Write-Once-Memory-Code Phase Change Memory

Jiayin Li and Kartik Mohanram
Department of Electrical and Computer Engineering, University of Pittsburgh, PA
jil135@pitt.edu   kartik.mohanram@gmail.com

Abstract

This paper describes a write-once-memory-code phase change memory (WOM-code PCM) architecture for next-generation non-volatile memory applications. Specifically, we address the long latency of the write operation in PCM — attributed to PCM SET — by proposing a novel PCM memory architecture that integrates WOM-codes at the memory organization and memory controller levels. The proposed $<2^2>^2/3$ WOM-code PCM architecture is able to reduce memory write (read) latency by 20.1% (10.2%) on average across general-purpose (SPEC CPU2006), embedded (MiBench), and high-performance (SPLASH-2) benchmarks. To further improve the write latency of WOM-code PCM, we propose a PCM-refresh approach that uses idle cycles to preemptively set PCM rows to the initial WOM-code state. Results show that WOM-code PCM with PCM-refresh can reduce memory write (read) latency by 54.9% (47.9%) on average across the benchmarks. Finally, to balance write latency improvements against WOM-code PCM overhead, we propose a WOM-code cached PCM (WCPCM) architecture that uses WOM-code PCM as the cache alongside conventional PCM main memory. For just 4.7% memory overhead, WCPCM reduces memory write (read) latency by 47.2% (44.0%) on average across the benchmarks.

1. Introduction

The demands on memory performance and capacity in various domains are increasing rapidly [1]. For example, exascale computing services require large memory capacity that is as much as $1000\times$ the requirement of current "petaflop" class systems [2]. Therefore, next generation memory systems must be capable of supporting fast read and write accesses, while also offering the memory capacity necessary to maintain large data structures. DRAM has played a major role in supporting the demands on memory capacity and performance for decades. However, scaling DRAM below 22nm is currently unknown [3], which makes DRAM less suitable for next generation main memory in the "big data" era.

Of the next-generation non-volatile memory candidates, phase-change memory (PCM), which offers better read latency than resistive RAM (ReRAM) and higher cell density than spin-transfer-torque RAM (STT-RAM) [4], is a promising candidate to fill this scalability gap. Unfortunately, PCM is an asymmetrical read-write technology with a write latency that is much longer than that of DRAM. The write process in PCM requires strict duration and strength of the programming current. Writing "0" to a PCM cell, i.e., the RESET operation, uses a short but high current to program the phase change material to the amorphous state. However, writing "1", i.e., the SET operation, utilizes a 5–10$\times$ longer but lower current to program the phase change material to the polycrystalline state. When a page of data is written to PCM, the write latency is determined by the long SET operation, which is usually 5–10$\times$ the read latency [1, 5, 6]. In general-purpose applications, an extensive study has already shown that the long write latency in PCM may result in as much as 61% performance degradation [7]. Thus, the long write latency inherent to PCM remains the biggest challenge that has to be overcome in order to realize scalable PCM memory for next generation memory architectures.

Scaling the PCM cell [8] and new phase-change material and cell designs [9, 10] have been proposed to decrease PCM write latency. Given current technology and existing phase change materials, most write-latency-aware PCM studies have focused on write scheduling [7, 11–13] and architecture-based write improvement [14–17] solutions. Write scheduling solutions primarily attempt to distribute writes among idle bank cycles [7, 11–13]; however, this is not suitable for high-performance computing where there are little-to-no idle cycles between memory accesses. On the other hand, architecture-based write improvement solutions rely on technology-specific features such as multi-level cells and PCM division write mode [14, 15] to reduce write latency. As a result, neither approach is general enough to address the limitation of long writes in PCM. Latency-aware coding schemes were proposed to improve PCM write latency [16, 17]. However, these approaches need to SET a minimum number of PCM bits in each write operation, limiting improvements in write latency. Note that other write-related issues such as power consumption, cell endurance, and error detection/correction have been addressed through design [18–22] and coding [23–26] approaches in literature.

This paper describes a novel write-once-memory-code phase change memory (WOM-code PCM) architecture that integrates WOM-codes at the memory organization and memory controller levels to reduce the write latency of PCM. First, we address the long latency of the write operation — attributed to PCM SET — by using an "inverted" WOM-code that transforms PCM writes to comprise of low latency RESET operations. We propose two memory organization methods to provision the extra memory space for the WOM-code encoded data: the wide-column method and the hidden-page method. In practice, the wide-column approach is suitable for fixed WOM-code PCM architectures that provide minimum memory controller support and demand better performance; the hidden-page approach is suitable for PCM memory with dynamic WOM-code capabilities, with the memory controller choosing between codes for flexible memory utilization and dynamic performance tradeoffs. Second, we propose a PCM-refresh approach to further improve the performance of WOM-code PCM. Our analysis shows that the write that occurs after the WOM-code reaches its rewrite limit is the bottleneck that gates the performance of WOM-code PCM. PCM-refresh periodically and opportunistically "refreshes" a PCM page that has reached the rewrite limit in idle cycles, improving the performance of WOM-code PCM. Finally, we design a WOM-code cached PCM (WCPCM) architecture that uses a small amount of WOM-code PCM array as the cache (WOM-cache), on top of the large conventional PCM main memory array. The WOM-cache increases memory access speed by exploiting locality in the memory access stream, requiring less memory capacity in comparison to implementing WOM-code across the entire PCM array. To the best of our knowledge, this is the first paper proposing a memory architecture that integrates WOM-codes with PCM to successfully mitigate the long write latency issue in PCM (attributed to the time-consuming SET operation).

We extended the DRAMSim2 [27] memory simulator to evaluate the WOM-code PCM architecture across a broad set of general-

This research was supported by awards CCF-1217738 and CCF-1208933 from the National Science Foundation.
978-3-9815370-2-4/DATE14/©2014 EDAA

purpose (SPEC CPU2006 [28]), embedded (MiBench [29]), and high-performance (SPLASH-2 [30]) computing benchmarks. Results show that the simple $<2^2>^2/3$ WOM-code PCM architecture can reduce memory write (read) latency by 20.1% (10.2%) on average across the benchmarks. When PCM-refresh is integrated with WOM-code PCM, the memory write (read) latency is reduced by 54.9% (47.9%) on average across the benchmarks. Finally, for just 4.7% memory overhead, WCPCM reduces memory write (read) latency by 47.2% (44.0%) on average across the benchmarks, providing a practical low overhead memory architecture solution to address the write latency problem in PCM.

2. Background and Motivation

2.1 Phase change memory

PCM, an emerging non-volatile memory, is a one-transistor-one-resistor (1T1R) or one-diode-one-resistor (1D1R) memory device technology. PCM stores data by switching chalcogenide material, such as $Ge_2Sb_2Te_5$ (GST), between amorphous and polycrystalline states. These two states are characterized by remarkably different resistance levels; amorphous chalcogenide material has high resistance, usually in the MΩ range, wheras polycrystalline chalcogenide material has low resistance, usually in the KΩ range [17].

There are three primary operations integral to the use of PCM in a modern memory system: read, SET, and RESET. The read operation loads the data from the memory to the processor or the cache hierarchy. The SET (RESET) operation writes the bit "1" ("0") to the memory cell, i.e., the SET (RESET) operation changes the state of the chalcogenide material in the cell to polycrystalline (amorphous). Each of the three operations has its own associated latency and this is discussed in the following paragraphs.

A PCM cell can be read by simply sensing the current flow. Due to the large gap between the two resistance levels of the chalcogenide material, the sensed current of these two states differ by three or more orders of magnitude. The latency of the read operation in PCM cells is typically tens of nanoseconds.

In the write operation, the programming circuit applies different heat-time profiles to switch cells from one state to the other. To RESET a PCM cell, a strong programming current pulse of short duration is required. The temperature of the chalcogenide material is raised by this programming pulse. After the chalcogenide material reaches the melting point, the programming pulse is quickly terminated. Subsequently, the small region of melted material cools quickly, leaving the chalcogenide material programmed in the amorphous state. Since the region of the melted chalcogenide material is small, the required duration of the RESET programming pulse is short, about tens of nanoseconds. Thus, the RESET latency of PCM is typically comparable to its read latency [11].

In contrast, to SET a PCM device, a long programming current pulse, which is weaker than the RESET programming current, is applied to program the cell from the amorphous state to the polycrystalline state. In the SET operation, the temperature of chalcogenide material should be raised above its crystallization temperature but below the melting point for a sufficient amount of time. As the crystallization rate is a function of temperature, given the variability of PCM cells within an array, reliable crystallization requires a programming pulse of hundreds of nanoseconds [11]. Therefore, the SET latency of a PCM cell is longer than both the RESET latency and the read latency, usually 5–10$\times$ the read latency [5, 6].

From a system perspective, in a typical PCM write, hundreds of bits in a memory line need to be programmed in PCM cells. It is highly likely that both RESET and SET operations occur in such a write. Thus, the write latency is determined by the slower of the two operations, which is the SET operation. Therefore, the write latency in PCM memory is longer than the read latency.

2.2 Write-once-memory

Memory technologies whose storage cells transit irreversibly between states, e.g., punch cards and optical discs, have been common since the beginning of the data storage technology. Most of these technologies can change the state "0" to state "1", but not vice versa. The "write-once" nature largely limits the application space of such memory technologies. To overcome the write-once nature and explore the true capabilities of such write-once memories (WOMs), Rivest and Shamir proposed a model to efficiently reuse WOMs [31]. They showed that for a fixed k, it is possible to allow t writes of k-bit data using only $t + O(t)$ bits of WOM memory.

In the WOM model, a WOM is an array of "write-once bits", i.e., *wits*, which are manufactured or predefined in a "0" state. Each wit can be transformed into the "1" state independently but irreversibly. A WOM-code is a coding scheme that is able to write the WOM multiple times. A WOM-code can be defined as follows: a "$<\mathrm{v}>^t/n$ WOM-code" is a coding scheme that uses n wits to represent one of v values so that the WOM can be written a total of t times. We use a simple $<2^2>^2/3$ WOM-code as a typical example to explain the basic idea of rewriting wits using Table 1. In the first write, any 2-bit value x is represented with the pattern $r(x)$ given in Table 1. Then, when another 2-bit value y ($x \neq y$) is written into the same 3-wit WOM, it is represented with pattern $r'(y)$. Observe that for any 2-bit value, the representation in the second write only changes wits from state "0" to state "1". For decoding this WOM-code, the pattern $r(x) =$ "abc" written in WOM can be decoded to 2-bit data $x =$ "uv" as ($u = b \bigoplus c$) and ($v = a \bigoplus c$). With this WOM-code, a set of 3-wit WOM can write 2-bit data twice.

Table 1: $<2^2>^2/3$ WOM-code [31]

Data $x =$ "uv"	First write pattern $r(x) =$ "abc"	Second write pattern $r'(x) =$ "$a'b'c'$"
00	000	111
01	100	011
10	010	101
11	001	110

Most recently, flash memories based on floating-gate cells have become a very promising storage technology, due to their high data density, fast read speed, and physical robustness. However, the programming mechanism in flash memories causes problems similar to that of WOMs. The SET operation in flash is relatively simple and fast, realized by injecting charge into the flash cell. But removing the charge, i.e., the RESET operation, is difficult and requires the removal of the entire charge from a large block of cells, i.e., block erasure, which is very time consuming. This programming mechanism makes it easier to change a bit from "0" to "1" (versus a change from "1" to "0") in flash memories. WOM-code schemes has been studied to achieve faster write speed in flash memories [32, 33]. With WOM-codes, the block of bits in flash can be rewritten multiple times without a RESET operation.

The asymmetric write latency problem in PCM is analogous to that of flash. Even though the PCM cell is able to change from "0" to "1", the performance cost of doing so is 5–10$\times$ higher than the cost of changing from "1" to "0". Using an inverted WOM-code in the PCM cell so that the rewrite to the PCM cell consists of only low latency RESET operations can reduce the write latency in PCM. Since various WOM coding schemes for flash and WOMs have been studied intensively in the literature, this paper focuses exclusively on the implementation challenges of WOM-codes in PCM at the architecture level; the WOM-codes discussed here and other existing WOM-codes can be integrated into the proposed WOM-code PCM framework. Note that the only work applying WOM-codes to PCM focuses on energy reduction [34] and is not directly related to the work described in this paper.

3. WOM-code PCM and PCM-refresh

In this section, we first discuss the implementation of WOM-code PCM and its benefits. We then describe PCM-refresh, which achieves faster write performance in WOM-code PCM.

3.1 WOM-code in PCM

In the conventional WOM-code, wits are all preset to the "0" state. In each write, wits can only be programmed from "0" to "1", i.e., the SET operation. However, in PCM, the SET operation has longer latency than the RESET operation. Thus, the basic principle in WOM-code PCM is that wits in PCM can only be programmed from "1" to "0". There are two approaches to apply the conventional WOM-code to PCM: the first is to connect an inverter to each bitline of the sense amplifier/write driver in the PCM array, as shown in Fig. 1(a); the other is to use an inverted WOM-code. An example of the inverted WOM-code is shown in Fig. 1(b). In each of rewrites in this WOM-code, wits either stay unchanged or are programed from state "1" to state "0". Both methods are compatible with any existing WOM-code. Since the inverted WOM-code can be generated off-line in advance, the runtime complexity of these methods are identical. However, the inverted WOM-code method is preferable in practice, since it does not require any additional inverters within the row buffer.

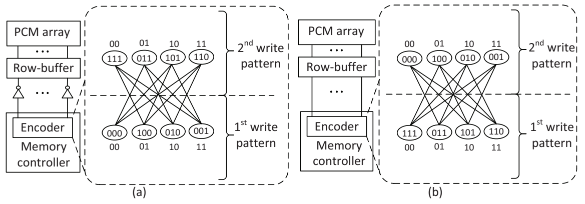


Figure 1: (a)WOM-code with inverters (b)Inverted WOM-code

Compared to the original data pattern, WOM-code utilizes extra bits to support multiple writes. For example, to encode the data of a page (e.g., 4KB) with the $<2^2>^2/3$ WOM-code, the memory controller should be able to manage a $0.5\times$ increase in page size (i.e., to 6KB) for every rewrite. We propose two approaches to adapt the memory architecture: the wide-column WOM-code PCM architecture and the hidden-page WOM-code PCM architecture.

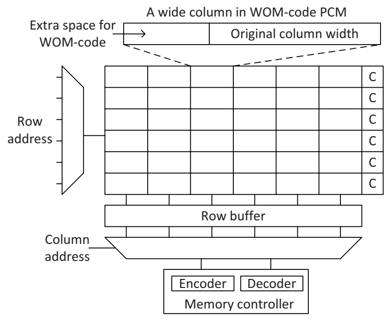


Figure 2: Wide-column WOM-code PCM architecture

Wide-column WOM-code PCM architecture (Fig. 2): In the conventional memory organization, memory cells are organized in a memory array with X rows, Y columns per row, and Z bits per column. In a memory architecture with data width D bits (e.g., $D = 64$), D/Z devices are organized together for memory access in a rank. In a memory access, the memory address is decoded as the row address and the column address. The row select signal is enabled based on the row address, resulting in the connection of the row buffer (a buffer of size YZ bits) to the selected row. The column select signal is also enabled based on the column address, and the given column of data (of width Z bits) is either output to the data-out buffer or input from the data-in buffer. To accommodate WOM-code-encoded data in the memory array, the width of a column, i.e., Z, should be increased. Given that the $<2^2>^2/3$ WOM-code is used, the width of a column should be increased to $1.5Z$. Accordingly, the total number of bits in a row should be increased to $1.5YZ$. Similar modifications are needed for the row buffer and the data-in/data-out buffers. In this memory architecture, memory data is encoded in the unit of a column. The encoded column data is stored in $1.5Z$ consecutive bits in the memory array.

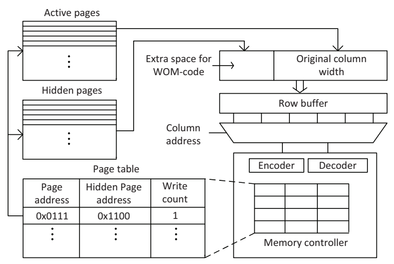


Figure 3: Hidden-page WOM-code PCM architecture

Hidden-page WOM-code PCM architecture (Fig. 3): In the hidden-page WOM-code PCM architecture, the memory controller manages hidden memory pages to provide the extra memory for the encoded data. The memory controller reserves a defined range of memory pages as hidden pages and makes them unavailable to the operating system. Again, consider the $<2^2>^2/3$ WOM-code as an example. To store the encoded data of a row (i.e., YZ bits), $1.5\times$ the original memory capacity is needed. The lower YZ bits are stored in the original page, while the upper $0.5YZ$ bits are stored in a hidden page reserved by the memory controller. Since the entire row of data is accessed by the row buffer in both write and read operations, the WOM-code can encode the data at the row-level.

Comparison — wide-column versus hidden-page: Both approaches presented above are able to provide the extra memory space for WOM-code-encoded data. The wide-column approach requires a simpler but faster memory controller, since its memory controller does not need to reserve pages for encoding. In contrast, the memory controller in the hidden-page approach also needs to manage the hidden pages, recruit unused pages, release pages when necessary, and maintain a page table. On the other hand, the hidden-page approach has its advantage in flexibility. Since the extra memory space required by WOM-code is provided in the hidden pages, the memory requirements of different WOM-codes can be satisfied dynamically by the memory controller. In contrast, the wide-column approach is strictly fixed in terms of the extra memory reserved for the WOM-code. Using the $<2^2>^2/3$ WOM-code as an example, each column has $1.5Z$ bits, which means it cannot accommodate any WOM-code with more than 50% memory overhead. In practice, the wide-column approach is suitable for fixed WOM-code architectures that provide minimum memory controller support and demand better performance; the hidden-page approach is suitable for PCM memories that require dynamic WOM-code configuration, with the memory controller choosing between codes for flexible memory utilization and dynamic performance tradeoffs.

3.2 PCM-refresh

In the WOM-code PCM architecture presented above, wits are programmed based on the WOM-code scheme before they reach the rewrite limit, resulting in PCM write latency that is as short as PCM RESET latency. However, once wits of a given page reach the rewrite limit, the next write to this page follows the pattern of the first write in this scheme, which requires both the SET and the RESET operation. Thus, the write latency of the WOM-code PCM on the first write pattern is still as long as the write latency in conventional PCM.

Assume that a k-rewrite WOM-code scheme is used in PCM with a RESET latency of L and a SET latency of SL, where $S \geq 1$ is the slowdown factor. The PCM wit can only be rewritten k times. Thus the total write latency of any k consecutive writes in WOM-code PCM is $(k-1)L + SL$. For PCM without the WOM-code, the write latency of any k consecutive writes is kSL. The performance improvement of k-rewrite WOM-code PCM is thus bounded by a factor of $(k-1+S)/(kS)$. Obviously, a higher limit on the number of rewrites increases this upper bound. However, a WOM-code with a higher limit on the number of rewrites imposes a larger memory overhead.

Another way to increase the bound on performance improvement in WOM-code PCM is to reduce the latency of the first write after the rewrite limit is reached (termed the α-write in this paper). We propose PCM-refresh that uses idle cycles to hide the long latency of the α-write by periodically and opportunistically refreshing the pages that are at the rewrite limit. PCM-refresh is inspired by the DRAM refresh operation, which is necessary for data readout and restoration in DRAM. In PCM-refresh, the PCM controller sends a PCM-refresh command to "refresh" a page in all banks in a target rank. The PCM controller periodically checks the status of all ranks in the PCM memory, and picks a target rank from the pool of idle ranks in round-robin fashion. The target row address of a given bank is given by a row address table that is implemented in the PCM controller. Each entry of this row address table is a row address buffer of a bank, which records the most recent 5 pages that have reached the rewrite limit. After the PCM-refresh command is issued, row addresses are sent to all banks in the target rank, and the data is read out and written back into the row following the first write pattern of the WOM-code. The entire PCM-refresh operation is handled in burst mode. Hence, the latency of PCM-refresh is $t_{\mathrm{WR}} + (N_{\mathrm{bank}} * L_{\mathrm{burst}}/2)$, where t_{WR} is the latency of the normal write in PCM, N_{bank} is the number of banks in a rank, and L_{burst} is the burst length. Note that $L_{\mathrm{burst}}/2$ is the data burst duration in the DDR3 standard. Since L_{burst} is much smaller than t_{WR}, batch processing PCM-refresh operations of multiple banks in burst mode can reduce the performance impact of PCM-refresh. To further reduce the impact of PCM-refresh on memory access blocking, we combine write pausing [7] with PCM-refresh. Write-pausing prioritises writes and reads that access banks undergoing a PCM-refresh by enabling them to preempt the ongoing PCM-refresh operation. PCM-refresh has limited impact on energy consumption. The energy consumption of PCM-refresh is equal to the energy consumption of a single row read followed by a single row write.

The "refreshed" PCM row can be immediately written by the pattern of the second write in the WOM-code. Ideally, if all memory accesses are perfectly distributed over time and memory ranks, PCM-refresh can hide the latency of the α-write in the idle cycles, and achieve a performance improvement of $S\times$, which is not limited by the rewrite limit of the WOM-code. To further improve the efficiency of PCM-refresh in practice, we introduce a refresh threshold parameter $r_{\mathrm{th}}\%$. When the PCM controller selects the target ranks from the idle ranks, only the ranks where more than $r_{\mathrm{th}}\%$ of the banks have at least one page that has reached the rewrite limit are selected for PCM-refresh.

4. WOM-code Cached PCM

Whereas WOM-code PCM can significantly reduce the write latency, it comes at the cost of memory capacity. For example, the $<2^2>^2/3$ WOM-code PCM uses 50% memory cell overhead to store the extra bits of encoded data. To enable tradeoffs between performance and memory overhead, we propose a WOM-code cached PCM (WCPCM) architecture that integrates a small WOM-code PCM array as the cache (WOM-cache) on top of the large conventional PCM main memory array, as shown in Fig. 4.

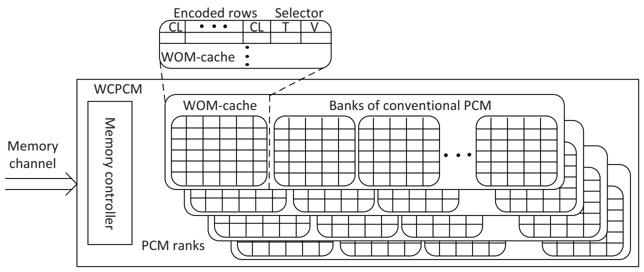


Figure 4: WOM-code Cached PCM (WCPCM)

WCPCM is implemented at the rank level, where each rank has its own WOM-cache array (wide-column design with PCM-refresh) with the same number of rows and columns as a conventional PCM array in a bank of the PCM main memory. Besides the wide-column in each row, there is a selector field associated with each row that includes T, the tag of the row, and V, the valid bit of a page. For a PCM memory architecture with N_{bank} banks per rank, the WOM-cache is an N_{bank}-way associative cache, with tag width T equal to $\log(N_{\mathrm{bank}})$. The valid bit of V is only one bit wide. Thus, for a memory architecture with 32 banks per rank, the width of the selector field is only 6 bits per row.

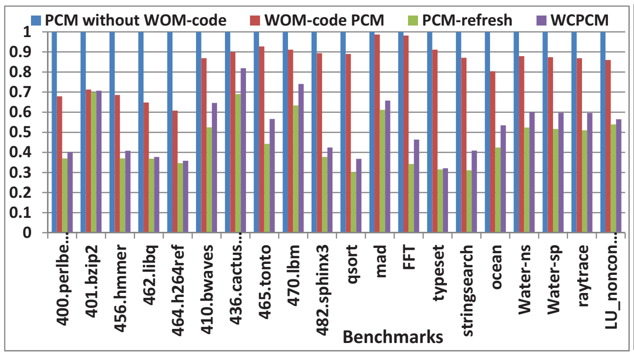
Write protocol in WCPCM: Since the objective of the WCPCM design is to provide fast write accesses in PCM, the WOM-cache only caches the write access. When a write access is issued to the corresponding rank of the PCM, the memory controller first selects the target row in the WOM-cache. After selecting the row, the controller checks the WOM-cache tag and the valid bit. In the case of a WOM-cache hit, where the valid bit is invalid, or the WOM-cache tag matches the bank address of this write access, the memory controller forwards the data in the row buffer and programs the cells accordingly. In the case of a WOM-cache miss, where the valid bit is valid and the WOM-cache tag does not match the bank address of this write access, the memory controller first outputs the current data and the bank address, i.e., the victim data, to a register. Then the controller programs the cells and updates the WOM-cache tag according to the new data. Finally, the write request of the victim data in the register is inserted into the queue of memory accesses that is issued to the PCM main memory.

Read protocol in WCPCM: In a read access, both accesses to the corresponding WOM-cache entry and the physical address in the PCM main memory are issued in parallel. In the case of a WOM-cache hit, the data in the PCM WOM-cache is forwarded to the output buffer. Otherwise, the data in the PCM main memory is forwarded to the output buffer. Note that regardless of a WOM-cache hit or miss, the content in the corresponding WOM-cache entry will not be changed in the read access.

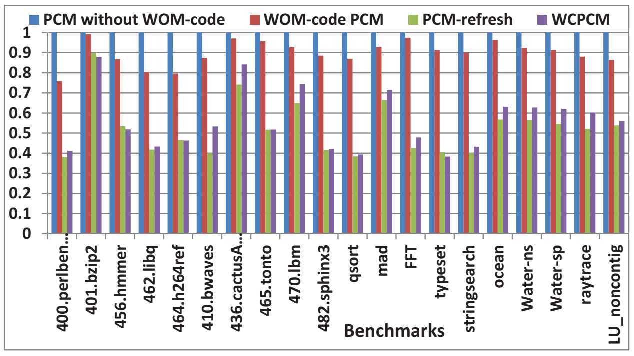
The WCPCM architecture has the following advantages:

- **Exploiting WOM-cache locality:** The locality in the on-chip last level cache (LLC) miss stream is very high [35]. The large entry size of the WOM-cache (e.g., the size of a row), and the high locality in the incoming access stream to the main memory enable the effective utilization of the short write access latency of the WOM-cache and the speedup of the whole PCM memory system.

- **Reducing WOM-code overhead:** Instead of provisioning at least 50% spare memory capacity when integrating WOM-codes into the PCM memory system, WCPCM only applies WOM-codes to a portion of the memory space. The number of addressable units in a WOM-cache is the same as that of a bank. Given that 50% memory capacity in each addressable unit is required by WOM-codes, the memory overhead of the



(a)



(b)

Figure 5: (a) Normalized write latency in PCM main memory (b) Normalized read latency in PCM main memory

entire WOM-cache is only $1.5/32 = 4.7\%$ in a PCM architecture with 32 banks per rank. For memory architectures with more banks per rank, the memory overhead of WOM-cache is even lower.

- **Increasing memory access speed:** In the case of a WOM-cache hit, the latency of a write is reduced to the latency of the RESET operation. On the other hand, in the case of a WOM-cache miss, the penalty of using WOM-cache is one to two clock cycles for the tag comparison. Given that the write latency is of the order of hundreds of cycles, the penalty of using WOM-cache can be ignored. Similar to the WOM-cache miss, a read access in the WCPCM only requires the overhead of the tag comparison, since the accesses to the WOM-cache and the PCM main memory are performed in parallel.

- **Practical cached memory solution:** Whereas the functionality of WCPCM is similar to that of hybrid DRAM/PCM [18], WCPCM is more practical since it uses only PCM cells and is hence easier to fabricate. Hybrid DRAM/PCM approaches not only need to integrate the fabrication of DRAM and PCM, but also face scalability issues inherent to DRAM. Note that the proposed WOM-cache can also be implemented as high-density on-chip cache.

5. Simulation setup and results

The proposed PCM architecture was evaluated using trace driven simulations. We use Pin [36] to capture the memory access trace on a machine with an Intel Core i7 CPU 980 running at 3.33GHz. We collect memory access traces from a set of workloads: integer benchmarks (400.perlbench, 401.bzip2, 456.hmmer, 462.libq, 464.h264ref) and floating point benchmarks (410.bwaves, 436.cactusADM, 465.tonto, 470.lbm, and 482.sphinx3) from SPEC CPU-2006 [28] in the general-purpose computing domain; qsort, mad, FFT, typeset, and stringsearch from MiBench [29] in the embedded computing domain; and ocean, water-ns, water-sp, raytrace, and LU-non-contiguous-block from SPLASH-2 [30] in the high-performance computing domain.

For accurate simulation of memory access latencies in the proposed PCM architecture, the simulation framework was configured based on accepted parameters and standards for memory modeling obtained from manufacturer data sheets. The PCM simulator is based on DRAMSim2 [27] and follows standard JEDEC protocols for DDR3 memory, since the PCM access protocol will not differ much from DRAM [37, 38]. The fundamental modifications in DRAMSim2 for PCM simulation are [38]: row read delay is 27ns, row write delay is 150ns, the RESET latency is 40ns, the SET latency is 150ns, and the PCM-refresh period is 4000ns. We used the PCM main memory architecture proposed in [37] as the baseline PCM memory architecture. The PCM device was configured as follows: the 16GB PCM main memory architecture is organized as single channel with 16 ranks with 32 banks/rank. Each PCM device has 32768 rows, 2048 columns/row, and 4 bits of data/column; thus, 16 devices are used in parallel to form the 64-bit data width of main memory.

In Fig. 5(a), we report the normalized average write latency of four PCM architectures: PCM without WOM-code (blue), WOM-code PCM (red), PCM-refresh, i.e., WOM-code PCM with PCM-refresh (green), and WCPCM (purple). PCM without WOM-code is the conventional PCM that serves as the baseline architecture. The write latencies of four PCM architectures are normalized to that of PCM without WOM-code. We observe that the write latency of PCM is reduced by 20.1% on average in WOM-code PCM. The best improvement occurs in the 464.h264ref benchmark, where the write latency is reduced by 39.2%. PCM-refresh further reduces the write latency, by 54.9% on average across the benchmarks. The best improvement with PCM-refresh is in the 464.h264ref benchmark, where it achieves 65.3% lower write latency. We also observe that the performance of WCPCM, in terms of write latency, is between PCM-refresh and WOM-code PCM. WCPCM achieves 47.2% lower write latency on average, in comparison to conventional PCM.

We also evaluate the average read latency of the four PCM architectures, as shown in Fig. 5(b). The read latency is normalized to that of PCM without WOM-code, similar to Fig. 5(a). As the longer write latency in PCM is likely to block read accesses to the same bank, we find significant improvement in the read latency of our proposed PCM architectures, compared to conventional PCM. It is consistent with the improvement of write latency shown in Fig. 5(a), even though the degree of improvement is smaller. WOM-code PCM can reduce the read latency by 10.2% on average. The read latency with PCM-refresh is the lowest among four architectures, analogous to the results for write latency. An average of 47.9% lower read latency is observed with PCM-refresh. WCPCM has 44.0% lower read access latency than conventional PCM, which is still better than WOM-code PCM.

We also evaluate the sensitivity of the tag design in the WCPCM. In Fig. 6, we measure the WOM-cache hit rate in WCPCM with four PCM organizations: 4 banks/rank (blue), 8 banks/rank (red), 16 banks/rank (green), and 32 banks/rank (purple). It is worth noting that the tag in the WOM-cache is the bank address. Changing the number of banks/rank impacts the associativity of WOM-cache. The more banks/rank, the lower the hit rate in the WOM-cache. However, increasing the number of banks/rank provides better parallelism in memory accesses, which helps in reducing access conflicts. In Fig. 7, we report the normalized write latency in WCPCM with four PCM organizations, similar to Fig. 6. The write latency

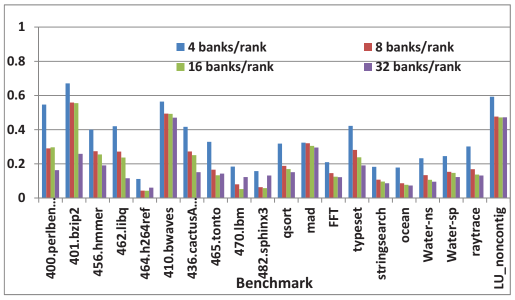


Figure 6: WOM-cache hit rate in WCPCM

is normalized to that of the 4 banks/rank organization. We observe that the write latency in WCPCM decreases as the number of banks/rank increases.

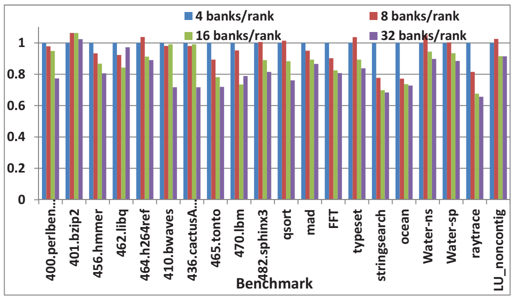


Figure 7: Normalized write latency in WCPCM

6. Conclusions

In this paper, we proposed a WOM-code PCM architecture for next-generation non-volatile memory applications. Two memory organization methods — wide-column and hidden-page — were proposed to provide the memory capacity required by WOM-codes. A PCM-refresh policy was proposed to increase the write performance of WOM-code PCM by periodically refreshing PCM rows that have reached the WOM-code rewrite limit. Finally, to balance memory overhead and write performance improvements in WOM-code PCM, we proposed a WOM-code cached PCM (WCPCM) architecture that uses WOM-code PCM as the cache alongside a conventional PCM main memory array. Note that the proposed WOM-code PCM architectures focus on reducing PCM write latency; their impact on the endurance of PCM is not explicitly addressed in this paper, and the problem remains open for future research. Evaluation on traces from multiple benchmarks using the modified DRAMSim2 simulator show that the WOM-code PCM architecture can reduce memory write (read) latency by 20.1% (10.2%) on average. When PCM-refresh is integrated with WOM-code PCM, the memory write (read) latency is reduced by 54.9% (47.9%) on average across the benchmarks. Finally, for just 4.7% memory overhead, WCPCM reduces memory write (read) latency by 47.2% (44.0%) on average across the benchmarks, providing a practical low overhead memory architecture solution to address the write latency problem in PCM.

References

[1] H. Li and Y. Chen, *Nonvolatile Memory Design: Magnetic, Resistive, and Phase Changing*. CRC Press, 2011.

[2] P. Kogge, K. Bergman, S. Borkar, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," tech. rep., DARPA, 2008.

[3] "International Technology Roadmap for Semiconductors," 2011.

[4] Y. Xie, "Future memory and interconnect technologies," in *Proc. Design Automation and Test in Europe*, 2013.

[5] S. Kang *et al.*, "A 0.1-μm 1.8-V 256-Mb phase-change random access memory (PRAM) with 66-mhz sysnchronous burst-read operation," *IEEE Journal of Solid-state Circuits*, vol. 42, no. 1, pp. 210–218, 2007.

[6] I. Song *et al.*, "A 20nm 1.8V 8GB PRAM with 40MB/s program bandwidth," in *Proc. Intl. Solid-state Circuits Conference*, 2012.

[7] M. K. Qureshi *et al.*, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. Intl. Symposium on High-performance Computer Architecture*, 2010.

[8] W. Wang *et al.*, "Fast phase transitions induced by picosecond electrical pulses on phase change memory cells," *Applied Physics Letters*, vol. 93, no. 4, pp. 043121– 043121–3, 2008.

[9] C. Lee *et al.*, "Performances of GeSnSbTe material for high-speed phase change memory," in *Proc. Intl. Symposium on VLSI Technology, Systems and Applications*, 2007.

[10] S. Lee *et al.*, "Low power and high speed phase-change memory devices with silicon-germanium heating layers," *Journal of Vacuum Science and Technology B*, vol. 25, no. 4, pp. 1244–1248, 2007.

[11] M. K. Qureshi *et al.*, "Preset: Improving performance of phase change memories by exploiting asymmetry in write time," in *Proc. Intl. Symposium on Computer Architecture*, 2012.

[12] Y. Kim, S. Yoo, and S. Lee, "Write performance improvement by hiding R drift latency in phase change RAM," in *Proc. Design Automation Conference*, 2012.

[13] S. Kwon *et al.*, "Optimizing video application design for phase-change RAM-based main memory," *IEEE Trans. VLSI Systems*, vol. 20, no. 11, 2012.

[14] L. Jiang *et al.*, "Improving write operations in MLC phase change memory," in *Proc. Intl. Symposium on High-performance Computer Architecture*, 2012.

[15] J. Yue and Y. Zhu, "Making write less blocking for read accesses in phase change memory," in *Proc. Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2009.

[16] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[17] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. Intl. Symposium on High-performance Computer Architecture*, 2013.

[18] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: a hybrid PRAM and DRAM main memory system," in *Proc. Design Automation Conference*, 2009.

[19] H. Park, S. Yoo, and S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," in *Proc. Design Automation Conference*, 2009.

[20] J. Dong *et al.*, "Wear rate leveling: lifetime enhancement of PRAM with endurance variation," in *Proc. Design Automation Conference*, 2011.

[21] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. Intl. Conference on Computer Design*, 2011.

[22] G. Sun *et al.*, "A frequent-value based pram memory architecture," in *Proc. Asia and South Pacific Design Automation Conference*, 2011.

[23] A. Mirhoseini *et al.*, "Coding-based energy minimization for phase change memory," in *Proc. Design Automation Conference*, 2012.

[24] A. Jiang *et al.*, "Constrained codes for phase-change memories," in *Information Theory Workshop*, 2010.

[25] M. Qin *et al.*, "Time-space cpmstraomed codes for phase-change memories," *IEEE Transactions on Information Theory*, vol. 59, no. 8, pp. 5102–5114, 2013.

[26] W. Xu and T. Zhang, "A time-aware fault tolerance scheme to improve reliability of multilevel phase-change memory in the presence of significant resistance drift," *IEEE Trans. VLSI Systems*, vol. 19, no. 8, pp. 1357–1367, 2011.

[27] P. Rosenfeld *et al.*, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[28] "SPEC CPU2006," 2006.

[29] M. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *Workshop on Workload Characterization*, 2001.

[30] S. C. Woo *et al.*, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Intl. Symposium on Computer Architecture*, 1995.

[31] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Information and Control*, vol. 55, pp. 1–19, 1982.

[32] A. Jiang *et al.*, "Rank modulation for flash memories," *IEEE Trans. Information Theory*, vol. 55, no. 6, pp. 2659–2673, 2009.

[33] A. Jiang *et al.*, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Trans. Information Theory*, vol. 56, no. 5, pp. 2112–2120, 2010.

[34] X. Zhang *et al.*, "WoM-SET: Lowering write power of proactive-SET based PCM write strategy using WoM code," in *Proc. Intl. Symposium on Low Power Electronics and Design*, 2013.

[35] Z. Zhang *et al.*, "Design and optimization of large size and low overhead off-chip caches," *IEEE Trans. Computers*, vol. 53, no. 7, pp. 843–855, 2004.

[36] V. Reddi *et al.*, "Pin: a binary instrumentation tool for computer architecture research and education," in *Workshop on Computer Architecture Education*, 2004.

[37] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[38] R. A. Bheda *et al.*, "Energy efficient phase change memory based main memory for future high performance systems," in *Proc. Intl. Green Computing Conference and Workshop*, 2011.