7], [9]. As an example, Mylavarapu et al. introduce algorithms that avoid erase operations and apply wear leveling (WL) to enhance lifetime and performance [7]. Because the problems associated with Flash endurance/performance [8], [9] are different from the ones for PCM, Flash WL algorithms are of limited use in a PCM main memory. Flash WL algorithms [7] avoid erasing a page on every write by allocating a new clean physical page. This allocation is unnecessary for PCM due to its bit-addressability. PCM also has a larger overall endurance (i.e., it can sustain 10^7 writes rather than the 10^4 to 10^6 writes for Flash) and does not require predefined blocking.

The use of PCM in main memory has recently been proposed with techniques applied to increase PCM lifetime. The PCM storage device presented in [10] implements a read-before-write (RW) loop at the bit level to improve reliability and extend lifetime. The work in [5] uses read-before-write, row-level rotation (RL) and segment swapping (SS) as endurance enhancements at the device level. RL equalizes wear at the row level by rotating cache lines. SS is done by swapping two segments: the one currently being written and the one that is least-frequently-written (LFW). However, the large segment size (1MByte) used in SS [5] degrades lifetime compared to a small segment size because the distribution of writes to a large segment can be skewed. Nevertheless, large segments are used in [5] to reduce the costs associated with searching for the LFW segment during a swap.

A system level approach is used in [3] to incorporate PCM in the memory hierarchy. This work proposes a hybrid memory, where a large PCM memory is augmented with a small DRAM that acts as a "page cache" for the PCM memory. The page cache helps performance by buffering frequently needed pages. It also helps endurance by reducing the number of writes to PCM with write combining and coalescing. Although the page cache filters writes to PCM, it does not fully mitigate the endurance problem. Additional techniques are applied at the cache line and block levels. At the cache line level, only the lines modified in a page are written to PCM. To avoid unbalanced damage from writes, cache lines are rotated on a page. Finally, swapping is used at the block level for wear leveling.

In this paper, we propose three new approaches to address the endurance problem when PCM is used in main memory. First, a novel cache replacement policy is applied to reduce writebacks from DRAM to PCM. Second, read-Write-Read (RWR) and page partitioning techniques are used to remove unnecessary writes and act as a failure detection mechanism. This technique is similar to device-level write differencing [10], [5]; however, our

approach is implemented in the memory controller to reduce write bandwidth. Finally, a new swap algorithm is used for endurance management. It has low overhead yet allows small block granularities for effective WL. Overall, our methods achieve a memory lifetime of at least eight years, effectively solving the wear problem for PCM main memory.

II. A PCM Main Memory Architecture

Writes are expensive in PCM: they are about 5-10x slower and consume 10x more energy than reads [6]. In current devices, a PCM cell typically supports around 10^7 writes [6]. Thus, PCM will wear-out quickly in a main memory. Because a DRAM page cache has been shown to be very important for performance (and endurance to a lesser degree) [3], we assume that a PCM main memory architecture will use one. The DRAM page cache is a set-associative structure, with a relatively high degree of associativity. Figure 1 contrasts a current DRAM architecture (left) with one that uses PCM (right). The memory controller (MC) manages the DRAM page cache and the PCM devices. It controls the information flow between the CPU, PCM and DRAM. Tags and status information are tracked by the MC and stored in its own internal SRAM. DRAM and PCM are managed in small pages (a few kilobytes) for good performance and low implementation overhead. Endurance is improved by writing to PCM devices only when a dirty page is evicted from DRAM (i.e., a writeback). A writeback policy reduces the number of PCM writes.

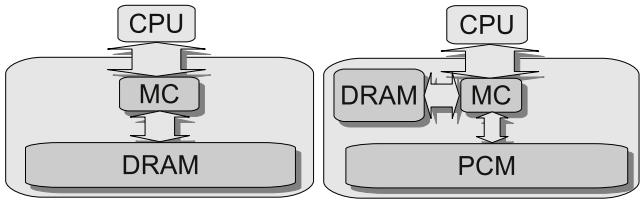

Fig. 1. DRAM and PCM + DRAM Architecture

III. Endurance Enhancement Techniques

Our proposed schemes manage writes since these operations are the ones that cause PCM cells to fail [6]. We describe several techniques, including (a) a *N-Chance cache replacement mechanism* for writeback minimization, (b) *Read-Write-Read and page partitioning* to avoid unnecessary writes and (c) a practical *swap WL algorithm* that distributes writes uniformly among PCM pages.

A. N-Chance Cache Replacement Policy

Set-associative processor caches usually implement a Least Recently Used (LRU) policy to select a victim block from a set for replacement. Such a policy could be used for the DRAM page cache. However, PCM has asymmetric read and write behavior: reads are fast, cheap (energy) and harmless, while writes are slow, expensive and harmful (i.e., wears out). The usual LRU policy does not consider the asymmetric nature of PCM read/write operations.

We depart from LRU and propose a new *clean-preferred victim selection policy* (CLP). This policy gives preference to unmodified entries (i.e., clean pages) in a page set when choosing a victim. CLP aims to keep dirty entries in the page cache to increase the probability that writes are coalesced. It can reduce latency and energy by trading an expensive writeback for a small number of low cost reads (i.e., it's cheaper to read a clean page from PCM than to write a dirty one). However, it can also have a negative impact by raising the page cache's miss rate too much. CLP always extends the PCM lifetime, but the effect on latency and energy depends on how much preference is given to clean pages when selecting victims.

Based on these observations, we propose a family of clean-preferred replacement policies, called *N-Chance*. The parameter "N" reflects how much preference is given to clean pages. The N-Chance policy is shown in Algorithm 1. This policy selects a victim from a "page set" as the oldest clean page among the N least recently used pages. If such a page does not exist, the LRU page is used. Note that 1-Chance is equivalent to LRU. Also, if N is equal to the cache associativity, N-Chance will evict all clean entries in a set before picking a dirty page.

Algorithm 1 N-Chance Algorithm in a M-Way Cache

```
Populate S with the entries with same cache index
Set S ⇐ SortByLastTimeUsed(S) {S[1] = LRU}
if CheckEntryStatus(S[1]) == INVALID then
   EntryToUse = 1
else
   EntryToUse = 1
   for I = 1 to N do
      if CheckEntryStatus(S[I]) == CLEAN then
         EntryToUse = I
         break
      end if
   end for
end if
return GetWayOfEntry(S[EntryToUse])
```

B. Avoiding Unnecessary Writes

PCM failures are due to write operations [6]. Thus, it is useful to avoid any unneeded writes, such as rewriting information that is already stored on a page. We propose two schemes to remove these writes: (a) page partitioning

and (b) page differencing with a RWR scheme. These schemes are used together: page partitioning avoids write-back of clean sub-pages and RWR removes unnecessary writes within a sub-page.

Page partitioning divides pages into sub-pages. Each sub-page has a separate dirty bit, which the memory controller uses on page replacement. Only sub-pages with their dirty bit set are actually written. Energy and endurance benefit from a small sub-page size, but a small size needs many dirty bits per page. Page partitioning applies only to writebacks – a full page is always read from the PCM memory on a page cache miss.

RWR is used to avoid writing PCM memory locations that already hold the value to be written. This situation can arise when, for example, only a single word is dirty in a sub-page. The original sub-page in the PCM is read and compared with the new sub-page to be stored. The differences between the original and new information are written to PCM. After the PCM write, a final read and comparison is done for fault detection. If the value read does not match what should have been written, then the PCM bit cells for the written location have failed.

Note that PCM memory has two interesting reliability characteristics: it is highly resistant to radiation and reads do not damage bit cells. A very reliable PCM memory can be achieved, without redundancy or Error Correcting Codes (ECC), by verifying that the correct value was indeed stored after the write. The read and comparison (after the write) detects this situation and action can be taken for the failure (e.g., allocating new PCM locations to hold the data). The extra read has negligible impact on performance and energy since reads are very efficient.

Page partitioning and RWR have the most impact on applications that have a high miss rate in the page cache. In these applications, each page lives in the cache only for a short period, which reduces the chance that multiple writes are captured before a page is evicted. In turn, the amount of modified data on a page is reduced.

C. Endurance Management

Although a reduction in writes due to the previous techniques will increase PCM lifetime, the distribution of writes across the PCM devices can still affect lifetime. Our experiments show that applications often have a highly skewed distribution of writes to PCM (i.e., from page cache writebacks). 70% of writes are directed to 1% of the pages and 90% of the writes are directed to 20% of the pages. The other pages are primarily read only. This behavior quickly wears the bit cells of the heavily written pages, leading to a lifetime for the overall PCM main memory of a mere months.

To more evenly wear pages, we propose a WL technique that swaps pages on page cache writebacks. This scheme distributes writes to help ensure that all pages have a similar amount of wear at any time. The scheme relies on the notion of a logical page, which is mapped to a PCM physical page (i.e., actual PCM locations). Our swap algorithm is shown in Algorithm 2, where L is the logical page to write and P is the physical page that is allocated to L. The swaps are done on pages, rather than sub-pages, to minimize implementation overhead.

Algorithm 2 Swap Algorithm

```
{Writeback of Logical page L onto a physical page P}
if SwapCondition() == FALSE then
   write L data on P
else
   P' = SelectSwapTargetPage()
   P ⇐ P' {Copy data from P' to P}
   write L data on P'
end if
```

The swap algorithm assumes that L will be written more than L', so a swap of the physical pages (for L and L') will equalize the number of writes. The "swap condition" in Algorithm 2 determines when a swap occurs. Since a swap operation adds an additional write (both pages have to be written), frequent page swapping will lower memory lifetime. A natural condition to swap is when the number of writes to L's physical page crosses a threshold. To implement this condition, a counter per physical page is needed. A less precise but much lower cost alternative is to use a single global counter. A swap is done when the total number of writes to the whole PCM crosses a threshold. The loss of precision means that there can be more uneven wear – some pages receive more writes than others. However, as the number of writes increases over time, the difference in wear will "even out". The global counter has the desirable property that highly written logical pages will be more frequently swapped, which happens in most applications due to their skewed write distributions.

The selection of a target page (P') for the swap is crucial to achieve uniform wear. A bad choice would trade two physical pages that have a similar number of writes. The best choice is to use the LFW physical page. However, finding the LFW page is extremely expensive due to the large number of physical pages that have to be searched to find the minimum count. Furthermore, LFW requires a counter per page. A much simpler solution picks a random physical page as the target. The probability of choosing a highly written target page is low due to the large number of physical pages and the small number of highly written pages. The random algorithm will ultimately approximate LFW for a large number of writes.

The swap algorithm uses a "mapping table" to map

logical and physical pages. It is indexed by logical page number; each table entry contains the physical page address for the logical page. The table size is small relative to PCM size. For example, a 4GBytes PCM with 2KBytes page size needs only 6MBytes (using 3 bytes per entry). The table is updated only when a swap is done. It is kept in both PCM and DRAM (for fast lookup on page replacement).

Another aspect of swapping is page size: a small page size causes less data to be copied and creates a large physical page space over which to distribute writes. A small page also has a high probability of avoiding specific application behaviors that would impact lifetime. The disadvantage of a small size is the need to track many pages, since each logical page can map to any physical page.

Our implementation of the swap algorithm uses a global write counter for the swap condition and randomly selects the target page. The global counter is a modulus counter. A swap is done when the counter overflows. We use random page selection due to its low cost – it avoids the need for page usage counters. The target page is selected by generating a random page number in the valid address space. This number identifies the physical/logical page to swap. The swap operation exchanges the logical page entries in the mapping table for the source and target pages. It also exchanges the data on the associated physical pages. Although a swap causes two page reads/writes on a writeback, it introduces only one additional page read/write beyond what would be required anyway.

IV. Experimental Evaluation

To understand the effectiveness of our techniques, we evaluated their impact on PCM lifetime. We used a main memory architecture (referred to as "PMMA") with PCM and a DRAM page cache that implements the model in Figure 1. Our evaluation is done with simulation. We configured PMMA with a 4GBytes address space, a 2KBytes page, and a 256Bytes write sub-page. This page/sub-page configuration worked the best on average for our benchmarks. We evaluated other page/sub-page sizes, but we do not report the results for brevity. The PCM subsystem is based on 1Gbit DDR-266 PCM devices [11]. The page cache is implemented with a 256MBytes DDR2-1066 DRAM based on Micron's 1Gbit 533MHz DDR2DRAM devices. The page cache uses 224MBytes of the DRAM. It is 14-way set associative. The remaining 32MBytes of DRAM holds the mapping table.

Our simulation strategy uses Simics with a custom simulator for PMMA. Simics is used to gather memory traces, which are input to the separate PMMA simulator. Simics models four 1.6 GHz x86 cores, with 32KBytes instruction and data caches. A 4MBytes L2 cache is shared per pair of cores. The PMMA simulator contains a configurable timing and power model for both DRAM and PCM. The simulator is exceptionally accurate because it models the system at the bus and device event level. This simulator was used to obtain performance and energy results for the endurance techniques. It accounts for the performance/energy overhead of the endurance management techniques but does not model directly endurance since a very large number of writes is necessary to damage a page when our techniques are modeled (e.g., a 10^7 write limitation, when multiplied by a few million pages would need almost 10^{13} memory requests). To measure lifetime, we used a separate fast behavioral simulator. A trace of writes to PCM from the PMMA simulator was input to the behavioral simulator. The trace can be repeated or mixed with other traces to get the required number of writes to damage a page.

Benchmarks with a large memory footprint and number of main memory accesses were chosen from PARSEC, SPECcpu2006 and SPECjbb2005. The benchmarks are: Canneal and Facesim from PARSEC; MCF, GCC, Bwaves and bzip2 from SPECcpu2006; and SPECjbb2005. A mix of SPECcpu2006 applications, composed of MCF, GCC, Bwaves and bzip2, were executed together to obtain a large memory footprint and utilization. Each benchmark was run in Simics for 2.5Billion requests to main memory. The collected memory traces were repetitively applied in the endurance management simulator until a page was damaged (10^7 writes to a bit on a page). The lifetime (years) was computed based on how many runs were necessary for the first page to die and the simulated time that each run takes.

A. Results

Figures 2, 3 and 4 show the impact of using the PCM-aware N-Chance policy instead of LRU. Figure 2 shows that the number of writes (and corresponding increase in lifetime of the PCM memory) always decreases as N is increased but the performance and energy gains (Figures 3, 4) do not show the same behavior. In this 14-way cache, a 7-Chance algorithm consistently gets most of the gains in the three metrics.

Figure 5 shows lifetime with RWR. The results show an increase in lifetime by a factor of 2 to 8, due to the write behavior of the applications. The pages that are highly written have only a small portion of each page modified. Because the smallest granularity for a write is a sub-page (256Bytes), any modified bit in the sub-page will cause the whole sub-page to be updated. As the figure shows, RWR is important to improve lifetime.

Figure 6 shows the lifetime achieved with different swap algorithms. A counter-per-page (CT) swap condition

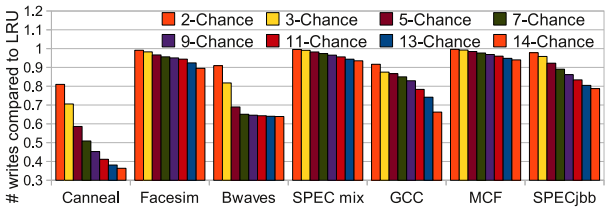

Fig. 2. N-Chance impact on #writes

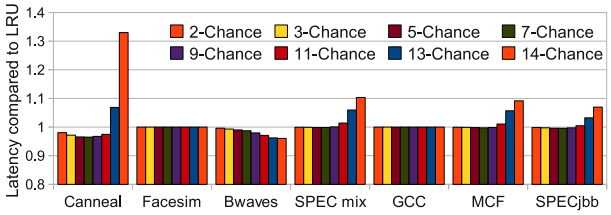

Fig. 3. N-Chance impact on latency

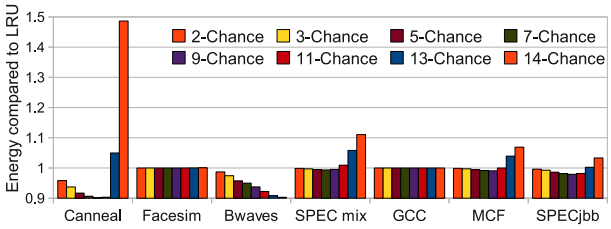

Fig. 4. N-Chance impact on energy

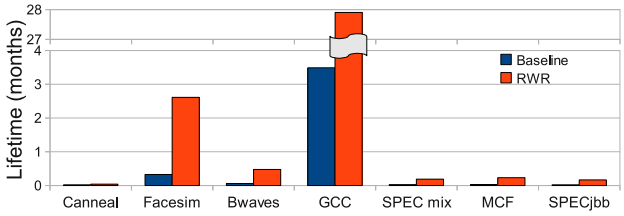

Fig. 5. Read-Write-Read impact on lifetime

with a LFW replacement selection is closest to the ideal WL (all pages have the same wear). We compare this algorithm to a global counter (GC) with LFW and random replacement selection algorithms. The lifetime was computed by running each benchmark 500 times and recording the number of writes per physical page. The number of runs necessary to damage each page (10^7 writes) is computed and the minimum overall is the predicted lifetime. The measured lifetime is a conservative estimate since it assumes that the distribution of writes per page will be the same for all subsequent runs.

The GC and random algorithms are expected to have better behavior for a larger number of writes, which will tend to make the distribution of writes per page more uniform as more runs are executed. The algorithms count each write to a sub-page as one write to PCM. CT256-LFW is an expensive algorithm that uses one counter per page, which searches to find the LFW page on each page swap. The other algorithms have less overhead. GC256-LFW reduces lifetime by 8% versus CT256-LFW because

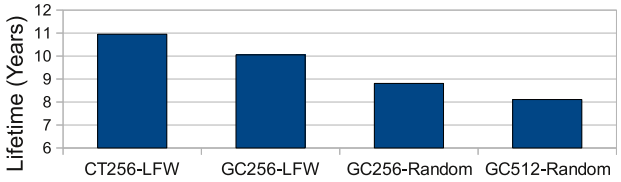

Fig. 6. Impact of wear-leveling on lifetime

it has a single counter to determine when to swap pages. It does not guarantee a swap when the number of writes to a *particular* physical page reaches a threshold. A random replacement algorithm can be used to avoid searching for the LFW page. GC256-Random uses a global counter with random replacement. It reduces lifetime by 12.4% in comparison to GC256-LFW and by 20% in comparison CT256-LFW. GC256-Random causes an additional 3% writes due to swapping. When the swap threshold is set to 512, GC512-Random has a 25.4% reduction in lifetime compared to GC256-LFW. In Figure 6 all algorithms have at least eight years of lifetime, but GC256-Random and GC512-Random have the lowest implementation cost and overhead.

Table I shows the cumulative impact of 7-Chance, RWR, and GC512-Random on lifetime as each one is successively applied. Without any technique, the PCM memory lasts only 15 days. 7-Chance nearly doubles the lifetime although the number of writes is reduced by only 17%. The impact of 7-Chance is large (83% of lifetime increase) because it eliminates writes from the most heavily written pages. RWR and swapping pages increases lifetime to 8.11 years (97.29 months). The swap algorithm is responsible for most of the lifetime; however, all techniques should be used together to achieve the best lifetime.

Technique	Lifetime	Cumulative Gain
Baseline (LRU)	0.47 month	
7-Chance	0.86 month	1.83
+ RWR	3.36 months	3.91
+ GC512-Random	97.29 months	28.91

TABLE I. Impact of each technique

B. Comparison with Other Techniques

The use of large 1MByte swapping pages, as in [5], has a negative impact on lifetime. In our experiments, CT256-LFW with 1MByte pages has a lifetime of 7.5 years. This value is 31% smaller than the lifetime obtained with 2KByte pages (CT256-LFW). GC512-Random gets a 6.3% better lifetime with significantly lower cost than swapping on 1MByte pages.

The RL algorithm is used in [3] for intra-page wear-leveling. Intra-page wear-leveling is needed when writes

are biased toward certain cache lines in a page. To understand how intra-page WL would work with our schemes, we measured the distribution of writes over the cache lines (64Byte line size). The distribution of writes is largely uniform, with some cache lines receiving 3.5% more writes. As a result, the increase in lifetime with intra-page WL would be at most 3.5%.

V. Related Work

PCM for main memory has been proposed by [4], [3], [5]. As discussed in Section I, the closest architecture to PMMA, [3] applies four techniques for endurance management: Lazy Writes, Line Level Write Back, Page Level Bypass and Fine-Grained WL. Lazy Writes avoid the first write to PCM for dirty pages; we achieve the same effect by restricting CPU writes to DRAM and only writing to PCM when a page is evicted. Our work also shows the significant impact of different victim selection policies on endurance, performance and energy consumption. Line Level Write Back tracks which lines in the page are dirty and writes only those lines to PCM when the DRAM page is evicted. We propose combining two techniques (page partitioning and RWR) to reduce wear and provide fault detection. Page Level Bypass avoids PCM writes for applications that have poor reuse in PCM, but requires changes to the OS. Fine-Grained WL helps distribute write traffic uniformly on all lines in the PCM page. However, this does not solve the problem of an unbalanced writes at the page level. In our experiments, the best balance on each page was achieved with a small swap size. [3] evaluates endurance based on the assumption that writes are uniformly distributed to the entire PCM memory. The GC256-Random algorithm is a feasible implementation of their ideal, high-level WL algorithm [3]. We evaluate PCM endurance impact by simulating the actual techniques to obtain much more accurate results.

In [4], PCM is proposed as main memory. Techniques, including redundant bit-write removal (RW), byte shifting at row level (BS) and segment swapping (SS), are proposed to improve the lifetime of PCM. RW only updates PCM bit cells with changed values instead of updating all bit cells on a page. This eliminates redundant bit-writes. It is implemented at the circuit level, requiring device changes. We implement page partitioning and sub-page RWR (similar to RW) in the memory controller. This obtains a reduction in memory bandwidth consumption since read bandwidth is much higher than write bandwidth [10]. We reduce PCM reads and writes by keeping recently accessed data in a writeback DRAM cache. BS and SS uniformly distribute PCM writes on bit and segment granularities. In contrast, we propose a more thorough set of write management schemes at the page level.

Zhou et al. describe a 3D main memory for PCM [5]. It uses read-before-write at the row level to avoid unnecessary writes. Row-level rotation and "segment swaps" are used for WL. The segment swap scheme is equivalent to CT-LFW with 1MByte pages. Our GC-Random algorithm avoids the search required in Zhou et al.'s scheme. It also permits a small page size. Overall, our schemes achieve a better lifetime (8 years with 10^7 writes per cell) with less complexity.

VI. Conclusion

PCM is a viable main memory technology but endurance is a weak point of PCM. We present a number of techniques to extend the lifetime of a PCM main memory to more than 8 years. Our writeback minimization cache replacement and a swap-based WL schemes are essential to achieve good lifetime. These techniques also have minimal overhead and low implementation complexity.

VII. Acknowledgments

This work was supported in part by the National Science Foundation through grants CCF-0811295, CCF-0811352, SES-0729456, CNS-0702236 and ANI-0325353.

References

[1] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem, "Power management in external memory using PA-CDRAM," in *The Int'l. Journal for Embedded Systems (IJES)*, vol. 3-1, 2007, pp. 65–72.

[2] W. Zhang and T. Li, "Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures," in *Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2009, pp. 101–112.

[3] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Int'l. Symp. on Computer Architecture (ISCA)*, 2009, pp. 24–33.

[4] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Int'l. Symp. on Computer Architecture (ISCA)*, 2009, pp. 2–13.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Int'l. Symp. on Computer Architecture (ISCA)*, 2009, pp. 14–23.

[6] S. Raoux et al., "Phase-change random access memory: A scalable technology," in *IBM Journal of Research and Development*, vol. 52, no. 4, 2008, pp. 465–479.

[7] S. Mylavarapu, S. Choudhuri, A. Shrivastava, J. Lee, and T. Givargis, "Fsaf: File system aware flash translation layer for nand flash memories," in *DATE '09.*, April 2009, pp. 399–404.

[8] K. Takeuchi, "Novel co-design of nand flash memory and nand flash controller circuits for sub-30 nm low-power high-speed solid-state drives (ssd)," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 4, pp. 1227–1234, April 2009.

[9] "Process integration, devices and structures," in *Int'l. Technology Roadmap for Semiconductors*, 2007.

[10] Lee, K. J. et al, "A 90 nm 1.8 v 512 mb diode-switch pram with 266 mb/s read throughput," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 150–162, Jan. 2008.

[11] Kang et al, "A 0.1 μm 1.8V 256Mb 66MHz Synchronous Burst PRAM," in *IEEE Int'l. Solid-State Circuits Conf. (ISSCC)*, 2006.