Techniques for Data Mapping and Buffering to Exploit Asymmetry in Multi-Level Cell (Phase Change) Memory

HanBin Yoon
hanbinyoon@cmu.edu

Naveen Muralimanohar†
naveen.muralimanohar@hp.com

Justin Meza
meza@cmu.edu

Onur Mutlu
onur@cmu.edu

Norman P. Jouppi†
norm.jouppi@hp.com

Computer Architecture Lab (CALCM)
Carnegie Mellon University

†HP Labs

SAFARI Technical Report No. 2013-002

May 16, 2013

Abstract

Phase Change Memory (PCM) is a promising alternative to DRAM to achieve high memory capacity at low cost per bit. Adding to its better projected scalability, PCM can also store multiple bits per cell (called multi-level cell, MLC), offering higher bit density. However, MLC requires precise sensing and control of PCM cell resistance, which incur higher memory access latency and energy.

We propose a new approach to mapping and buffering data in MLC PCM to improve memory system performance and energy efficiency. The latency and energy to read or write to MLC PCM varies depending on the resistance state of the multi-level cell, such that one bit in a multi-bit cell can be accessed at lower latency and energy than another bit in the same cell. We propose to exploit this asymmetry between the different bits by decoupling the bits and mapping them to logically separate memory addresses. This exposes reduced read latency and energy in one half of the memory space, and reduced write latency and energy in the other half of memory, to system software. We effectively utilize the reduced latency and energy by mapping read-intensive pages to the read-efficient half of memory, and write-intensive pages to the write-efficient half. Decoupling the bits also provides flexibility in the way data is buffered in the memory device, which we exploit to manipulate the physical row buffer as two logical row buffers for increased data locality in the row buffer.

Our evaluations for a multi-core system show that our proposal improves system performance by 19.2%, memory energy efficiency by 14.4%, and thread fairness by 19.3% over the state-of-the-art MLC PCM baseline system that does not employ bit decoupling. The improvements are robust across a wide variety of workloads and system configurations.

1 Introduction

With the growing need to process large amounts of data in diverse computing applications, there is a growing demand for large capacity memories at low cost. This trend, coupled with the technology scalability challenges of DRAM, has triggered interest in scalable resistive memories [35, 44, 11, 40], and a promising contender among them is Phase Change Memory (PCM). Adding to its better projected scalability than DRAM [22], PCM supports multi-level cell (MLC) technology where each cell stores two or more bits, providing a compelling density advantage over DRAM, which can only store a single bit per cell.

Most prior studies on PCM are generic in that they focus on alleviating problems common to both single-level cells (SLCs) and multi-level cells (MLCs) such as endurance, access delay, or write overhead issues [49, 30, 5, 13, 38, 45, 31, 22, 34]. In addition to improving density, MLC PCM has some characteristics different from SLC PCM, the architectural significance of which is open to further study. For instance, unlike SLC PCM, MLC PCM employs an iterative write mechanism to precisely control the analog resistance value of cells [2, 27]. In addition, different resistance levels of the MLC PCM cell take varying latencies to sense during a read operation [33].

Such characteristics of MLC PCM profoundly affect the latency and energy of a read or a write operation, which can vary by 16–80% depending upon the current and final states of the cell [2, 27, 16, 33]. This leads to asymmetry in access latency and energy between the different bits in a multi-bit cell. This cell access cost is especially important for MLC PCM as it accounts for 99% of the overall device write latency and more than 95% of the overall device read latency [32, 33, 22], which is unlike existing charge-based memories such as SRAM or DRAM where the latency of reading from or writing to the cell itself is small compared to the wire delays for transferring memory address and data.

In this paper, we propose exposing this asymmetry in latency and energy between the bits of an MLC cell to the memory controller and the system software (OS), to improve both performance and energy efficiency. Instead of delaying the serving of a memory request until the slowest bit in a multi-bit cell is ready (as is commonly assumed in PCM studies), we expose the lower latency of the faster bit, by mapping the different bits of a cell to logically separate memory addresses. This scheme, which we call *Decoupled Bit Mapping (DeBiM)*, divides a two-bit MLC memory space into a half that has reduced read latency and energy (read-efficient), and another half that has reduced write latency and energy (write-efficient).

Having made this divide, we devise *Asymmetry-aware Page Mapping (AsPaM)* to effectively utilize the reduced latency and energy of reads and writes to these two halves of MLC memory. This is performed using a hardware-software cooperative mechanism: The OS allocates a page to a physical frame either in the read-efficient half of memory or the write-efficient half using hardware prediction. We find that the instruction address that first-touches a page is a good predictor for whether the page will be read-intensive (mostly read without writebacks) or write-intensive (incurs many writebacks), and devise a hardware-based page read-write intensity predictor based on this insight.

DeBiM also provides more flexibility in the way row buffers[1] can be managed in PCM devices, which we exploit to increase the row buffer hit rate (the fraction of memory accesses that are row buffer hits). *Split Half-Row Buffering (SplitRB)* provides two logical row buffers from a single physical row buffer, enabling data from multiple rows to be buffered simultaneously in MLC memory. The effect of this is similar to adding another row buffer to the memory device, at low hardware cost.

Our evaluations compare our proposed schemes to a state-of-the-art MLC PCM system that does not take advantage of latency and energy asymmetry of reads and writes in the multi-level cell bits, nor is able to simultaneously buffer multiple rows. Our results show that the proposed schemes achieve higher system performance and energy efficiency for a variety of workloads and system configurations.

We make the following **contributions** in this paper:

- We introduce a new approach to mapping data in MLC PCM, which exposes and takes advantage of the lower access latency and energy of the faster bit in a multi-bit cell (for reads and writes). With this approach, a memory request is not conservatively delayed for the slowest bit in a cell, which results in reduced average memory latency and energy.
- We devise a hardware-software cooperative mechanism to effectively utilize the reduced latency and energy of reads and writes by mapping the virtual pages of an application to read-efficient or write-efficient frames in MLC memory. For this, we introduce a hardware predictor to estimate each page's read-write intensity.
- With the introduced data mapping scheme, we propose to manipulate the physical row buffer as two logical half-row buffers. We show that this data buffering scheme increases row buffer locality in MLC memory.
- We evaluate our proposed schemes for a variety of workloads, and show that our combined scheme improves system performance by 19.2% and energy efficiency by 14.4% over a state-of-the-art MLC PCM system. The benefits of the proposed mechanisms are robust to a variety of system configurations, and we show that our proposals require low implementation overhead.

[1] A row buffer is the sense amplifier that stores the currently-open memory row in a PCM or DRAM bank. Data can only be accessed after being brought into the row buffer [26].

2 Background

2.1 Memory Device Architecture

A typical memory device comprises arrays of memory cells and peripheral circuitry (shown in Figure 1). In an array, memory cells are organized into *rows* and *columns*, where all of the cells in each row are connected to a common word line, and all of the cells in each column are connected to a common bit line.When accessing data in the array, a row of cells is accessed simultaneously. In doing this, a row decoder asserts a word line to select all of the cells in the target row, and the bit lines transmit data between the cells and the peripheral circuits.

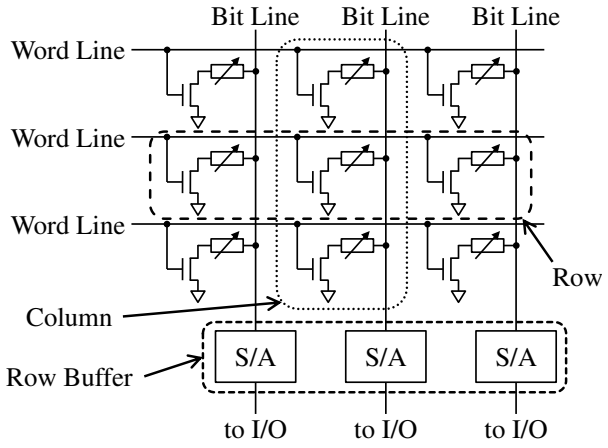


Figure 1: Memory cell array shown with PCM cells. S/A = sense amplifier.

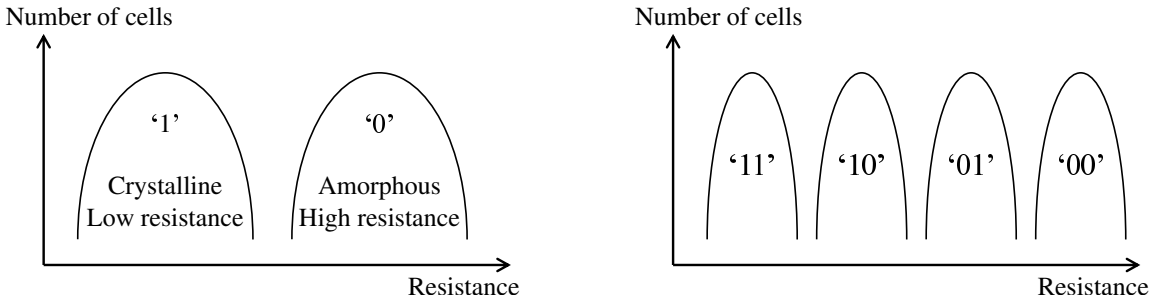
In the peripheral circuits, data signals from the bit lines are detected by sense amplifiers and latched in the *row buffer*, and a column decoder selects a subset of the row buffer to be communicated with the I/O pads. Once a row's data is placed in the row buffer, subsequent data requests to the same row can be served by accessing the data in this buffer. Such an access is known as a *row buffer hit*, and can be served quickly at the access latency of the row buffer without interacting with the slower cell array. However, in order to serve a data request to another row, data must be accessed from the array (replacing the contents of the row buffer). This type of access is known as a *row buffer miss*, and incurs higher latency and energy consumption due to activating a row of cells in the array.

Applications with high data locality benefit from large row buffers and incur reduced memory access time. But with multi-core processors, memory requests from multiple threads (processes) become interleaved while accessing the same bank, resulting in increased row buffer misses (hence high row buffer miss rates). This also increases the contention at the memory controller as memory requests tend to wait longer at the memory controller before being issued [26, 37, 23]. A possible solution to this problem is to increase memory parallelism by supporting multiple row buffers for each bank [24]. Thus an active row buffer's content will be less likely to get thrashed due to a conflicting access from another thread (process). However, this approach significantly increases the area overhead and memory cost [43]. In section 4.3, we discuss how we can exploit MLC PCM characteristics to achieve the benefits of multiple row buffers at low area overhead.

2.2 Phase Change Memory

Phase Change Memory (PCM) is an emerging memory technology that stores data by varying the electrical resistance of a material known as chalcogenide [35, 44]. By applying heat, and then allowing it to cool at different rates, chalcogenide can be manipulated to settle between an amorphous (quickly quenched) high resistance state, and a crystalline (slowly cooled) low resistance state (shown in Figure 2a). PCM is non-volatile, as the state of chalcogenide is retained in the absense of electrical power.

PCM's approach to storing data is fundamentally different from that of DRAM which stores data as a small amount of electrical charge in a capacitor. This gives PCM the potential for better technology scaling, leading to higher density, lower cost per bit, and larger capacity memory than DRAM [22]. PCM's low static power further adds to PCM's competitiveness as a scalable DRAM alternative.



(a) Single-level cell resistance distribution.

(b) Multi-level cell resistance distribution.

Figure 2: PCM cell resistance distribution.

Compared to DRAM, PCM's drawbacks [22] are that it exhibits higher access latency (by 4–32×) and dynamic energy (by 2× for read, 10–140× for writes). PCM also has limited write endurance, estimated at 10^4–10^9 writes. The development of PCM is ongoing work, and many solutions have been proposed to overcome PCM's shortcomings to achieve the goal of more scalable main memory. Some proposals include DRAM caching [31], fine-grained row buffering [22], wear-leveling/reduction [30, 5, 49], and fault-tolerance measures [13, 38, 45].

2.3 Multi-Level Cell Memory

Multi-Level Cell (MLC, or multi-bit cell) is the concept of storing more than one bit per memory cell. This is made possible in PCM due to the large resistance difference (by three orders of magnitude) between the amorphous and crystalline states of a cell. This large resistance range can be partitioned into more than two regions, each of which represents a distinct multi-bit value. Figure 2b shows an example where the cell's resistance range is partitioned into four regions representing the 2-bit (*most significant bit, MSB*, and *least significant bit, LSB*) values '11', '10', '01', and '00' respectively. More than one bit can be stored in a cell by controlling its resistance precisely to lie within one of these resistance regions. Thus, MLC increases the bit density of PCM at a given technology node, further lowering the cost per bit for memory.

However, supporting MLC in PCM incurs higher access latency and energy. MLC requires the cell resistance to be controlled precisely to lie within a narrower range, which necessitates iterative writing techniques that lead to higher write latency and energy [2, 27]. To read a multi-bit value from a cell, multiple sensing iterations are required to determine which resistance region a cell lies in, which increases read latency and energy [33].

3 Motivation

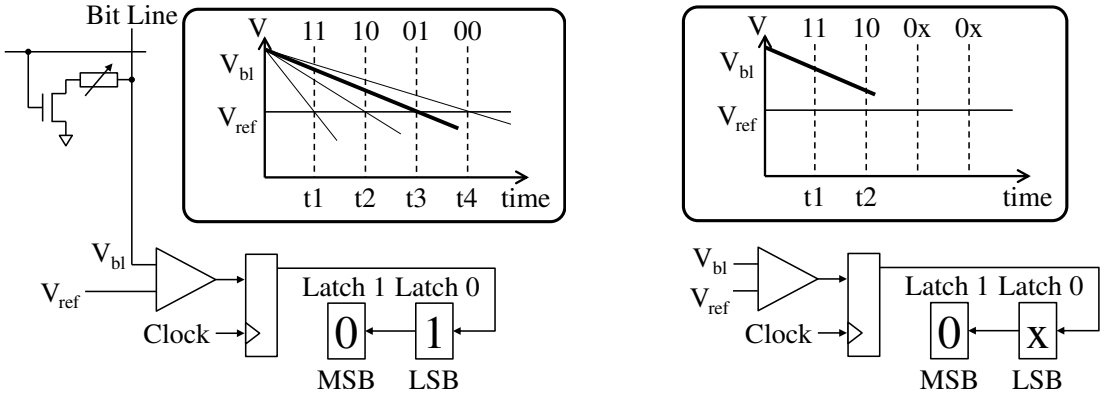
3.1 Read and Write Asymmetries

We examine the operation of MLC PCM, and identify the following properties that highlight the inefficiencies of the conventional MLC PCM architecture:

- *Read asymmetry:* The bits in an MLC PCM cell have different read latencies.
- *Write asymmetry:* The bits in an MLC PCM cell have different write latencies (also noted by [16]).

Read asymmetry. The read latency of an MLC PCM cell depends on the state of the cell. This is illustrated in Figure 3a, which depicts a 4-level PCM read operation. An integrating analog-to-digital converter (ADC) quantizes the resistance of an MLC PCM cell to a 2-bit [MSB, LSB] value by sensing the time taken to pulse an electrical current through the cell (counting down '11', '10', '01', '00' until V_{bl} crosses over V_{ref}) [33]. The higher the cell resistance, the longer the discharge time, hence the longer the sensing (read) time. Consequently, the read latency is bounded by the time taken to sense the highest cell resistance.

However, it is possible to discern some information about the cell's data before the read operation is fully done. As illustrated in Figure 3b, the MSB's value can be determined half way through the read operation. If the discharge



(a) Sensing time is longer for higher cell resistances.

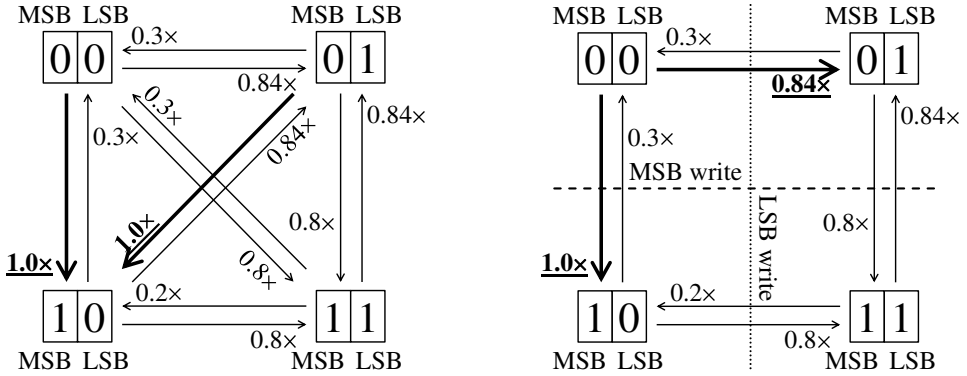
(b) MSB is determined before LSB.

Figure 3: MLC PCM read operation [33].

is sensed (if V_{bl} has crossed over V_{ref}) by half way through the read operation, the MSB is a '1', else it is a '0' (as is the case in the illustrated example), irrespective of the LSB.

This observation indicates that the MSB has lower read latency (and energy) than the LSB. However, this property is not exploited in the conventional MLC PCM architecture, where a logical block of data (e.g., belonging to the same logical page or cache block) is spread across MSBs and LSBs. This delays the serving of a memory read request until the slower LSBs are sensed.

Write asymmetry. The write latency of an MLC PCM cell depends on two things; the initial state of the cell, and the target state of the cell. This is illustrated in Figure 4a that shows the latencies incurred in transitioning a cell from any one state to another in a 4-level PCM write operation. For any transition, the write latency is optimized by using the programming method (either partially crystallizing amorphous chalcogenide, or partially amorphizing crystalline chalcogenide) that achieves the target cell resistance at a lower latency [27, 10, 16]. When writing arbitrary data to blocks of cells, the write latency is bounded by the longest time to complete any of the transitions (bold in Figure 4a).



(a) Cell programming: All possible transitions of cell states.

(b) Restraining the simultaneous alteration of the MSB and LSB.

Figure 4: MLC PCM write latencies (normalized to the '00' → '10' program latency) [16, 27, 10].

However, if we do not alter both MSB and LSB in a single write operation (such that the diagonal transitions in Figure 4a are not being used), then altering the LSB incurs lower latency than altering the MSB. Figure 4b highlights that in this case, changing the MSB only is bounded by a 1.0× latency ('00' → '10'), whereas changing the LSB only is bounded by a lower 0.84× latency ('00' → '01').[2]

This observation indicates that the LSB has lower write latency (and energy) than the MSB. However, similar to read asymmetry, this property is not leveraged when a block of data is spread across LSBs and MSBs, as is the case in

[2]This is because programming to '10' incurs a 0.2× latency only when transitioning from '11' that is already in the crystalline state, where partial amorphizing only requires reset pulses to be applied [10, 16].

the conventional MLC PCM architecture.

3.2 Exploiting Access Asymmetries

In this paper, we propose to exploit the read and write asymmetry properties of MLC PCM by decoupling the MSBs and LSBs of cells and mapping them to logically separate memory addresses. This *Decoupled Bit Mapping* scheme is illustrated in Figure 5, where data blocks are now composed of a collection of bits that are entirely MSBs or LSBs. Reading a data block that is stored in MSBs requires the sensing of only MSB values, thus such a block can be read at the lower MSB read latency (while data blocks stored in LSBs could be read at the same latency as before). Likewise, writing a data block that is stored in LSBs requires the programming of only LSB values, thus such a block can be written at the lower LSB write latency (while data blocks stored in MSBs could be written at the same latency as before).

To fully exploit the read and write asymmetry properties, we propose three techniques for mapping and buffering data in MLC PCM for higher memory system performance and energy efficiency: First, in *Decoupled Bit Mapping (DeBiM)* we map the MSBs and LSBs of cells to logically separate memory addresses. This divides the memory space into an MSB half that has reduced read latency and energy, and an LSB half that has reduced write latency and energy. Second, in *Asymmetry-aware Page Mapping (AsPaM)*, the system software (OS) maps an application's virtual page to a physical frame either in MSB or LSB, depending on the page's read-write access intensity that is predicted in hardware. Third, *Split Half-Row Buffering (SplitRB)* leverages how DeBiM divides a row of cells into an MSB half-row and an LSB half-row, and manipulates the row buffer as two separate half-row buffers to increase row buffer hit rate.

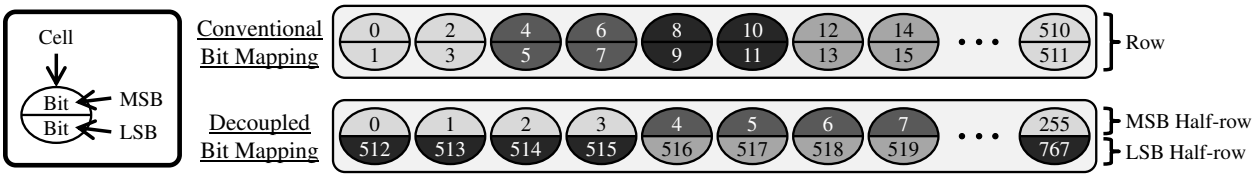


Figure 5: Decoupled Bit Mapping (DeBiM). For illustration, data blocks the size of 4 bits are highlighted in different shades. We use 64 byte data blocks (cache blocks) in our evaluations (Table 2).

The benefits of DeBiM, AsPaM, and SplitRB are illustrated in Figure 6, which also compares the memory service timelines between the conventional MLC PCM architecture and systems employing these schemes, for an example case of serving reads and writes in memory. Compared to the conventional scheme, DeBiM is able serve read requests to blocks 17 and 18 at the lower MSB latency and energy, as these blocks now only occupy MSBs (additionally, blocks in the LSB half-rows can be written to at the lower LSB latency and energy). With AsPaM, the logical half-row containing block 1 is allocated to an LSB half-row, whereas this logical half-row is allocated to an MSB half-row with only DeBiM (by its natural address progression). Through memory-usage-aware allocation, AsPaM is able to exploit the lower LSB latency and energy to serve the write request to block 1, which was served at the higher MSB latency and energy with only DeBiM. Serving the read request to block 18 results in a row buffer miss access without SplitRB, as the row buffer will be holding the row containing the previously accessed block 10. However with SplitRB, the half-rows containing blocks 10 and 17 (accessed before block 10) can be simultaneously buffered in the row buffer, causing the read request to block 18 (which resides in the same half-row as block 17) to be a row buffer hit access. This further reduces the time and energy taken to serve all of the memory requests.

4 Mechanism

4.1 Decoupled Bit Mapping (DeBiM)

Decoupled Bit Mapping (DeBiM) is designed to expose the lower latency and energy of MSB reads and LSB writes. To this end, the bits of an MLC cell are mapped to separate logical addresses, as illustrated in Figure 5. Whereas these bits are coupled to form a single contiguous memory address along a row in the conventional scheme, DeBiM groups the MSBs along a row to form one contiguous address, and groups the LSBs along the same row to form another. This

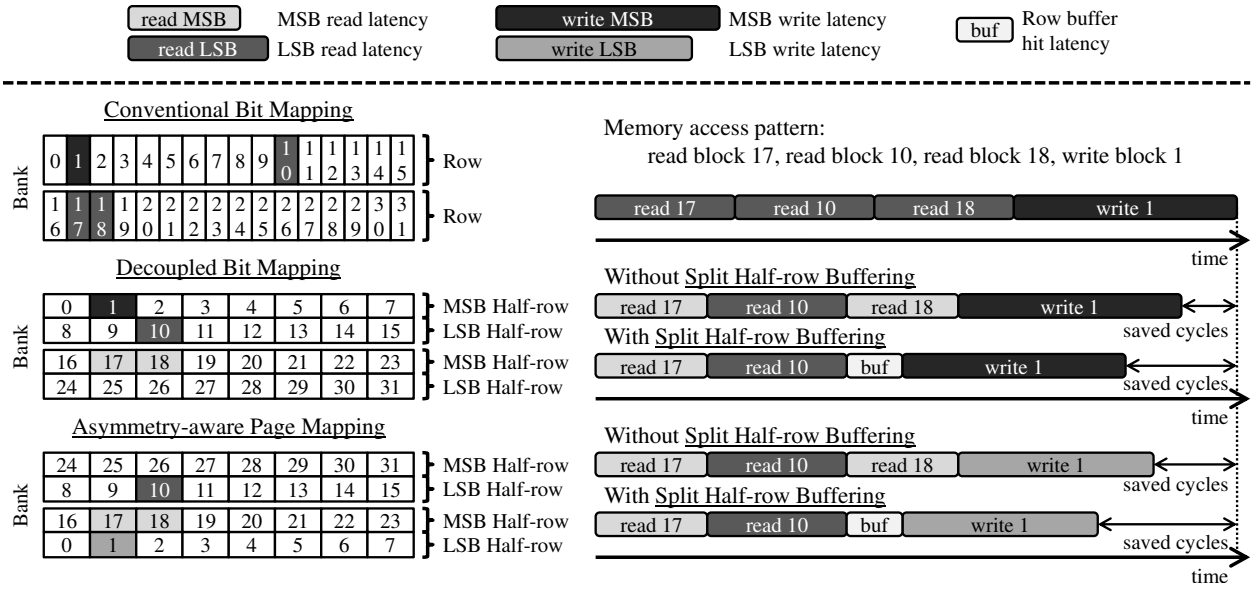


Figure 6: A conceptual example illustrating the benefits of Decoupled Bit Mapping, Asymmetry-aware Page Mapping, and Split Half-Row Buffering. The logical progression of data block (cache block) addresses are shown for 64 byte blocks and 4k cell rows.

way, a data block (e.g. a 64 byte cache block) that resides at a certain logical address physically occupies only MSBs or only LSBs. If the data block is in MSBs, read asymmetry is exploited to read the block at a reduced latency and energy. If the data block is in LSBs, write asymmetry is exploited to write to the block at a reduced latency and energy.

DeBiM divides all rows in memory into two logical addresses; one using MSBs, the other using LSBs. Figure 6 contrasts the data block address mapping in DeBiM and the conventional scheme. Assuming an arbitrary random translation from an application's virtual page address to a physical frame address in memory, typically half of the application's working set would be in MSBs and the other half in LSBs. Hence with DeBiM, on average, 50% of the memory reads are served at reduced latency (by 48%) and energy (by 48%), and 50% of the memory writes are served at reduced latency (by 16%) and energy (by 26%).

The downside of DeBiM is that it increases the number of cells that are programmed during a write operation, incurring an endurance overhead. This is because a data block uses only one bit from each 2-bit cell, involving a number of cells equal to the number of bits in a block when writing to the block. However, this does not double the write endurance overhead compared to the conventional scheme, as the probability of redundant bit-writes [49] (where the cell is already in the target state to be programmed to) is lower in the conventional scheme. For programming a cell to be redundant in the conventional scheme, both the MSB and LSB have to match with the write data. On the other hand, write data to a block in DeBiM targets only MSBs or only LSBs. If the write is to MSBs, the LSBs are unchanged, and vice versa. Because only one bit per cell needs to match with the write data in DeBiM, the probability of redundant bit-writes are higher in DeBiM than in the conventional scheme (also noted by [49, 15]), and we observe on average a 28.2% endurance overhead in our results (Section 7.4). This is small enough to achieve the typical server design lifetime of 5 years, considering that prior work has shown PCM main memory lifetime of 8.8 years [45].

By employing DeBiM, two separate logical address spaces share the row buffer space, each address space occupying half of the row buffer. This reduces the longest contiguous address space that can be held in the row buffer, potentially decreasing row buffer locality. However, the reduced memory access latency and energy exposed by DeBiM more than compensate for this effect, significantly improving system performance and energy efficiency over the conventional MLC PCM architecture, while not incurring any major modifications to the memory circuitry and architecture (implementation discussed in Section 5). To utilize read and write asymmetries more effectively, we next propose to map a page to a frame either in MSB or LSB depending on the page's memory read-write intensity.

4.2 Asymmetry-aware Page Mapping (AsPaM)

With DeBiM alone, there is only a 50% probability that a read or a write to memory will benefit from reduced latency and energy. This is because only half of the memory space is efficient in serving reads, while the other half of memory is efficient in serving writes. However, many applications typically exhibit skewed page access behavior where a small fraction of pages account for a large portion of the memory accesses. Table 1 shows this trend for SPEC CPU2006 benchmarks, listing the fraction of pages that account for 80% of the memory reads (RP) and 80% of the memory writes (WP). On average, 80% of the reads and writes are accounted for by 35% and 11% of the pages respectively. Furthermore, the table also lists the fraction of the frequently-read pages (in top 10% in terms of reads to page) serving more than 10 reads per write (RB), and the fraction of frequently-written pages (in top 10% in terms of writes to page) serving less than 2 reads per write (WB). This fraction is greater than 80% for 17 benchmarks in either reads or writes, indicating that many applications frequently access pages with a strong bias toward reads or writes. Thus, by placing these access-intensive pages in frames that are efficient in serving the type of access (read or write) the pages frequently receive, it is possible to serve a larger portion of memory requests at reduced latency and energy.

There are two different approaches to achieving this page placement. The first approach is to monitor the memory access patterns to pages, and migrate access-intensive pages to favorable frames (read-intensive pages to MSB frames and write-intensive pages to LSB frames) if they are not already placed in such frames. This straightforward approach allows matching pages to frames with high confidence, based on the observed memory access characteristics of pages. However, the disadvantages of this approach are that it incurs high endurance and energy overhead, as well as increasing bank occupancy due to the movement of data in memory. An alternative approach is to predict the memory access patterns of pages and place them in favorable frames at their initial allocation. This method avoids the shortcomings of the first approach, however, it requires accurate predictions of the memory read-write characteristics of pages to be allocated.

	perlbench	bzip2	gcc	bwaves	gamess	mcf	milc	zeusmp	gromacs	cactus	leslie3d	namd	gobmk	dealII
RP	36	17	44	70	22	7	68	60	38	57	51	54	60	23
RB	80	54	81	55	99	100	25	40	36	83	0	44	9	15
RI	54	7	11	44	7	2	3	37	22	43	20	2	9	9
WP	8	10	22	3	7	< 1	37	53	23	26	32	39	58	26
WB	77	100	70	1	45	1	43	54	36	84	8	100	93	60
WI	84	4	8	40	2	2	4	20	11	23	12	1	8	9
	soplex	povray	calculix	hmmer	sjeng	Gems	libq	h264ref	lbm	omnetpp	astar	sphinx3	xalanc	
RP	20	22	24	17	40	63	73	19	75	46	54	39	12	
RB	17	65	95	0	27	94	100	0	0	63	52	88	99	
RI	18	44	4	3	3	33	2	22	2	13	4	12	20	
WP	10	4	3	16	26	7	< 1	15	75	22	55	1	1	
WB	35	27	8	100	5	2	6	11	100	1	98	29	9	
WI	16	11	1	3	3	6	2	21	2	11	2	4	18	

Table 1: Memory access patterns of SPEC CPU2006 benchmarks. RP (WP) = percent of pages accounting for 80% of reads (writes), RB (WB) = percent of frequently-read(written) pages serving more than 10 (less than 2) reads per write, RI (WI) = instructions that first-touch pages accounting for 80% of reads (writes).

We devise Asymmetry-aware Page Mapping (AsPaM) that makes this prediction and advises the OS page allocation routine on the type of frame (MSB or LSB) to map the page to. AsPaM predicts whether the page to be allocated will be read-intensive or write-intensive[3], which the OS allocates to MSB and LSB frames respectively. If the favored type of frame is unavailable (i.e., all occupied) whereas the other type is available, then the page is allocated to the available frame type.

Table 1 shows that typically fewer instructions in a program first-touch (and trigger the OS to allocate physical frames for) the pages that experience the majority of memory writes, compared to the case for reads.[4] Leveraging this, AsPaM tracks N (= 16) instructions (identified by the program counter) that generate the most writes to memory, and predicts a page to be write-intensive if its first-touch instruction is one of those N instructions; else the page is classified as read-intensive. We find that using an N value of 16 achieves the system performance within 1% of using an N value of 256.

[3]Although writes are not on the critical path, they occupy the banks for a long time (8× that of reads), and reducing this latency decreases the queuing delay of read requests.

[4]Modern operating systems typically allocate a virtual page to a physical frame only when the data is accessed in the memory for the first time, as opposed to allocating it right after a request for memory space [18, 4, 8].

To approximate the N instructions in run-time, a hardware PC (program counter) table of N entries is maintained in the processor core, as shown in gray in Figure 7. Each entry in the PC table holds an instruction address and a corresponding counter for counting the number of memory writes (dirty evictions from the cache) generated by the instruction. The PC table is used to tally the instructions with the number of memory writes they generated, while evicting instructions with low memory-write counts from the PC table. In order to track the instruction that wrote to a cache block, each cache block entry in the cache is augmented with a $\log_2 N$ (= 4) bit field (also gray in Figure 7) that stores an index value to the PC table that is containing the instruction that wrote to the cache block.

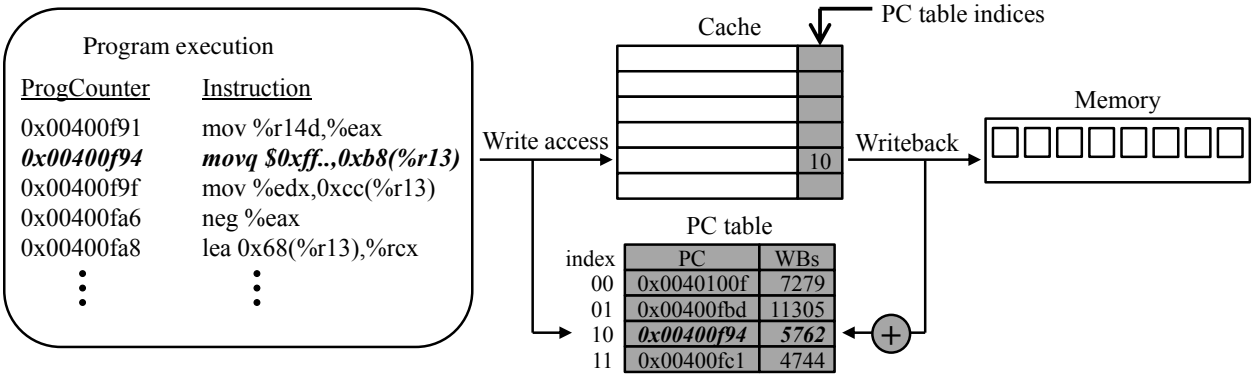


Figure 7: PC table that tracks the instructions that generate the most writes to memory.

The PC table operates as follows: For every write to the cache during the execution of a program, the address of the instruction (in the program counter) causing the write is checked to see if it matches with any of the instruction addresses in the PC table. If there is no match, the instruction in the PC table with the smallest memory-write counter value is replaced with the write-causing instruction address, and the corresponding counter value is reset to 0. Now that there is an entry in the PC table with the write-causing instruction, the index of this entry is stored in the cache, in the $\log_2 N$ bit field (mentioned previously) associated with the cache block that was written to. In the event that this cache block is written back to memory, its associated index value is used to index the PC table and increment the memory-write counter. Because entries in the PC table can be replaced with new instructions, it is possible for a memory-write counter to be incremented for an instruction that has already been evicted from the PC table.

During OS page allocation, a custom instruction is executed that checks whether the page's first-touch instruction address matches with any of the instruction addresses in the PC table. If there is a match, a register flag is set signalling that the page is predicted to be write-intensive (otherwise the page is predicted to be read-intensive). Depending on the prediction, the OS allocates the page to either an MSB frame or an LSB frame (which can be told apart from the physical memory address as shown in Figure 6). All memory-write counters in the PC table are multipled by 0.25 (right-shifted by 2) every 10 million cycles (empirically set) to adjust for phase changes in the program.

4.3 Split Half-Row Buffering (SplitRB)

So far we discussed how DeBiM exploits read and write asymmetry properties in MLC PCM to provide reduced memory access latency and energy, and how AsPaM facilitates their effective utilization. In addition to these benefits, DeBiM enables more flexibility in managing the row buffer, which is not possible in the conventional MLC architecture.

In multi-level PCM, a sensing circuit should have as many latches as the number of bits stored in a cell, to be able to retrieve all of the bits of a cell. This is true for both the conventional MLC PCM architecture and our proposal, since sensing the MSB requires the LSB value to be first stored as shown in Figure 3.

DeBiM divides a row into two half-rows, each with their own contiguous logical address. This enables the row buffer to be manipulated as two half-row buffers as shown in Figure 8. One of the two latches in every sensing circuit along a row buffer is grouped to form one half-row buffer, and the other latches along the same row buffer are grouped to form another half-row buffer. We propose this fine-granularity organization of half-row buffers which we call *Split Half-Row Buffering (SplitRB)*.

SplitRB enables both half-row buffers to buffer MSB and LSB half-rows. Furthermore, two half-rows from different rows (or the same row) can be buffered simultaneously. The two half-row buffers are used associatively: On a

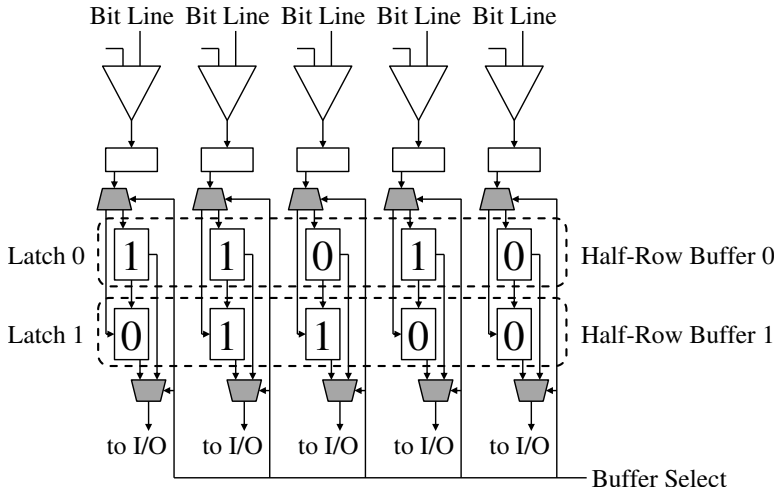


Figure 8: Split Half-Row Buffering (SplitRB). The row buffer is manipulated as two half-row buffers.

row buffer miss access, the least recently used half-row buffer's content is chosen for eviction. SplitRB has an effect similar to adding another row buffer to the memory device, at low hardware cost (more details in Section 5). This dual half-row buffer organization increases the row buffer hit rate in MLC memory compared to when using a single full-row buffer, thereby achieving higher performance and energy efficiency.

A limitation of SplitRB is that while two LSB half-rows can be simultaneously buffered, two MSB half-rows cannot. Though sensing an LSB can be performed using a single latch in the sensing circuit, sensing an MSB needs both latches as it requires the latch buffering the LSB to count down the bit values ('11', '10', '01', and '00') together (see Figure 3). Hence reading LSBs into a half-row buffer can be done without affecting the contents of the other half-row buffer, whereas reading MSBs into a half-row buffer invalidates the contents of the other half-row buffer.

For data correctness, the MSBs and LSBs of the same row must be present in the row buffer before receiving any write data in the row buffer. The reasoning for this is as follows: To program an MLC cell, both the target MSB and LSB values are required, while it is possible that only one of these is present in the row buffer. However, the missing MSB or LSB values needed to perform the program operation cannot be read from the cell array if the latches in the row buffer are already occupied with write data. To avoid such a scenario, the memory controller follows an algorithm[5] shown in Figure 9 when serving memory read and write requests. The purpose of this algorithm is to ensure the presence of MSBs and LSBs of the same row in the row buffer before receiving any write data. When serving a memory write request that hits in the row buffer, the algorithm first checks whether the row buffer holds MSBs and LSBs of the same row, and if not, fetches the missing data from the cell array. When serving a memory read or write request that misses in the row buffer, dirty row buffer contents are written back to the cell array, and if the request is a write, both MSB and LSB data are fetched to the row buffer. Despite these constraints, SplitRB improves row buffer locality as we show in Section 7.1.

Dividing the row buffer into many shorter row buffers to achieve higher row buffer hit rate was previously proposed by Lee et al. [22]. The main tradeoff regarding the number of shorter row buffers to form is between benefiting from the plurality of buffers, while not being significantly disadvantaged on exploiting spatial locality due to the shorter individual buffer length (the die area occupied by the row buffers is not increased). SplitRB differs from this prior approach in that spatial locality is not affected (beyond the point that is already affected by DeBiM). This is because the row buffer is divided between the MSB and LSB half-rows (that each have their own contiguous logical address) instead of dividing in the column direction. Thus, SplitRB provides an effective fine-granularity row buffer organization to improve the MLC performance and energy efficiency.

[5]This is implementation-dependent, and can be foregone at the cost of additional latches dedicated to storing write data.

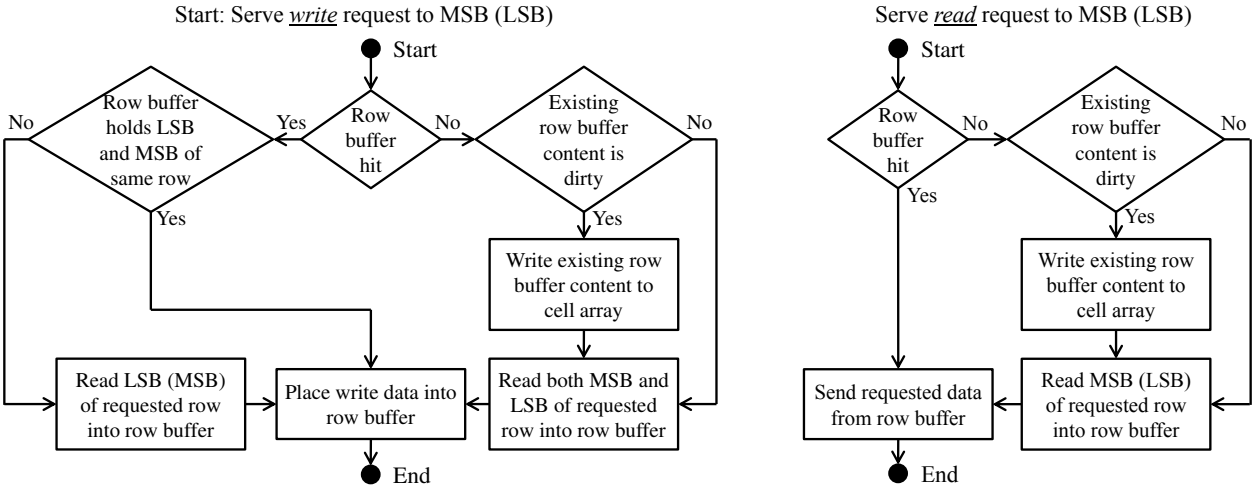


Figure 9: Memory controller algorithm for data correctness. It ensures the presence of MSBs and LSBs of the same row in the row buffer before receiving write data in the row buffer.

5 Implementation and Cost

PCM array. DeBiM does not require any additional die area in the cell array, and can be implemented with minimal modifications to the control signal and bus organization. As discussed in Section 2.1, memory devices are accessed in two steps: First, the row decoder activates a row and its contents are stored in the row buffer. Second, the column decoder selects a subset of the row buffer to be communicated with the I/O pads. With DeBiM, during row activation, an additional control signal decoded from the memory address notifies the sense amplifiers whether to retrieve the MSB or LSB. The overhead of this is negligible since a single signal can be shared by an entire row of sense amplifiers.

Row buffer. To select a data block (cache block) in the row buffer, DeBiM simply requires reorganizing the latch-to-I/O multiplexers such that only MSBs or only LSBs are selected for communication with the I/O pads (the number of bits communicated is the same in DeBiM and the conventional scheme). Depending upon the implementation of the sensing circuit, SplitRB will require reconfiguring the bit-line-to-latch multiplexers, such that MSBs and LSBs can be buffered in either of the two latches in the row buffer. These multiplexers are shown shaded in gray in Figure 8.

We used NVSim [9] to calculate the cost of the additional multiplexing in a PCM memory modeled after that in [19], and found the overall area of a 256Mb array increases by 0.7%.

Memory controller. The memory scheduler should be aware of the half-row access latencies (known from the physical memory address, see Figure 6) and the different half-row buffers within the row buffer. In addition, serving memory requests should follow the algorithm shown in Figure 9 to ensure data correctness (details in Section 4.3).

Processor. AsPaM requires maintaining a PC table of N (= 16) program counter values and memory-write counters attributed to each. Assuming a 64-bit address space and 18-bit write counters, this amounts to 164 bytes of storage, which is 0.5% of the L1 cache size assumed in the evaluation. Augmenting the cache with a $\log_2 N$ (= 4) bit field for each cache block increases the cache size by 0.7% for the both the L1 and L2 caches.

6 Experimental Methodology

For the evaluation of our proposed MLC PCM architecture we develop a cycle-level MLC memory simulator, which interfaces with an in-house x86 multi-core simulator that has a front-end based on Pin. Representative phases of benchmarks were executed as profiled by PinPoints [29]. The major system parameters used in our study are shown in Table 2. Results were collected for one billion cycles following a warmup period of 0.4 billion cycles.

The use of a DRAM cache has been proposed in prior work to mitigate the latency, dynamic energy, and wearout issues of PCM [31, 25, 7]. In this study, we assume a large on-chip eDRAM L3 cache similar to that of the IBM Power7 processor [17].

We use the four benchmarks from the NAS Parallel Benchmark suite and the six benchmarks from the SPEC CPU2006 suite that show misses per kilo-instructions (MPKI) higher than 1.75 and 2.5 respectively at the on-chip

Processor	8 cores, 4 GHz, single-issue in-order.
L1 cache	Private 32 KB per core, 4-way, 64 B blocks.
L2 cache	Private 512 KB per core, 8-way, 64 B blocks.
eDRAM	Shared 16 MB, 16-way, on-chip [17].
Memory ctrl.	2 controllers each with 64-bit channel, 128-entry read/write request queues, FR-FCFS request scheduling [36, 50].
Main memory	MLC PCM, 2 ranks with 8 banks each, 4k cells in a row ($\times$8 devices in a rank).
Memory timing	Baseline: Array read (write) = 250 ns (2 μs). Decoupled: MSB read (write) = 125 ns (2 μs), LSB read (write) = 250 ns (1.68 μs). Both: Row buffer access = 12.5 ns.
Memory energy	Baseline: Array read (write) = 10.89 (368) pJ/bit. Decoupled: MSB read (write) = 5.68 (368) pJ/bit, LSB read (write) = 10.89 (272) pJ/bit. Both: Row buffer read (write) = 0.93 (1.02) pJ/bit.

Table 2: Major simulation parameters. Memory timing and energy are based on [16, 32, 22].

eDRAM cache: Scalar Penta-diagonal solver, Data Cube, Conjugate Gradient, Integer Sort; and milc, mcf, omnetpp, libquantum, lbm, soplex. We form two groups of 8-core system workloads from each benchmark suite: Homogeneous workloads running multiple single-threaded instantiations of each benchmark, and heterogeneous workloads formed by a random selection of benchmarks. In addition, we evaluate the execution of 8 threads during the dominant parallel execution phase (exec_updates()) of the GraphLab framework for a Twitter social network data set [21] (in Table 3).

#	Composition	R-Util	W-Util
N1	ScalarPenta$\times$8 (abbrev.: SPsolv)	24.9	4.8
N2	DataCube$\times$8 (abbrev.: DCube)	28.1	1.0
N3	Conj.Grad.$\times$8 (abbrev.: CGrad)	15.2	0.1
N4	IntegerSort$\times$8 (abbrev.: ISort)	25.3	15.1
O1	SPsolv, DCube, CGrad$\times$4, ISort$\times$2	23.2	3.9
O2	SPsolv, ISort$\times$4, DCube$\times$3	26.3	7.4
O3	ISort$\times$5, CGrad, SPsolv, DCube	25.6	9.4
O4	SPsolv$\times$3, ISort$\times$3, DCube$\times$2	26.3	5.4
S1	milc$\times$8	27.2	9.0
S2	mcf$\times$8	31.0	2.0
S3	omnetpp$\times$8 (abbrev.: omnet)	27.9	11.7
S4	libquantum$\times$8 (abbrev.: libq)	12.4	0.001
S5	lbm$\times$8	22.8	24.7
S6	soplex$\times$8 (abbrev.: sop)	27.3	7.2
T1	omnet$\times$3, milc, sop, mcf$\times$2, libq	27.8	6.8
T2	milc$\times$2, sop$\times$2, lbm$\times$2, mcf, omnet	27.9	11.2
T3	omnet$\times$2, lbm, milc, sop$\times$2, libq$\times$2	26.8	8.6
T4	milc, libq$\times$3, lbm$\times$2, omnet$\times$2	26.2	10.2
G1	GraphLab: page_rank	18.6	11.1
G2	GraphLab: connected_components	31.8	8.0
G3	GraphLab: triangle_counting	22.5	19.5

Table 3: 8-core workloads. R-Util = PCM bank utilization for reads, W-Util = PCM bank utilization for writes.

We adopt *redundant bit-write* removal in PCM to reduce memory wear. This involves writing to a cell only if the write data is different from that already stored in the cell. This technique removes 77% and 85% of writes in the baseline case and the DeBiM case respectively [49, 15]. Less writes are removed in the baseline case, as both the MSB and LSB of a cell have to match with the write data for the write to be redundant.

We report the speedup of multiprogrammed workloads using the *weighted speedup* metric [39], which is the

sum of the speedups of the benchmarks when executed together on the evaluated system, relative to when executed individually on the baseline system. For parallel workloads, we report the speedup of the time it takes for all of the threads to complete execution. For multi-core fairness, we use the *maximum slowdown* [3] metric, which is the highest slowdown (reciprocal of speedup) experienced by any benchmark in the workload.

7 Experimental Results

We evaluate six different MLC PCM systems, which we describe below. We then discuss their impact on memory access latency, system performance, energy efficiency, and memory lifetime.

- *Conventional system:* The baseline MLC PCM architecture where the bits of MLC cells are coupled to form a single contiguous memory address along a row.
- *All-LSB:* Models a scenario where all of the pages are mapped to LSB frames. Hence, memory writes always incur reduced latency and energy.
- *All-MSB:* Models a scenario where all of the pages are mapped to MSB frames. In this case, memory reads always incur reduced latency and energy.
- *DeBiM:* Models our proposed Decoupled Bit Mapping scheme that maps the MSBs and LSBs of cells to logically separate memory addresses.
- *Enhanced-DeBiM (E-DeBiM):* In addition to DeBiM, this includes Asymmetry-aware Page Mapping (AsPaM) and Split Half-Row Buffering (SplitRB).
- *Always-Low:* This models a hypothetical scenario where both memory reads and writes always incur reduced latency and energy (not achievable with MLC via known methods).

7.1 Memory Read Latency

There are three major components to memory access time: (1) The queuing delay of a request from when the request arrives at the memory controller to when it is issued to a bank, (2) the time taken to serve the memory access at the bank, and (3) the data transfer time through the memory channel and chip interconnects. The primary focus of this paper is to improve components one and two, which account for 95-99% of PCM access time [32, 33, 22]. Specifically, DeBiM and AsPaM reduce the average bank access latency, and SplitRB increases the row buffer hit rate (which also reduces the average bank access latency). All of these techniques reduce the access turn-around time and hence the queuing delay at the memory controller.

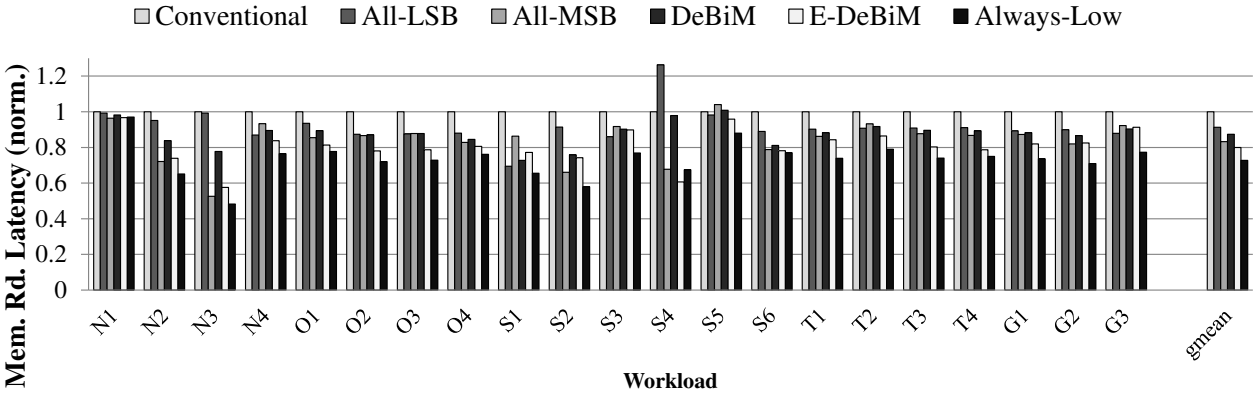


Figure 10: Average memory read latency of the six different MLC PCM systems.

Figure 10 shows the average memory read latency of the different MLC PCM systems. The plot is normalized to the Conventional system, and the geometric mean is plotted on the right. DeBiM alone reduces the average memory

read latency by 12.6% (and write latency by 5.4%) relative to the Conventional system. Whereas DeBiM serves 50% of reads in MSB frames and 51% of writes in LSB frames, Enhanced-DeBiM, with AsPaM, serves 57% of reads in MSB frames and 68% of writes in LSB frames (Figures 11a and 11b). This further reduces the read latency by 20.0% (and write latency by 7.2%) relative to the Conventional system. Enhanced-DeBiM, with SplitRB, also exhibits a row buffer hit rate 21.7% higher than DeBiM ((Figure 12a).

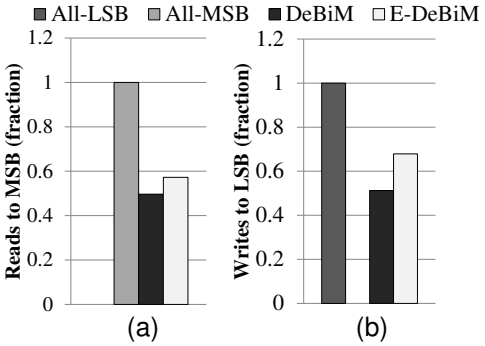


Figure 11: Fraction of reads to MSB and writes to LSB.

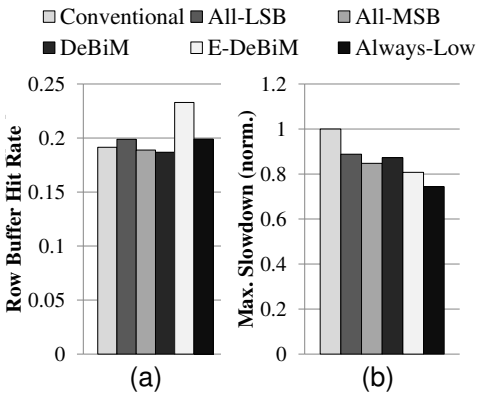


Figure 12: Row buffer hit rate and maximum slowdown.

The All-MSB system shows a memory read latency decrease of 16.7% relative to the Conventional system, benefiting from the low MSB read latency that it is always able to exploit. On the other hand, the All-LSB system shows a smaller reduction (8.7%) in memory read latency, which is interesting considering that the write latency is close to an order of magnitude larger than the read latency, highlighting the criticality of reads over writes. The (unachievable) Always-Low system provides the largest memory read latency reduction (27.2%) by having both the lower MSB read latency and lower LSB write latency simultaneously.[6]

7.2 System Performance and Fairness

Figure 13 shows the speedup of the MLC PCM systems. The effect of reduced memory read latency carries forward to system performance, with DeBiM seeing an 13.0% improvement in performance over the Conventional system. The Enhanced-DeBiM system that has lower memory read latency increases performance by a larger amount; 19.2% over the Conventional system. The All-MSB system demonstrates the gain achievable (15.9%) if all of the applications' pages exploit low-latency MSB reads. The All-LSB system shows a smaller gain (10.1%) by exploiting only the low-latency LSB writes. If both fast reads and fast writes could be adopted simultaneously, the Always-Low system would have a performance improvement of 30.5% over the Conventional.

[6]It is possible for E-DeBiM to show a lower memory read latency than Always-Low due to SplitRB.

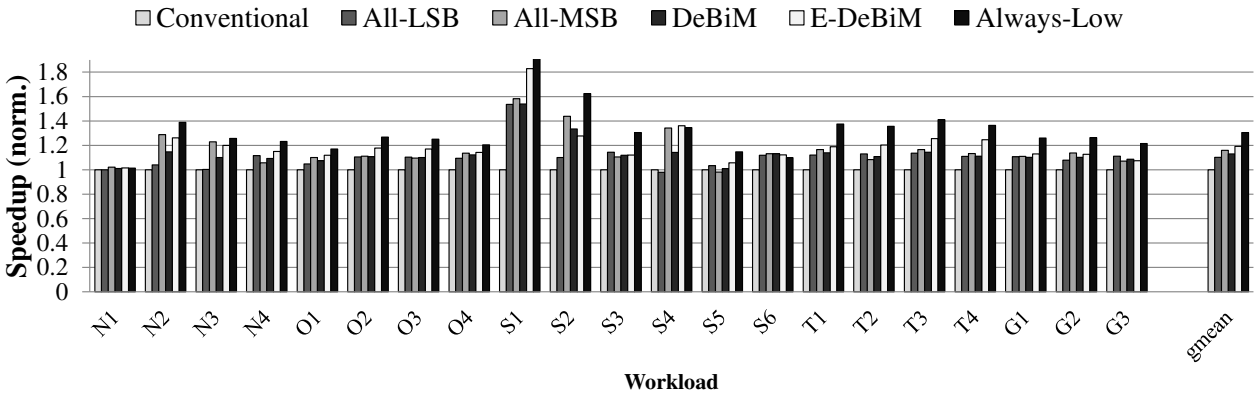


Figure 13: Speedup of the six different MLC PCM systems.

Figure 12b shows the maximum slowdown (lower is better) of threads experienced in the different MLC PCM systems. The improved performance of DeBiM contributes to lowering the maximum slowdown by 12.7% relative to the Conventional system. The Enhanced-DeBiM system further enhances thread fairness with higher performance and SplitRB that reduces the contention among threads for use of the row buffer, bringing down the maximum slowdown by 19.3% relative to the Conventional system. This is better than the fairness of the All-MSB system (15.2% improvement over the Conventional system), which also highlights the importance of optimizing for both reads and writes in PCM systems.

7.3 Memory Energy Efficiency

The energy efficiency of the memory system is shown in Figure 14 in terms of system performance per memory Watt. DeBiM enables the efficient activation of half-rows instead of full rows during MSB reads, which reduces the overfetching of data in the array [1, 42, 48]. DeBiM improves energy efficiency by 7.9% compared to the Conventional system. The Enhanced-DeBiM system facilitates higher utilization of MSB reads (and efficient LSB writes) through AsPaM while also increasing the row buffer hit rate (reducing array access) through SplitRB, raising the energy efficiency improvement to 14.4% over the Conventional system.

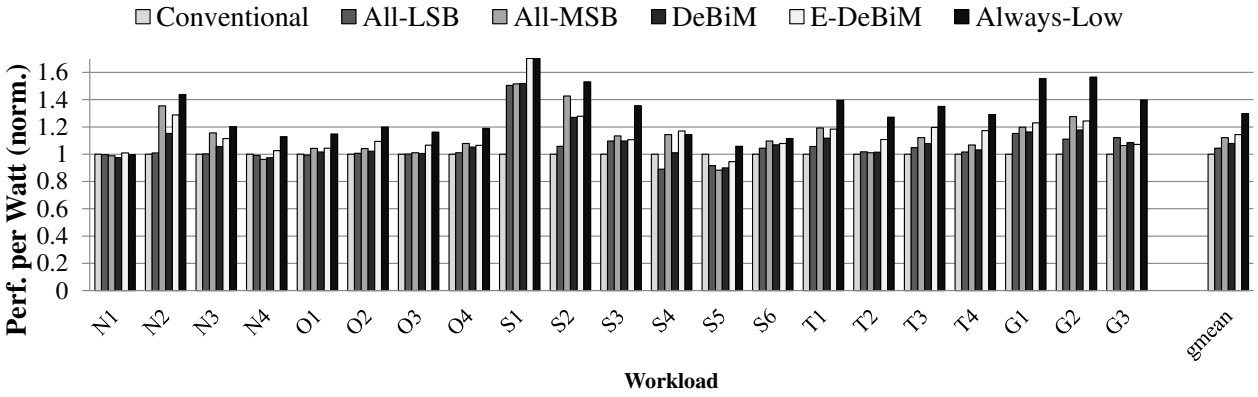


Figure 14: Energy efficiency of the six different MLC PCM systems.

The All-MSB system shows an energy efficiency improvement of 12.2% compared to the Conventional system by always activating MSB half-rows during reads. However, it always incurs the high-energy MSB write cost, and thus its energy efficiency is lower than Enhanced-DeBiM's. Although the All-LSB system exploits the low-energy LSB writes, it is unable to exploit efficient MSB reads. The All-LSB system's performance is also lower than the All-MSB system's, showing a smaller energy efficiency improvement of 4.4% compared to the Conventional system. The Always-Low system assumes the energy savings of both MSB reads and LSB writes, marking a hypothetical energy

efficiency improvement of 29.8% over the Conventional system.

7.4 Memory Lifetime

Figure 15a shows the memory lifetime in years. We assume a cell endurance of 10^6 writes and a wear-leveling scheme that distributes writes throughout 16 GB of MLC PCM evenly. Two factors reduce the lifetimes of the DeBiM and Enhanced-DeBiM systems relative to the Conventional system. First, Decoupled Bit Mapping doubles the number of cells that are associated with a block, since a block takes on one bit (MSB or LSB) from each cell. Hence to write to a block, more cells need to be programmed (however, the probability of redundant bit-writes is lower in the Conventional system than in DeBiM [49, 15]). Second, DeBiM and Enhanced-DeBiM have higher system performance, which simply makes the two systems issue more writes per second.

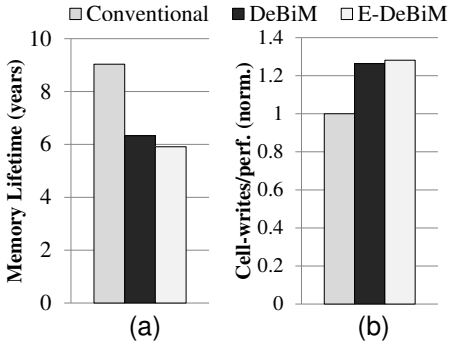


Figure 15: Memory system lifetime and cell-writes normalized to system performance.

To filter out the effect of system performance and isolate the effect of Decoupled Bit Mapping on memory lifetime, we normalize the total number of cell-writes by performance in Figure 15b. The overhead for the DeBiM and Enhanced-DeBiM systems are 26.4% and 28.2% respectively. We believe it is worthwhile to make this endurance tradeoff for performance, as recent work reports that very high PCM endurance is possible ($> 10^{15}$ writes) [20]. The observation is that the asymmetric properties in MLC read and write will continue to exist (due to the multi-level sensing required for reads and the complex programming process required for writes), while PCM technology improves in terms of endurance.

7.5 Sensitivity to Last-level Cache Size

Figure 16 shows the sensitivity of instruction throughput to differing last-level cache sizes (plots are normalized to the Conventional scheme in the 8 MB last-level cache size). The trends for throughput improvement are similar for the different last-level cache sizes (as are the trends for speedup). Relative to the Conventional scheme, DeBiM improves instruction throughput by 14.6%, 12.8%, and 11.6% for the 8 MB, 16 MB, and 32 MB last-level cache sizes respectively. For Enhanced-DeBiM, the improvements are 23.3%, 19.6%, and 18.3% for the same last-level cache sizes, outperforming the All-MSB (18.6%, 16.1%, and 15.4%) and All-LSB (10.7%, 9.6%, and 7.7%) scenarios. The Always-Low system improvements are 34.5%, 30.2%, and 30.3%. With increasing last-level cache size, the performance improvement decreases, as more requests for data are served by the last-level cache and less by the memory subsystem.

7.6 Sensitivity to Row Size

Figure 17 shows the sensitivity of speedup of the proposed DeBiM, AsPaM, and SplitRB schemes to differing row sizes (plots are normalized to the Conventional scheme for each row size). All of the schemes consistently show speedup improvement over the Conventional scheme. Across the four different row sizes examined (1k, 2k, 4k, and 8k cells in a row), DeBiM shows an average 10.1% speedup improvement over the Conventional scheme. Building on top of DeBiM, AsPaM shows higher speedup improvement (13.4% over Conventional) than SplitRB (12.4% over Conventional). Combining all of these schemes, Enhanced-DeBiM gives a speedup improvement of 16.0% over the

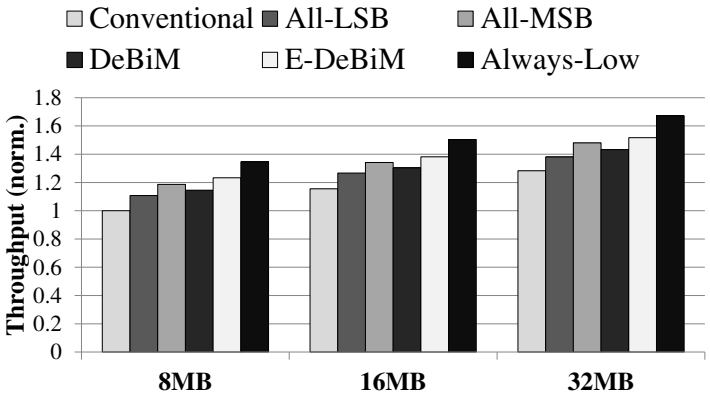


Figure 16: Performance sensitivity to different last-level cache sizes.

Conventional scheme, outperforming the All-MSB (13.2%) and All-LSB (7.0%) scenarios. The hypothetical Always-Low scenario gives the highest 28.1% speedup improvement.

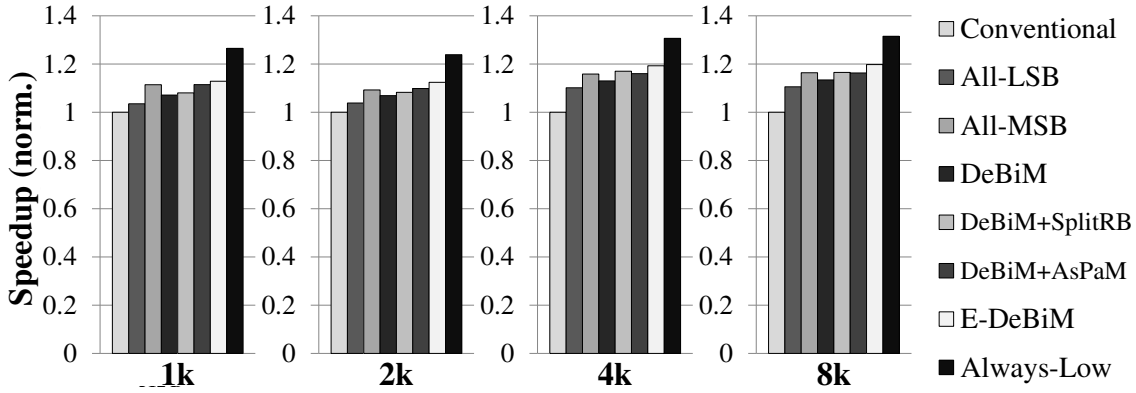


Figure 17: Performance sensitivity to different row sizes.

8 Related Work

Related Concepts in other memory technologies. The high-speed programming method in MLC Flash memory requires mapping the bits of an MLC cell to separate logical addresses. Although this resembles DeBiM, the two are fundamentally different. High-speed programming in Flash is designed to reduce the number of intermediate read verifications between successive programming iterations [41]. On the other hand, DeBiM is motivated by MLC PCM's read and write asymmetry properties, which are exploited to expose the lower latency and energy of MSB reads and LSB writes. Jiang et al. [14] propose mapping the bits of an MLC STT-MRAM cell to separate logical addresses to increase the performance of on-chip caches in embedded processors. Their mechanism migrates data to favorable memory locations depending on the observed type of accesses (reads or writes) frequently issued to the data, whereas AsPaM predicts whether data are read- or write-intensive before allocating them to memory locations.

Multi-Level Cell PCM. Qureshi et al. [33] proposed dynamically adapting the usage of PCM either as high-density high-latency MLC memory or as low-density low-latency SLC memory, depending on the memory requirements of the application. This work assumes that a system has over-provisioned memory capacity, and trades away surplus capacity for reduced latency. Our proposal reduces the latency while maintaining the high density of MLC PCM, through data mapping that exploits lower bit latency and enables fine-granularity row buffering. This does not prevent the memory from being configured as MLC or SLC, and is thus orthogonal work.

Other works have introduced ways to mitigate the high write latency of MLC PCM. Write cancellation and pausing allow reads to be served faster by interrupting long write operations [32]. Jiang et al. [15] proposed write truncation

that improves write latency by foregoing the programming of the few slower bits in a row (or block), compensating for the loss in data integrity with stronger ECC. They also proposed to store data in SLC form if the data and associated ECC bits can altogether be compressed to less than half of the original data size. These studies primarily focus on alleviating the high MLC PCM write latency (with read enhancements depending upon the compressibility of the data), and both are orthogonal to our proposals.

The practicability of MLC PCM technology has been demonstrated by various works [2, 6, 28]. Architectural solutions have been proposed to prolong memory lifetime through wear-leveling [49, 30, 5], enhancing fault-tolerance [13, 38, 45], data buffering [31, 22], and programming-current provisioning [46]. These schemes are generic to SLC and MLC PCM, and can be used in conjunction with our proposals.

Multiple Row Buffers. Many prior studies [12, 47, 24, 22] evaluated the use of multiple row buffers, and showed that it generally leads to increased row buffer locality and memory performance gains. However, when resizing the length of each row buffer (e.g. for the purpose of reducing or maintaining the die area occupied by row buffers), [12, 47, 22] show that it harms performance if the row buffers are too short. This is because shorter row buffers exploit less spatial locality. In other words, for a constant die area dedicated to row buffers, there is a point where increasing the number of row buffers at the expense of having them shorter is no longer desirable.

We propose a fine-granularity row buffer organization that divides the row buffer between the bits of an MLC cell (i.e. between the MSB half-row and the LSB half-row), instead of dividing the row buffer along the column direction. For DeBiM where the bits of a cell are mapped to logically separate memory addresses, our scheme enables a fine-granularity row buffer organization while not diminishing spatial locality.

9 Conclusion

We presented new data mapping and buffering mechanisms that enable higher performance and energy efficiency by exploiting characteristics of multi-level cell main memory technology. Our proposed approach consists of two key ideas. First, we develop a hardware-software cooperative technique to map data in MLC PCM in a way to exploit the asymmetric read and write characteristics between the bits of an MLC PCM cell. Decoupled Bit Mapping exposes reduced latency and energy present in different bits (i.e., for MSB reads and LSB writes) to system software. The system software then exploits this asymmetry by mapping read-intensive pages to MSBs and write-intensive pages to LSBs. Second, we exploit the flexibility in row buffering enabled by Decoupled Bit Mapping to provide two logical row buffers from a single physical row buffer. This increases row buffer locality, enhancing both memory performance and energy efficiency. Our evaluations for a multi-core system show that the proposed techniques provide better system performance, energy efficiency, and thread fairness compared to the conventional MLC PCM baseline. We conclude that our proposal provides an effective way of taking advantage of MLC technology in main memory and envision that it is applicable to other emerging technologies with similar characteristics.

References

[1] J. Ahn, J. Leverich, R. S. Schreiber, and N. Jouppi. Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs. *CAL*, 2008.

[2] F. Bedeschi et al. A Bipolar-Selected Phase Change Memory Featuring Multi-Level Cell Storage. *JSSC*, 44(1), 2009.

[3] M. Bender et al. Flow and stretch metrics for scheduling continuous job streams. In *SODA*, 1998.

[4] B. Buros. Tuning and optimizing malloc on PowerLinux. *IBM developerWorks (PowerLinux Architecture)*, 2012.

[5] S. Cho and H. Lee. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *MICRO*, 2009.

[6] G. Close et al. A 512Mb phase-change memory (PCM) in 90nm CMOS achieving 2b/cell. In *VLSIC*, 2011.

[7] G. Dhiman et al. PDRAM: a hybrid PRAM and DRAM main memory system. In *DAC*, 2009.

[8] W. Doerner. Memory Allocation and First-Touch. *Intel Developer Zone*, 2012.

[9] X. Dong, C. Xu, Y. Xie, and N. Jouppi. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *TCAD*, 2012.

[10] T. Happ et al. Novel One-Mask Self-Heating Pillar Phase Change Memory. In *VLSIT*, 2006.

[11] M. Hosomi et al. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. In *IEDM*, 2005.

[12] W.-C. Hsu and J. Smith. Performance Of Cached DRAM Organizations In Vector Supercomputers. In *ISCA*, 1993.

[13] E. Ipek et al. Dynamically replicated memory: building reliable systems from nanoscale resistive memories. In *ASPLOS*, 2010.

[14] L. Jiang et al. Constructing large and fast multi-level cell STT-MRAM based cache for embedded processors. In *DAC*, 2012.

[15] L. Jiang et al. Improving write operations in MLC phase change memory. In *HPCA*, 2012.

[16] M. Joshi et al. Mercury: A fast and energy-efficient multi-level cell based Phase Change Memory system. In *HPCA*, 2011.

[17] R. Kalla et al. Power7: IBM's Next-Generation Server Processor. *Micro, IEEE*, 30(2), 2010.

[18] P. Kaminski. NUMA aware heap memory manager. *AMD Developer Central*, 2009.

[19] S. Kang et al. A 0.1-um 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) With 66-MHz Synchronous Burst-Read Operation. *JSSC*, 42(1), 2007.

[20] I. Kim et al. High performance PRAM cell scalable to sub-20nm technology with below 4F2 cell size, extendable to DRAM applications. In *VLSIT*, 2010.

[21] H. Kwak et al. What is Twitter, a social network or a news media? In *WWW*, 2010.

[22] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 2009.

[23] C. J. Lee et al. Prefetch-Aware DRAM Controllers. In *MICRO*, 2008.

[24] G. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *ISCA*, 2008.

[25] J. Meza et al. Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management. *CAL*, 2012.

[26] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*, 2007.

[27] T. Nirschl et al. Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory. In *IEDM*, 2007.

[28] N. Papandreou et al. Drift-Tolerant Multilevel Phase-Change Memory. In *Intl. Memory Workshop (IMW)*, 2011.

[29] H. Patil et al. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. In *MICRO*, 2004.

[30] M. K. Qureshi et al. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO*, 2009.

[31] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[32] M. K. Qureshi et al. Improving read performance of Phase Change Memories via Write Cancellation and Write Pausing. In *HPCA*, 2010.

[33] M. K. Qureshi et al. Morphable memory system: a robust architecture for exploiting multi-level phase change memories. In *ISCA*, 2010.

[34] M. K. Qureshi et al. PreSET: Improving performance of phase change memories by exploiting asymmetry in write times. In *ISCA*, 2012.

[35] S. Raoux et al. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.*, 52, 2008.

[36] S. Rixner et al. Memory access scheduling. In *ISCA*, 2000.

[37] B. Rogers et al. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In *ISCA*, 2009.

[38] S. Schechter et al. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA*, 2010.

[39] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. In *ASPLOS*, 2000.

[40] D. B. Strukov et al. The missing memristor found. *Nature*, 2008.

[41] K. Takeuchi et al. A multipage cell architecture for high-speed programming multilevel NAND flash memories. *JSSC*, 1998.

[42] A. N. Udipi et al. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *ISCA*, 2010.

[43] T. Vogelsang. Understanding the Energy Consumption of Dynamic Random Access Memories. In *MICRO*, 2010.

[44] H. Wong et al. Phase Change Memory. *Proc. of the IEEE*, 2010.

[45] D. H. Yoon et al. FREE-p: Protecting non-volatile memory against both hard and soft errors. In *HPCA*, 2011.

[46] W. Zhang et al. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO*, 2009.

[47] Z. Zhang et al. Cached DRAM for ILP processor memory access latency reduction. *Micro, IEEE*, 2001.

[48] H. Zheng et al. Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency. In *MICRO*, 2008.

[49] P. Zhou et al. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.

[50] W. K. Zuravleff and T. Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. In *U.S. Patent Number 5,630,096*, 1997.