

FPB: Fine-grained Power Budgeting to Improve Write Throughput of Multi-level Cell Phase Change Memory

Lei Jiang † Youtao Zhang § Bruce R. Childers§ Jun Yang †

† Electrical and Computer Engineering Department

§ Department of Computer Science

University of Pittsburgh

†{lej16, juy9}@pitt.edu §{zhangyt, childers}@cs.pitt.edu

Abstract

As a promising nonvolatile memory technology, Phase Change Memory (PCM) has many advantages over traditional DRAM. Multi-level Cell PCM (MLC) has the benefit of increased memory capacity with low fabrication cost. Due to high per-cell write power and long write latency, MLC PCM requires careful power management to ensure write reliability. Unfortunately, existing power management schemes applied to MLC PCM result in low write throughput and large performance degradation.

In this paper, we propose Fine-grained write Power Budgeting (FPB) for MLC PCM. We first identify two major problems for MLC write operations: (i) managing write power without consideration of the iterative write process used by MLC is overly pessimistic; (ii) a heavily written (hot) chip may block the memory from accepting further writes due to chip power restrictions, although most chips may be available. To address these problems, we propose two FPB schemes. First, FPB-IPM observes a global power budget and regulates power across write iterations according to the step-down power demand of each iteration. Second, FPB-GCP integrates a global charge pump on a DIMM to boost power for hot PCM chips while staying within the global power budget. Our experimental results show that these techniques achieve significant improvement on write throughput and system performance. Our schemes also interact positively with PCM effective read latency reduction techniques, such as write cancellation, write pausing and write truncation.

1. Introduction

Phase Change Memory (PCM) has emerged as a leading technology to alleviate the leakage and scalability problems of traditional DRAM [24]. With advantages over DRAM, such as near zero cell leakage, better scalability and comparable read speed, PCM is poised to replace a significant portion of DRAM in main memory [12, 19, 31]. A PCM cell uses different resistances to represent logic bits. Single-level cell PCM (SLC) differentiates between two resistance levels to store a bit (logic ‘0’ or ‘1’). Due to the large resistance contrast between ‘0’ and ‘1’, intermediate levels can be used to store multiple bits per cell in multi-level cell PCM (MLC).

Although memory capacity is effectively increased with low per bit fabrication cost in MLC PCM, this technology has shorter write endurance, longer access latency, and larger write power than SLC PCM. Many schemes have been proposed to address some of these issues. In addition to schemes for SLC PCM [12, 18, 19, 25, 26, 31], Qureshi *et al.* proposed to transform MLC PCM pages to SLC pages for fast access [21]; Qureshi *et al.* proposed to pause MLC write operations and prioritize read operations to improve performance [20]; Jiang *et al.* proposed write truncation to reduce average MLC write time and use ECC to correct write errors [10]. Joshi *et al.* proposed an energy-efficient programming scheme for MLC PCM [11].

While past research has made significant strides, high PCM write power remains a major obstacle to improving throughput. For example, a recent study showed that the power provided by DDR3-1066×16 memory allows only 560 SLC PCM cells to be written in parallel [8], i.e., at most two 64B lines can be written simultaneously using Flip-n-Write [4]. Hay *et al.* proposed to track the available power budget and issue writes continuously as long as power demands can be satisfied [8]. Their heuristic works well for SLC PCM based main memory.

Unfortunately, applying the heuristic to MLC PCM results in low write throughput and large performance degradation: On average, we observed a 51% performance degradation over an ideal baseline without power limit. We identified two major problems for MLC PCM that limit throughput and performance for this heuristic.

The first problem is that allocating the same power budget for the entire duration of an MLC line write is often too pessimistic. A MLC PCM write is done in iterations, starting with a RESET pulse and followed by a varying number of SET pulses. The RESET pulse is short and of large magnitude while the SET pulse is long and of low magnitude. In addition, when writing one PCM line, most cells in the line require only a small number of SET pulses [10]. Allocating power according to the RESET power request and for the duration of the longest cell write is power inefficient.

The second problem is that one heavily written (hot) PCM chip may block the memory subsystem even though most memory chips are idle. This phenomenon arises because the power that each chip can provide is restricted by the area of its charge pump. When multiple writes compete for a single chip, some writes have to wait to avoid exceeding the charge pump’s capability. Otherwise, cell writes become unreliable.

We propose two new fine-grained power budgeting (FPB) schemes to address these problems:

- **FPB-IPM** is a scheme that regulates write power on each write iteration in MLC PCM. Since writing one MLC line requires multiple iterations with step-down power requirements, FPB-IPM aims to (i) reclaim any *unused* write power after each iteration and (ii) reduce the maximum power requested in a write operation by splitting the first RESET iteration into several RESET iterations. By enabling more MLC line writes in parallel, FPB-IPM improves memory throughput.
- **FPB-GCP** is a scheme that mitigates power restrictions at chip level. Rather than enlarging the charge pump in an individual chip, FPB-GCP integrates a single global charge pump (GCP) on a DIMM. It dynamically pumps extra power to hot chips in the DIMM. Since GCP has lower effective power efficiency (i.e., the percentage of power that can be utilized for writes), we consider different cell mapping optimizations to maximize throughput.

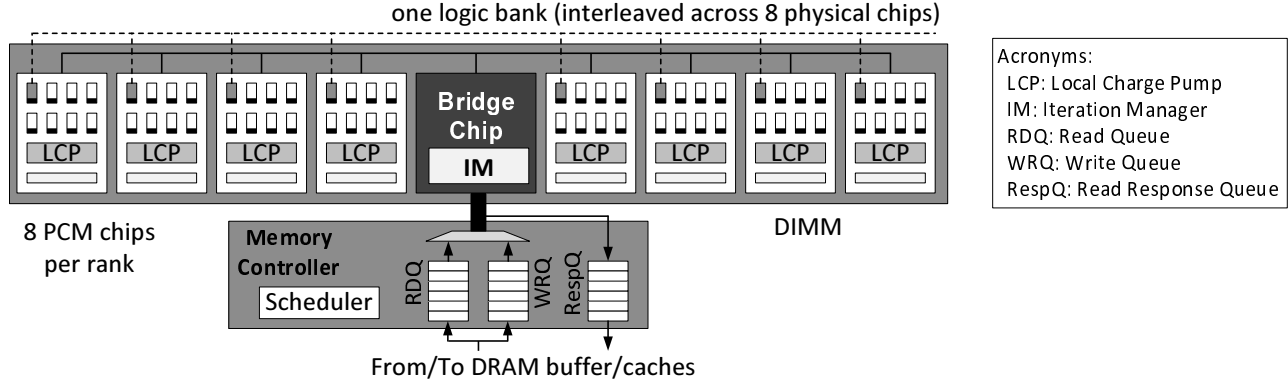


Figure 1: The baseline architecture of a MLC PCM-based memory subsystem (One DIMM).

We evaluate our proposed schemes and compare them to state-of-the-art PCM power management. Our results show that FPB-IPM and FPB-GCP are orthogonal designs that together successfully address the problems described above for MLC write operations. On average, FPB achieves a $3.4\times$ throughput improvement and 76% performance improvement over state-of-the-art power budgeting.

The rest of the paper presents background and motivation for our designs in Section 2. FPB-IPM and FPB-GCP are described in Sections 3 and 4. We present our experimental methodology and analyze results in Section 5 and 6 respectively. Related work is discussed in Section 7. Section 8 concludes the paper.

2. Background and motivation

In this section, we first discuss a typical MLC PCM memory architecture and details of MLC write operations. Next, we motivate our designs by analyzing how simple power management heuristics behave for MLC PCM.

2.1. MLC PCM memory architecture

Our baseline architecture of a MLC PCM memory subsystem is shown in Figure 1. Similar to a traditional DRAM organization, a DIMM has eight memory chips (PCM) that are organized into eight logical banks. Due to non-deterministic MLC PCM write characteristics [10], we adopt the *universal memory interface* design proposed by Fang *et al.* [7]. In Figure 1, device control is performed collaboratively between the on-CPU memory controller and the on-DIMM bridge. The memory controller’s scheduler issues requests in the read queue (RDQ) and write queue (WRQ) according to bus availability, bank availability, circuit timing constraints, and global DIMM and local chip power budgets. Completed read requests (from MLC PCM banks) wait in the read response queue (RespQ) until the bus or interconnect is available at which point the read data is sent back to the cores [7].

Memory interface: a *universal memory interface* [7] makes PCM timing and device-specific management issues transparent to memory controller. Instead of memory controller, a bridge chip on DIMM tracks the status of each DIMM and ongoing access operations. A new protocol is proposed to avoid conflicts on the shared data bus and to reply memory controller when requested data is ready. In this paper, we adopted this design to handle the communication between memory controller and bridge chip and leave PCM DIMM/chip management to bridge chip.

Different cell stripping methods: In this paper, we strip cells from one memory line across all chips in our baseline configuration, so that we can access all cells in one memory line in one round. There are two design alternatives:

- Stripping cells across half of the chips, and accessing one line in one round. Each chip handles twice as many cells and requires wider bus/peripherals. This is similar to chopping each chip into two sub-chips, or simply doubling the number of chips and using only half of them for one access. Our techniques can be applied to either case.
- Stripping cells across half of the chips and accessing one line in two rounds. Each chip handles the same number of cells as stripping cells across all chips. However, the read and write latency to memory array is doubled, which will harm system performance.

2.1.1. Non-deterministic MLC write MLC PCM devices widely adopt *program-and-verify* (P&V) [2, 15] to ensure programming (write) accuracy. For a given PCM line write, only a subset of cells in the line need to be changed [4, 31]. For these cells, the write circuit first injects a RESET pulse with large voltage magnitude to place them in similar states, and then injects a sequence of SET pulses with low voltage magnitudes. After each SET pulse, a read/verify operation is performed. A cell write is terminated when its target MLC resistance level is reached. The line write finishes when all cell writes are completed.

Due to process variations and material fluctuation [3, 14], non-determinism arises for MLC PCM writes. The cells comprising a MLC PCM line can take a varying number of iterations to finish (e.g., one cell might take a few iterations, while another may take the worst case number). Further complicating cell programming, the *same* cell may require a different number of iterations to finish for different write instances. Studies have shown that most cells finish in only a small number of iterations [20]. Jiang *et al.* proposed write truncation to speed up MLC write accesses [10].

To handle non-deterministic MLC PCM writes, it is beneficial to divide PCM device control between the memory controller and the bridge chip. Fang *et al.* evaluated the details of this division [7]. If an approach similar to DRAM is employed, i.e., the on-CPU memory controller does all device control, the memory controller may have to assume that all MLC write operations take the worst case number of iterations, which greatly degrades performance.

2.1.2. DIMM power budget PCM requires much higher per-cell write power than DRAM. Hay *et al.* calculated that the power provided by a typical DDR3-1066 \times 16 DRAM memory allows up to 560 SLC PCM simultaneous cell writes. In comparison, a single DRAM refresh round can simultaneously write one 2KB row, or 16,384 DRAM cells.

The DIMM power budget is a critical parameter in a PCM memory subsystem as it restricts the number of simultaneous cell changes.

Figure 2 reports the average number of cell changes per PCM line write under different configurations for 2-bit MLC.¹

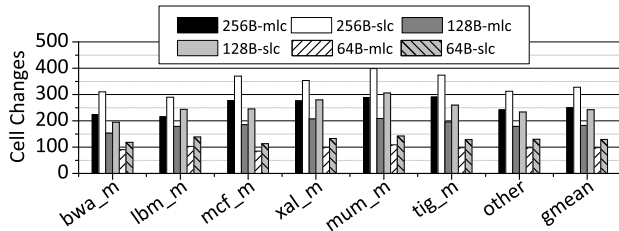


Figure 2: The cell changes under different settings.

According to Figure 2, 2-bit MLC tends to change a smaller number of cells than SLC. In addition, a larger line size results in more cell changes. In this paper, we assume the power budget per DIMM can support 560 MLC cell writes, which is the same number for SLC cell changes in previous work [8]. This represents a relaxed DIMM budget as MLC often needs more write power [11]. To explore configurations with different cell changes and thus power budget demands, we perform a wide design space exploration with different line sizes and power budgets. This also addresses the designs that use different write row buffer sizes at the device level [12].

Note that in future memory subsystems, the DIMM power budget is unlikely to increase significantly. First, PCM based main memory tends to be big to support large scale parallelizable workloads [19], which limits the budget available to a DIMM. Second, recent years have seen the need for low power DIMMs [13, 29].

2.1.3. Chip level power budget Another power restriction is the chip-level power budget. Since PCM writes require higher voltages than V_{dd} , PCM chips integrate CMOS-compatible charge pumps [6, 17] to supply required voltage and power. Studies have shown that the area of a charge pump is proportional to the maximum current that it can provide [17]:

$$A_{\text{tot}} = k \cdot \frac{N^2}{(N+1) \cdot V_{dd} - V_{\text{out}}} \frac{I_L}{f} \quad (1)$$

Here, A_{tot} is the total area overhead of the charge pump. k is a constant that depends on the process used to realize the capacitors. N indicates the number of stages in the charge pump. V_{dd} is the supply voltage and V_{out} is the target programming voltage. f denotes the charge pump's working frequency. I_L is the total write current.

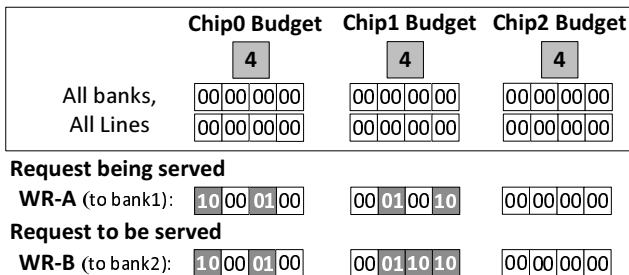


Figure 3: Writes blocked by chip level power budget (assuming three chips/bank for discussion purpose).

The write throughput of MLC PCM may be constrained by a chip power budget. In Figure 3, we assume (i) one bank spreads across

three chips; (ii) the memory initially contains all 0s; (iii) the chip power budget can support 4 cell changes; (iv) the system is serving request WR-A when request WR-B arrives. They write to different banks and change 4 and 5 cells respectively (shown as shaded boxes with white font).

While these two writes change 9 cells in total and the DIMM power budget allows 12 cell changes, WR-B cannot be issued as the sum of cell changes for chip 1 is 5, which is larger than the chip's budget. If WR-B is issued, both writes may fail as there is not enough power for reliable programming.

A typical charge pump occupies 15% to 20% of a PCM chip's area [16]. Thus, it is undesirable to enlarge the charge pump to increase its maximum output current/power.

2.1.4. Our model In this paper, we adopt two-phase modeling [10, 21] for MLC PCM writes. For the DIMM power limit, we adopt the same one as past work [8]. To get the default chip power limit, we divide the DIMM power limit by 8 (i.e., eight chips per DIMM and the sum of the chip power limits equal the DIMM power limit). Our experiments consider an extensive design space. The results show that our schemes are independent of concrete model parameters. The designs are robust under a wide range of configurations.

2.2. Design motivation

To evaluate the impact of DIMM and chip power budgets for MLC PCM, Figure 4 compares several simple power management heuristics. The results are normalized to Ideal, which is a scheme that does not restrict power, i.e., a MLC write can be issued whenever a requested bank is idle.¹

In Figure 4, DIMM-only is a case where only the DIMM power limit is enforced. There is no chip power budget, i.e., a MLC write can be issued whenever the DIMM has enough power to satisfy the write's power demand. DIMM-only adopts Hay et al.'s power management heuristic [8] to prevent the DIMM from drawing too much power. From the figure, the heuristic incurs 33% performance loss for MLC PCM, which is significantly worse than the small 2% loss for SLC PCM [8]. The reason for this discrepancy is DIMM-only does not consider MLC write iterations and it allocates the same power for the full duration of a complete line write. However, the maximum power demand happens only in the first iteration of the MLC PCM write: (i) RESET power is much larger than SET power; and, (ii) many MLC cells finish in a small number of iterations. Clearly this heuristic is overly pessimistic by budgeting the maximum write power for a line for the entire duration of the longest cell write.

Figure 4 also illustrates the impact of a PCM chip power budget. DIMM+chip uses the same heuristic as DIMM-only but it enforces both DIMM and chip power budgets. On average, there is a 51% performance loss. The increased loss over DIMM-only (i.e., the portion beyond DIMM-only's 33% loss) is due to the chip power budget. When several writes compete for a busy chip, some writes must wait to avoid exceeding the chip power budget, even though the DIMM power budget may not be exceeded. Violating the chip power budget leads to unreliable MLC writes.

To alleviate this problem for an individual chip, we tried three schemes. First, we tried to remove power competition at the chip level. PWL is an enhanced heuristic that adopts overhead-free near-perfect intra-line wear leveling. Since the lower order bits within a data block (words, double words, etc.) are more likely to be changed, intra-line wear leveling has been proposed to balance bit changes across all chips to *extend lifetime* [31]. We used intra-line wear-leveling to balance *write power requests* across chips. We assume

¹The simulation framework and parameters are summarized in Section 5.

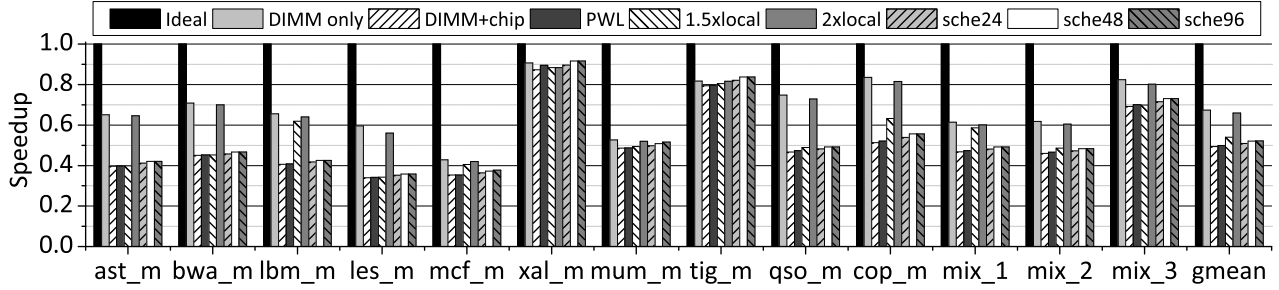


Figure 4: The performance under power restrictions for MLC PCM.

that each line is shifted by a random offset after every 8 to 100 writes and report the best results. From the figure, PwL achieves approximately a 2% improvement over DIMM+chip. We also tried different cell mapping schemes (i.e., cells are interleaved across chips) but observed similar small gains.

Second, we increased the chip’s maximum power: 1.5xlocal and 2xlocal increased the chip’s power budgets by 50% and 100%, respectively. From Figure 4, if the charge pump can provide 2× power, the performance loss relative to DIMM-only is negligible. Note, we have shown that the loss from Ideal to DIMM-only is due to iteration-oblivious power budgeting. The results show that the fluctuation in chip power demand is below 2× on average. However, for 50% more power, the loss is still significant, on average 20% loss. Increasing the maximum power is effective but has large area overhead.

Finally, we scheduled writes in the write queue (out-of-order) based on chip power availability. Sche-X is this scheme with an X-entry write queue. Figure 4 shows that a large write queue has little effect in mitigating performance loss.

To summarize, high MLC write power demand has a large performance impact. It cannot be resolved by state-of-the-art power management heuristics and/or simple adjustments at different levels.

3. FPB-IPM: iteration power management

In this section, we describe FPB-IPM, an iteration power management scheme for MLC PCM. For discussion purposes, we consider only the DIMM power budget in this section. The chip power budget is considered in the next section.

Figure 5 illustrates how FPB-IPM works. The scheme is token driven: in order for writes to proceed, there must be enough power tokens available to satisfy the number of bit changes required by a write. Each token represents the power for a single cell RESET. Assume that (i) two writes WR-A and WR-B arrive at the bridge chip and request to change 50 and 40 cells, respectively; and, (ii) the available DIMM power budget can support a RESET on 80 cells simultaneously, i.e., there are 80 available power tokens (APT).

Consider a simple per-write power management heuristic, as shown in Figure 5(a). This heuristic tracks APT with a counter, and releases a write only when there are enough unused tokens. Since WR-A arrives first and it requests fewer tokens than the DIMM’s budget of 80 tokens, WR-A is served immediately. As a consequence, APT is reduced to 30 until WR-A finishes. In this case, WR-B stalls until WR-A returns its tokens. From Figure 5(a), the write throughput is low as the two writes do not overlap. However, some of the tokens allocated to WR-A are not actually used. For example, in the fourth iteration of WR-A, a SET is done to only 6 cells, and thus, only 3 tokens are used (SET power is half of RESET power). Nevertheless, WR-A holds all 50 tokens until the write is finished.

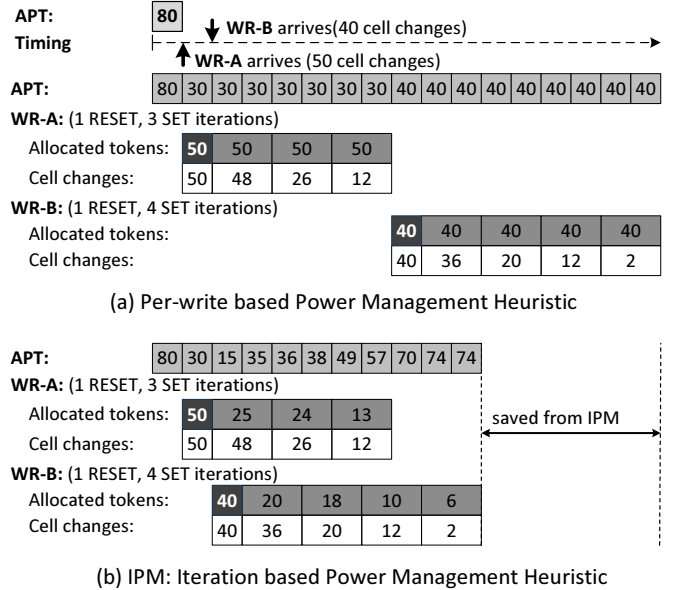


Figure 5: FPB-IPM: iteration power management (assuming SET power is half of RESET power and RESET pulse is half the length of SET pulse).

To resolve this problem, we designed FPB-IPM to reclaim unused power tokens as early as possible, which increases the number of simultaneous writes. Figure 5(b) illustrates our improved scheme. In this scheme, FPB-IPM first allocates power tokens to incoming write requests (e.g., WR-A) if there are enough ones. This is similar to the simple per-write management heuristic in Figure 5(a).

Next, after the first RESET iteration, FPB-IPM reclaims $((C-1)/C) \times PT_{RESET}$ tokens, where $RESET_power = C \times SET_power$ and PT_{RESET} is the number of tokens allocated in the first iteration. For example, half of the allocated tokens are reclaimed in write iteration 2, as shown in Figure 5(b). Because a MLC write operation finishes in a non-deterministic number of iterations, the number of cells that need to be written decreases after each SET iteration. The consumed write power also drops as the write operation proceeds. Thus, FPB-IPM also reclaims tokens after SET iterations. To reclaim unused tokens as early as possible, FPB-IPM dynamically adjusts the power token allocation on each iteration.

Starting from the 3rd iteration, FPB-IPM allocates write tokens based on cell changes in preceding iterations. In Figure 5(b), 24 tokens are allocated for the 3rd iteration of WR-A, which can SET 48 cells. This is enough tokens. Because the 2nd iteration changes 48 MLC cells, it is impossible to change more than 48 cells in the 3rd iteration and beyond.

3.1. Architecture enhancement

To enable iteration power management, FPB-IPM needs to know how many cells will be changed in each iteration. Hay *et al.* tracks SLC cell changes in the last-level cache [8]. However, this approach cannot be applied to FPB-IPM as MLC writes are non-deterministic and FPB-IPM regulates the power tokens at iteration granularity.

FPB-IPM integrates the power management logic in the bridge chip and includes two enhancements. One enhancement does a read before a write operation. The old data is compared with the new data to determine how many cells will be changed. This is slightly more expensive than *differential-write* [31] and *Flip-n-Write* [4] as these schemes perform the comparison inside the PCM chip. In FPB-IPM, the extra read increases bus contention within the DIMM. However, the read does not compete for the bus between the DIMM and the memory controller, which is a more precious resource in a multiple-DIMM memory subsystem. In the experiments, we model the cost of doing the full read before each write.

The other enhancement is each PCM chip reports the number of cells that finish after the verification operation in each write iteration. This helps FPB-IPM reclaim unused power tokens. The allocation for write iteration i , where $i \geq 3$, is determined by the number of cell changes that remain after iteration $i - 2$. This value can be computed during iteration $i - 1$ using the information reported by the PCM devices at the end of iteration $i - 2$. As a result, the allocation is available at the start of iteration i and the computation has no impact on write latency (overhead). For example, in Figure 5(b), 22 cells finished in the 2nd iteration of WR-A, which means 13 tokens are allocated in iteration 4 (i.e., $13 = (2-1)/2 \times (48-22)$).

3.2. Multi-RESET

By reclaiming unused power tokens after each iteration, the available power tokens accumulate fast. However, due to the large ratio between RESET and SET power, a write is often blocked because there are not enough tokens for the write’s RESET iteration. If this iteration had a lower power demand, then the write would be more likely to go ahead without delay.

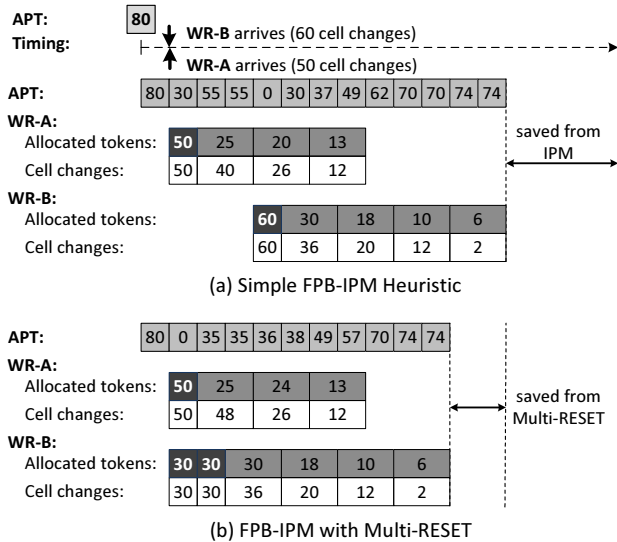


Figure 6: Multi-RESET reduces maximum power demand.

Based on this observation, we propose Multi-RESET, a technique that breaks a write’s RESET iteration into several RESET iterations.

Only a subset of cells are RESET in each iteration. After all cells are reset, the write does the normal SET iterations. By reducing the maximum power demand, Multi-RESET has the potential to enable more simultaneous writes. The disadvantage is increased write latency: if the RESET iteration is split into m RESET iterations, then the write latency increases by $m-1$ RESET iterations.

Figure 6 shows how Multi-RESET works. After issuing WR-A, the APT is 30. Since WR-B requests 60 power tokens, it has to wait until there are enough tokens (Figure 6(a)). By adopting Multi-RESET, WR-B splits the single, power-expensive RESET iteration into two less power-expensive iterations. Each iteration does a RESET for 30 cells. With this strategy, WR-B can be issued immediately. In this way, WR-A and WR-B have more overlap, resulting in improved write throughput (Figure 6(b)).

Implementing Multi-RESET requires that cells are grouped carefully. There are two approaches. One approach groups cells based on the cells to be changed. The other groups cells no matter if they are changed or not. The former tends to perform better while the latter has lower hardware overhead. In this paper, we choose the latter scheme and split cells from one chip into three groups. This requires a 2-bit control signal to a PCM chip to enable individual groups (‘11’ indicates all groups are RESET in one iteration).

Comparison. Multi-RESET shares similarity with *write pausing* [20], which pauses MLC writes to prioritize reads. Multi-RESET stalls the cells written in early RESET iterations until all cells to be changed are RESET. However, the design goal is different. Multi-RESET aims to lower the maximum power demand while *write pausing* aims to improve read performance. Due to the short latency pause after RESET, MLC resistance drift [30] can be ignored.

Multi-RESET also shares similarity with a *multi-round write* operation. If the DIMM has 560 power tokens, it is impossible to write a 512B line when half of all cells must be changed (i.e., 1024 cells). In this scenario, the line is written in two rounds and each round writes 512 cells. The difference is that *multi-round write* breaks one write into two non-overlapped writes, which doubles the write latency. Multi-RESET has much less latency overhead.

4. FPB-GCP: mitigating chip power restrictions by a global charge pump

In this section, we propose using a global charge pump (GCP) to mitigate performance loss due to a PCM chip’s power budget. We present the architecture details and design trade-offs.

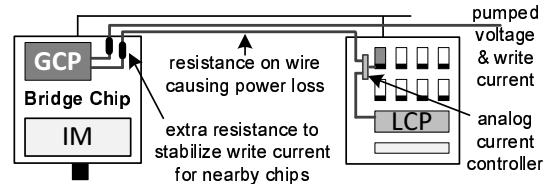


Figure 7: Integrating a global charge pump (GCP).

4.1. FPB-GCP Scheme

Section 2 describes how doubling the maximum power of the charge pumps in all chips on the DIMM can effectively eliminate the performance loss due to the chip power budget. This strategy incurs a large area overhead. Instead of making each local charge pump (LCP) in a PCM chip larger, we add a global charge pump (GCP)

into the bridge chip. As shown in Figure 7, the GCP resides in the bridge chip and uses a dedicated wire to supply the pumped voltage and write current to each PCM chip. Each bank segment (within a PCM chip) has an analog current controller to choose the write voltage from either the LCP or GCP (but not both). By default, the maximum power that the GCP can provide is set to the same power as one LCP.

While the GCP can provide extra power, the existing power budgets still need to be enforced: (i) the DIMM and chip power budgets must be obeyed and (ii) the DIMM and chip power budgets are not changed by introducing the GCP. In other words, the power that the GCP provides to one chip is actually “borrowed” from other chips.

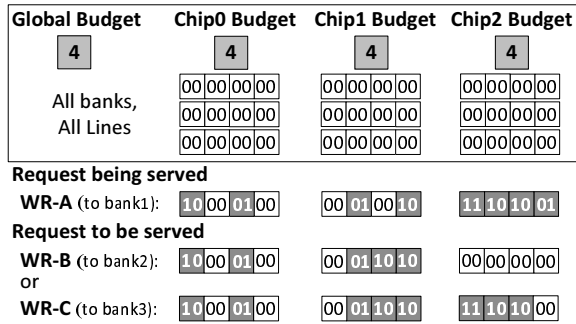


Figure 8: Schedule MLC PCM writes under FPB-GCP (assuming three chips/bank).

Figure 8 illustrates how FPB-GCP works. It has the same assumption as the earlier case in Figure 3. Now, the GCP has 4 power tokens. When WR-A is served, the available tokens are 2/2/0 for PCM chips 0/1/2 respectively. WR-B is chosen to be served next. Since it changes three cells for the 2nd segment (in chip 1), WR-B needs three tokens for chip 1. Given that only two tokens are available on chip 1, the write cannot be served using only the LCP. Thus, the GCP kicks in and injects extra power to write the segment in chip 1. Meanwhile, the LCP on chip 0 is used to write the first segment of WR-B: this segment asks for two tokens, which chip 0 has available.

In FPB-GCP, one segment uses either LCP or GCP, but not both. For example, it may still be impossible to serve WR-C and WR-A simultaneously because the GCP does not have enough tokens. Assume the GCP is used to write the 2nd segment of WR-C. Now, only one GCP token is available. Since WR-C changes three cells in its 3rd segment and chip 2 has no tokens available, the GCP is needed. However, WR-C cannot proceed as there are no enough GCP tokens.

In this example, the GCP might dynamically borrow tokens from chip 1 (has two available tokens). However, due to GCP power efficiency (discussed next), the two LCP tokens from chip 1 may correspond to only one GCP token. Thus, WR-A and WR-C still cannot be served simultaneously.

4.2. Power efficiency

An important parameter for a charge pump is its power efficiency, i.e., what percentage of input power can be utilized to write cells. Since LCP and GCP use the same CMOS-compatible charge pumps [6, 17], they have the same power efficiency by themselves. However, the wire from the GCP to the write driver is much longer than the wire from the LCP. The pumped voltage and write current from the GCP needs to travel a long distance before it is consumed. While wide wires can be used to reduce wire resistance, the long

distance will cause an inevitable power loss. Power loss is common even within one chip, e.g., Oh *et al.* observed around a 10% power loss within a PCM cell array [16]. To compensate for the loss, the GCP needs to output slightly larger current to ensure that the desired current can reach the farthest chip. This indicates a lower effective power efficiency. Given a limited number of pumping stages, the GCP may also need to add extra resistance to provide stable write current for nearby chips. A design alternative is to perform per-chip regulation to compensate different power losses between GCP and each chip, which tends to achieve better power efficiency at the cost of more complicated control logic. In addition, there is an efficiency loss from the pin to the write driver. Since the overall efficiency of the GCP depends on both technology and a combination these factors, the design of a highly power efficient GCP is beyond the scope of this paper.

To evaluate the effectiveness of GCP, we assume that the LCP has a 95% power efficiency, while the GCP has an effective power efficiency in the range [30%, 95%].

4.3. Cell mapping optimization

Due to low GCP power efficiency, the GCP wastes a non-negligible portion of input power. The more frequently the GCP is used, the more energy it wastes. Clearly, when two schemes have the same performance improvement, the one that uses the GCP less is preferred. In this section, we propose cell mapping optimizations to maximize throughput while minimizing GCP usage.

Our analysis shows that GCP usage is proportional to the imbalance of power demands at the chip level because the GCP “borrows” power tokens from the LCP. If all chips had exactly the same power demands, they would use up their power tokens at the same time, which leaves no tokens available to borrow. In practice, the imbalance exists due to memory access characteristics at the application level. For example, studies have shown that the lower-order bits of integer values are more likely to change. To minimize imbalance, and thus, the frequency of using the GCP, we study different mapping schemes that interleave cells across the chip.

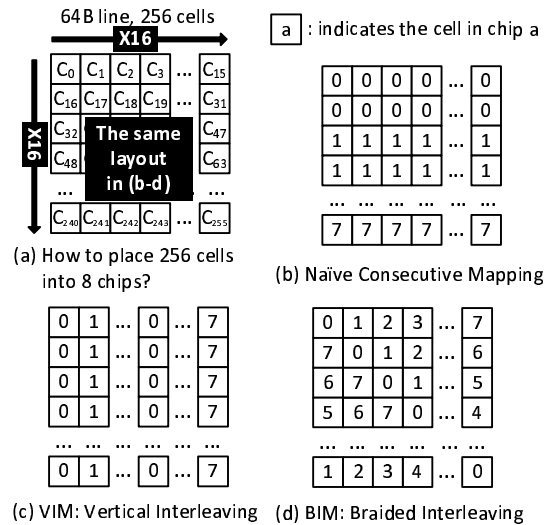


Figure 9: Different cell mapping schemes.

As shown in Figure 9(a), storing one 64B PCM line needs 256 2-bit MLC cells. We put 246 MLC cells into a 16×16 matrix layout

for illustration. The cell position keeps the same in Figure 9(b,c,d). A naïve mapping stores consecutive cells within one chip, e.g., the first 32 cells could be stored in chip 0 (Figure 9(b)).

For floating point (FP) programs, changing a FP value may lead to changing cells in one word (i.e., consecutively 16 logical cells), which incurs a request for more tokens from one chip. To distribute these changes, we propose Vertical Interleaving Mapping (VIM) that maps cells to chips as shown in Figure 9(c). The mapping function can be written as:

$$\text{chip_index} = \text{cell_index} \bmod 8 \quad (2)$$

For integer benchmark programs, the lower-orders bits in a word are more likely to change. To further balance cell changes, we propose Braided Interleaving Mapping (BIM) that distributes the lower-order cells from different words to different chips (Figure 9(d)). The mapping function is:

$$\text{chip_index} = \left(\text{cell_index} - \frac{\text{cell_index}}{16} \right) \bmod 8 \quad (3)$$

Cell mapping optimization overhead: BIM and VIM are static cell mapping optimization for distributing logic cells into different physical chips at the device level. On the contrary, intra-line wear leveling [31] periodically shifts a logical memory line by several bytes. Our cell mapping optimization is orthogonal to intra-line wear leveling: the inputs of BIM and VIM cell mapping functions are the outputs from intra-line wear leveling. Our schemes map logic MLC cell positions, instead of bit positions, to different chips. We observed that mapping bit positions by separating two consecutive bits in one cell would increase cell changes. BIM and VIM do not affect read procedure in PCM chip, since static cell mapping optimization does change row activation and wordline/bitline structure. Therefore, our schemes do not need any extra read/write dynamic energy. The cell mapping translation logic for 256 cells costs 87ps and $49\mu\text{W}$ at each access to PCM chip under 45nm technology.

5. Experimental methodology

5.1. Baseline configuration

To evaluate the effectiveness of our proposed schemes, we adopted the same simulation framework from [8] and compared our schemes to existing heuristics. The simulator is built as a PIN tool, which is used to collect long memory traces. Since our study focuses on memory subsystem performance and power characteristics, we used a memory trace-driven simulator (instead of a detailed pipeline simulator) to model accesses to and from MLC PCM main memory.

Our simulator faithfully models the entire memory hierarchy, including L1, L2 and DRAM last-level caches, the memory controller, and MLC PCM main memory. Several traces can be combined and interleaved by the simulator to create a multi-programmed workload. The simulator considers cache-to-cache and cache-to-memory bus contention, bank conflicts, and memory bus scheduling constraints. The memory controller gives higher priority to read requests. A write request is scheduled only when there is no read request. When the write queue is full, the memory controller schedules a write burst, which blocks any pending read requests until all the writes in the queue are finished. This strategy was also used by [8]. In addition to the normal bus and chip scheduling policies, writes can only be scheduled when there are enough available power tokens. We also consider the integration of our schemes with write cancellation, write pausing and write truncation.

CPU	8-core, 4GHz, single-issue, in-order
L1 I/D	private, I/D 32KB each/core, 64B line, 2-cycle hit
L2	private, 2MB/core, 4-way LRU, 64B line, write back 2-cycle tag, 5-cycle data hit, 16-cycle CPU to L2
DRAM L3	private, offchip, 32MB/core, 8-way LRU, write back 256B line, 50ns (200-cycle hit), 64-cycle CPU to L3
Memory Controller	onchip, 24-entry R/W queues, MC to bank 64-cycle scheduling reads first, issuing writes when there is no read, issuing write burst when W queue is full [8]
PCM Main Memory	4GB, the same line size as L3, single-rank 8 banks, MLC read 250ns (1000 cycles) RESET: 125ns (500 cycles), $300\mu\text{A}$, 1.6V, $480\mu\text{W}$ SET: 250ns (1000 cycles), T_{off} [9] included, $150\mu\text{A}$ 1.2V, $90\mu\text{W}$ [12], MLC Write Model: 2-bit MLC[20, 10] '01': $i/F1/F2 = 2/0.375/0.625$, 8 iterations on average; '10': $i/F1/F2 = 2/0.425/0.675$, 6 iterations on average; '00': fixed 1 iteration; '11': fixed 2 iterations

Table 1: Baseline configuration

Our baseline configuration follows past work [10, 20]. There are eight cores in our CMP system. Each core is single-issue, in-order and can be operated at 4GHz. Our trace-driven simulation methodology limits the simulated cores to be in-order. Each core in the baseline has a 32MB private write-back DRAM cache to alleviate pressure on MLC PCM main memory bandwidth. The DRAM cache has a default 256B line size. We also examine 64B and 128B line sizes in a sensitivity study. The detailed parameters can be found in Table 1. The results showed that our techniques can obtain significant improvement on a wide range of baseline configurations.

We consider a main memory with a single 4GB MLC PCM DIMM. The 4GB PCM main memory is divided into 8 banks. A bank spreads across 8 PCM chips. Therefore, 8 banks share 8 PCM chips. The programming current of one chip is supplied by local charge pump.

We use the same DIMM power token number PT_{DIMM} as past work [8]. Let E_{LCP} and E_{GCP} represent the power efficiency of LCP and GCP, respectively. The following formula computes the maximum power tokens PT_{LCP} that each chip has:

$$PT_{\text{LCP}} = \frac{PT_{\text{DIMM}} \times E_{\text{LCP}}}{8} \quad (4)$$

Assume the GCP borrows Borrowed_i tokens from each chip ($1 \leq i \leq 8$ and $0 \leq \text{Borrowed}_i \leq PT_{\text{LCP}}$). The following formula computes the power tokens that the GCP can provide:

$$PT_{\text{GCP}} = \sum_{i=1}^8 \frac{\text{Borrowed}_i}{E_{\text{LCP}}} \times E_{\text{GCP}} \quad (5)$$

Thus, clearly, we have:

$$PT_{\text{DIMM}} = \sum_{i=1}^8 \frac{PT_{\text{LCP}} - \text{Borrowed}_i}{E_{\text{LCP}}} + \frac{PT_{\text{GCP}}}{E_{\text{GCP}}} \quad (6)$$

5.2. Simulated workloads

We modeled a CMP that executes multi-programmed workloads. We chose a subset of programs from the SPEC2006, BioBench, MiBench and STREAM suites to construct workloads that exhibit different memory access characteristics. Table 2 lists the R/W-PKI (Read/Write accesses per thousand instructions) of each workload. We used SimPoint [27] to pickup representative phase. We simulate 1 billion instructions to obtain performance results.

Name	Description	RPKI	WPKI
ast_m	SPEC-CPU2006 (C), 8 C.astar	2.45	1.12
bwa_m	SPEC-CPU2006 (C), 8 C.bwaves	3.59	1.68
lbn_m	SPEC-CPU2006 (C), 8 C.lbn	3.63	1.82
les_m	SPEC-CPU2006 (C), 8 C.leslie3d	2.59	1.29
mcf_m	SPEC-CPU2006 (C), 8 C.mcf	4.74	2.29
xal_m	SPEC-CPU2006 (C), 8 C.xalancbmk	0.08	0.07
mum_m	BioBench (B), 8 B.mummer	10.8	4.16
tig_m	BioBench (B), 8 B.tigr	6.94	0.81
qso_m	MiBench (M), 8 M.qsort	0.51	0.47
cop_m	STREAM (S), 8 S.copy	0.57	0.42
mix_1	2S.add-2C.lbn-2C.xalan-2B.mummer	1.16	0.58
mix_2	2S.scale-2C.mcf-2C.xalan-2C.bwaves	0.94	0.61
mix_3	2S.triad-2B.tigr-2C.xalan-2C.leslie3d	0.96	0.58

Table 2: Simulated applications

For our results, we define *speedup* as:

$$\text{Speedup} = \frac{\text{CPI}_{\text{baseline}}}{\text{CPI}_{\text{tech}}} \quad (7)$$

where $\text{CPI}_{\text{baseline}}$ and CPI_{tech} are the CPIs of the baseline setting and the setting with scheme *tech*, respectively. This metric is also used by previous closely related research [10, 20].

Write burst: We adopted a write scheduling strategy from [8]. When write queue is 100% full, a write burst postponing all read requests is issued. And it is finished when the write queue is drained to be empty. The percentage of time in write burst of our baseline directly shapes the performance improvement achieved by our schemes. Figure 10 shows the percentage of write burst in the entire application simulation time for baseline. Since most of our simulated benchmarks are write intensity, the average percentage of time in write burst for our baseline is 52.2%, which is a strong motivation to improve heavily power constrained MLC PCM write throughput. Our result on write burst percentage is higher than that in [8] for several reasons: (i) MLC PCM has $\times 8$ long write latency than SLC PCM; (ii) compared to the baseline configuration in [8], the CPU frequency in our baseline is doubled; (iii) larger memory line size and chip level power restriction have more significant negative influence on write throughput than Flip-n-Write [4].

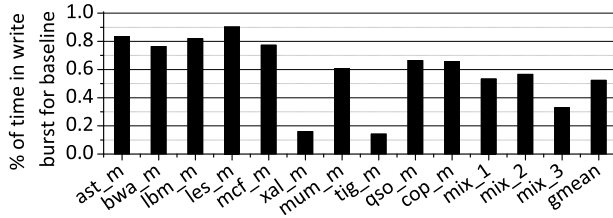


Figure 10: Percentage of execution cycles in write burst for baseline.

6. Experimental results

We implemented and compared the following schemes.

- **Ideal:** an ideal scheme that has an unlimited power budget.
- **DIMM-only:** a scheme that enforces only a DIMM power budget ($\text{PT}_{\text{DIMM}}=560$) [8]. The chip power is unrestricted.
- **DIMM+chip:** a scheme that enforces both DIMM and chip power budgets using Hay *et al.*'s technique [8]. To adopt this scheme, an oracle counter is introduced to provide the exact number of chip-level cell changes with no latency overhead. Here, $\text{PT}_{\text{LCP}} = \text{PT}_{\text{DIMM}} \times 0.95 / 8$.

- **GCP-CL-E:** a scheme that uses only FPB-GCP. The cell mapping, CL, may be NE (naïve mapping), VIM, or BIM. The GCP's power efficiency, *E*, ranges from 50% to 95% (0.5 to 0.95).
- **GCP+IPM:** a scheme that uses both FPB-GCP and FPB-IPM. By default, we use GCP-BIM-0.7 for the GCP. Multi-RESET (MR) is also evaluated.

Evaluation order and normalization. In Figure 4, we showed that the performance drop from Ideal to DIMM-only is due to iteration-oblivious budgeting, and the drop from DIMM-only to DIMM+chip is due to the chip power budget. In this section, we aim to restore these performance drops in reverse order. We first evaluate FPB-GCP with the goal to restore performance close to DIMM-only. Next, we add FPB-IPM with the goal to restore performance close to Ideal. In this section, the speedup values are normalized to DIMM+chip.

6.1. Effectiveness of FPB-GCP

6.1.1. Performance improvement Figure 11 shows IPB-GCP's effectiveness for different GCP power efficiency values. We used the naïve cell mapping (NE) in this experiment. We compared GCP with DIMM-only as IPB-GCP aims to eliminate the chip power budget.

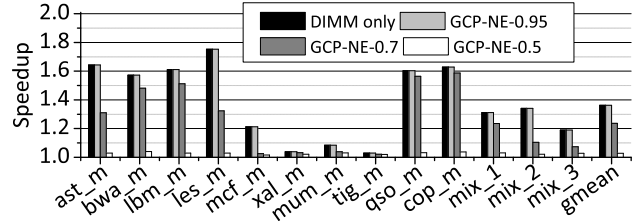


Figure 11: Speedup with different GCP power efficiencies.

From Figure 11, the GCP's power efficiency has a large performance impact. When $E_{\text{LCP}} = E_{\text{GCP}}$, GCP-NE-0.95 is 36.3% better than DIMM+chip. We found that GCP-NE-0.95 and DIMM-only have the same performance. The reason is, when $E_{\text{LCP}} = E_{\text{GCP}}$, there is no waste to let IPB-GCP borrow tokens from the LCPs (Equation 5).

In practice, the GCP is likely to be less efficient than the LCP. When $E_{\text{GCP}}=70\%$ (a typical value for an off-chip power supply), GCP-NE-0.7 improves performance by 23.7% over DIMM+chip. However, when the power efficiency is decreased further, its effectiveness diminishes. When $E_{\text{GCP}}=50\%$, the GCP cannot help at all: on average, only 2.8% improvement was observed.

In Figure 11, some programs are less sensitive to chip power budget. There are three scenarios. (i) When write operations are intensive, e.g., *mcf* or *mum*, the bottleneck shifts to the DIMM power budget. IPB-GCP often cannot borrow enough tokens to help. (ii) When a program has few writes, e.g., *xal*, the writes have little performance impact. (iii) When a program has many more reads than writes, e.g., *tig*, the performance bottleneck shifts from the writes to the reads such that the chip power budget has a small impact.

6.1.2. Cell mapping optimization Figure 12 compares different cell mapping. In these results, the GCP has practical power efficiency values. When $E_{\text{GCP}}=70\%$, VIM and BIM effectively mask the chip power budget; the performance loss versus DIMM-only is only 2% and 1.4%, respectively. VIM and BIM are comparably effective with BIM being slightly better. BIM better balances cell changes when a PCM line stores either FP or integer values.

More importantly, VIM and BIM make FPB-GCP effective for a 50% GCP power efficiency. These advanced cell mappings better

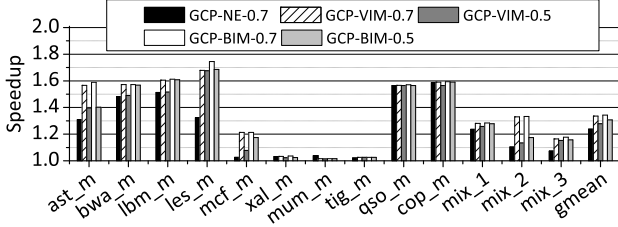


Figure 12: Speedup of cell mapping optimizations.

balance cell changes, which reduces how often the GCP needs to be employed. With fewer requests sent to the GCP, the advanced mappings relax the demands on the highly power-inefficient GCP.

6.1.3. GCP area overhead We next estimated the GCP’s area overhead. Since the area of the charge pump is proportional to the maximum power that it can provide, we collected the maximal power tokens requested for GCP under different cell mappings and compared their area overheads.

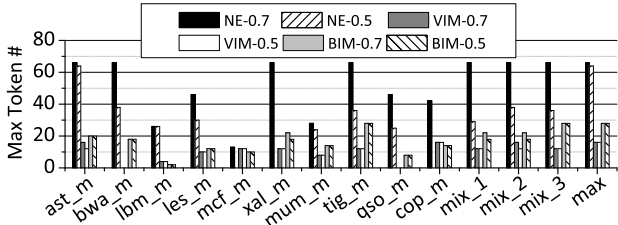


Figure 13: Maximum number of tokens requested by the GCP.

Figure 13 reports the maximum power tokens for each workload when E_{GCP} is 70% and 50%. The maximum requested power tokens are 66, 16, and 28 for the naïve mapping, VIM, and BIM, respectively. Interestingly, with VIM, there is no request to the GCP for the *bwa* benchmark, which indicates VIM balanced the cell writes very well across the PCM chips.

Using the maximum power tokens requested, Table 3 estimates the area overheads under different schemes. As discussed in Section 2.2, $2\times Local$ can also mask the chip power budget. However, this scheme doubles the LCP area in each chip, i.e., 100% overhead. Using the GCP greatly reduces area overhead. For example, with VIM and 70% GCP power efficiency, the GCP overhead is only 4.1% of $2\times Local$.

Scheme	Power Tokens	Overhead
Baseline (8 chips)	$70 * 8 = 560$	—
$2\times Local$ (8 chips)	$140 * 8 = 1120$	100%
GCP-NE-0.95	$66/0.95 = 70$	12.5%
GCP-NE-0.70	$64/0.70 = 92$	16.4%
GCP-VIM-0.95	$16/0.95 = 17$	3.1%
GCP-VIM-0.70	$16/0.70 = 23$	4.1%
GCP-BIM-0.95	$28/0.95 = 30$	5.4%
GCP-BIM-0.70	$28/0.7 = 40$	7.1%

Table 3: Charge pump overhead as measured by power tokens

We only compare the charge pump size. FPB-GCP also needs a dedicated pin for each PCM chip to inject the extra power from the GCP. A wire is needed on the DIMM to connect the GCP to each PCM chip as well.

6.1.4. GCP pin and packaging overhead To realize GCP, an extra pin is needed per PCM chip to deliver the high voltage and large cur-

rent produced by the GCP on the DIMM. This overhead can be justified by the large performance improvement and small GCP size. In addition, the pin overhead is localized to the DIMM, rather between the memory controller and the CPU/PCM chips. Several recent designs use a similar approach. Raghavan *et al.* [23] use additional pins to inject extra power for concurrent computing in embedded systems. A similar external current supply interface has been implemented in a recent PCM chip prototype [5].

The wire overhead between the GCP and PCM chips on the DIMM is negligible. Although write throughput on a DIMM is significantly increased with our techniques, thermal dissipation is also increased from more simultaneous writes, which requires better thermal control.

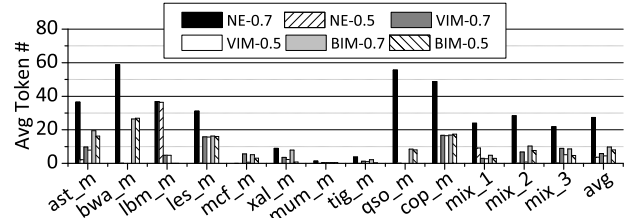


Figure 14: Average power tokens requested by NE, VIM and BIM.

6.1.5. Minimize wasted energy When different cell mappings have similar performance improvement, the mapping that needs fewer power tokens from the GCP is preferred as it reduces energy waste on the wire. Figure 14 reports the average number of tokens requested per line write from the GCP. VIM and BIM greatly reduce the total number of tokens requested. And thus, on average, VIM and BIM reduce energy waste by 78.5% and 64.4% over the naïve mapping at 70% GCP power efficiency.

6.1.6. BIM overall effectiveness The last experiment considers BIM effectiveness as the GCP’s power efficiency is decreased. Figure 15 reports speedup for three typical workloads. BIM helps preserve the performance benefit relative to DIMM+chip with very low GCP power efficiency. For example, in *mix_1*, BIM is still effective, although GCP power efficiency is as low as 20%.

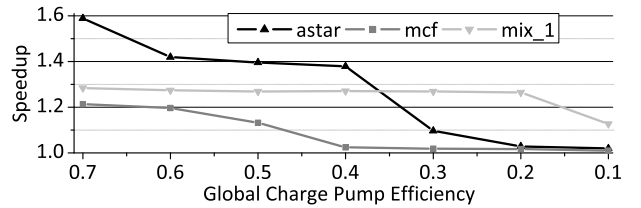


Figure 15: Speedup with BIM as GCP efficiency is decreased.

6.2. Effectiveness of FPB-IPM

6.2.1. Performance improvement We evaluated the effectiveness of FPB-IPM. The goal is, together with FPB-GCP, to make performance close to ideal. Figure 16 reports the speedup achieved by IPM and Multi-RESET over DIMM+chip. GCP is used with BIM at 70% GCP power efficiency. Figure 16 also shows the performance improvement for GCP power efficiency values of 50% (gm0.5) and 30% (gm0.3).

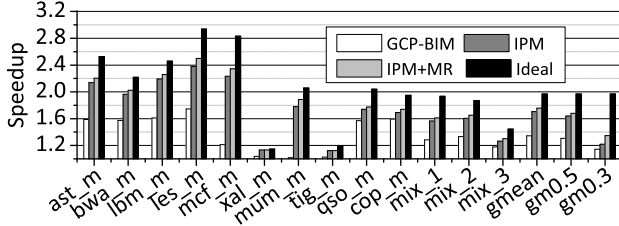


Figure 16: IPM and Multi-RESET Speedup.

On average, IPM improves performance by 26.9% over GCP-BIM. IPM+MR includes Multi-RESET that splits the first RESET iteration of a write into 3 new iterations, when there are not enough power tokens. IPM+MR has a 30.7% performance improvement over GCP-BIM and 75.6% improvement over DIMM+chip. This value is within 12.2% of Ideal, which has no power restrictions.

Also, from Figure 16, the overall performance improvement decreases with decreasing GCP power efficiency (compare gmean, gm0.5 and gm0.3). In addition, the improvement from IPM is stable from 70% GCP efficiency to 50% efficiency but drops at 30%. Multi-RESET tends to be more beneficial as efficiency decreases. For the benchmarks with a large number cell changes and a large WPKI, e.g., *mcf* and *mum*, IPM achieves significant improvements over GCP-BIM, indicating IPM makes better use of DIMM power for these benchmarks.

Intuitively, Multi-RESET increases the overlap among the long SET portions of multiple write requests. With a small available budget that cannot support all RESETs in one write, Multi-RESET adopts a greedy strategy to start a portion of RESETs in one write as early as possible. Without Multi-RESET, the many small pieces of available budget will be wasted for at least the current iteration. By fully utilizing these small available budget fragments, the power consuming RESETs in a write finish in multiple rounds. In this way, the processing time in every write burst can be reduced.

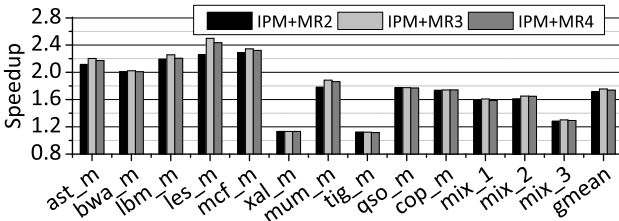


Figure 17: Multi-RESET iteration split limit.

6.2.2. Multi-RESET iteration count Multi-RESET introduces more RESET iterations. In turn, this lowers the maximum power demand but lengthens write latency. We examined how Multi-RESET should split the RESET iteration; i.e., how many new iterations should a single RESET be split into. Figure 17 reports performance when Mutli-RESET splits the first RESET iteration into 2, 3, or 4 new RESET iterations. As shown in Figure 17, the best improvement is achieved for 3 iterations. There is a 2% performance decrease at 4 iterations due to the longer write latency. Thus, we use 3 as the limit when applying Mutli-RESET.

6.3. Throughput improvement

As the performance improvement comes mainly from improved write throughput, we report overall throughput gains in Figure 18.

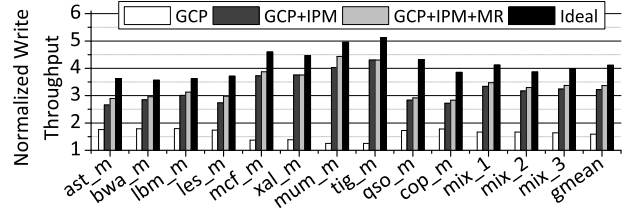


Figure 18: Write throughput improvement.

The results are normalized to DIMM+chip. From Figure 18, FPB achieves around 58.8% throughput improvement from GCP and 3.4× improvement when GCP, IPM and MR are applied. The write throughput obtained by GCP, IPM and MR is smaller than the Ideal (no power restrictions) write throughput by 22%. This throughput gap is due to the tight DIMM level power budget and the large memory line size.

6.4. Design space exploration

To evaluate the effectiveness of our proposed fine-grained power budgeting schemes under different settings, we did experiments in a wide design space with different memory line sizes, last-level cache capacities, number of entries in the write queue, and number of power tokens. We also integrated our FPB schemes with the state-of-the-art designs for MLC PCM: write cancellation, write pausing [20] and write truncation [10]. These methods are orthogonal to power budgeting. In the following design exploration, we use IPM+MR with BIM and $E_{GCP}=70\%$. We abbreviated this combined scheme as FPB.

In the comparison, when studying the sensitivity of parameter X, each bar is normalized to DIMM+chip that has the same X value. Different bars show different X values.

6.4.1. Cache/memory line size Figure 19 compares the performance impact with different memory line sizes. We assume that the MLC PCM memory line size is the same as the last-level cache’s line size. For 64B line size, Hay *et al.* observed that the existing DIMM power budget barely meets the demand for eight simultaneous line writes [8]. The improvement that FPB achieves is modest for 64B line size. For large line sizes (or large row buffer sizes), the number of line writes are reduced but each line write changes more cells, which creates contention for the power budget as writes are issued. From Figure 19, FPB achieves a larger improvement with bigger line sizes due to better utilization of the DIMM power budget. On average, FPB has a 41.3%, 61.8% and 75.6% improvement for 64B, 128B and 256B line sizes.

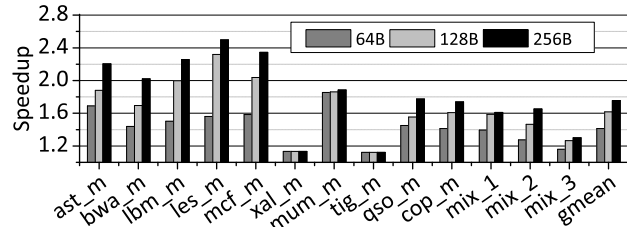


Figure 19: Speedup of FPB for different line sizes.

6.4.2. Last-level cache capacity Figure 20 compares performance for FPB under different last-level cache (LLC) capacities. With a small LLC, e.g., 8MB, there are a large number of memory accesses, which causes the system bottleneck to be main memory bandwidth.

Enforcing the DIMM and chip power budget with DIMM+chip results in even lower memory throughput and performance. On average, FPB achieves 39.9% improvement over DIMM+chip in this setting.

However, as LLC capacity is increased, the number of writes is reduced, yet each line write tends to have more cells to be changed. An improvement in the memory throughput exhibits large performance improvement. On average, FPB achieves 62.1% and 75.6% performance gains for 16MB and 32MB LLC capacities.

We also tried 128MB LLC size per core (1GB LLC for 8 cores). With a large LLC capacity, the offchip read and write traffic in the benchmarks is substantially reduced. Our power management schemes improve performance by 23.4% due to the short write burst time in this case. Most part of performance gain is achieved on streaming benchmarks, such as *qso* and *cop*.

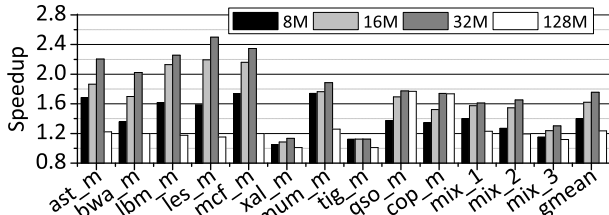


Figure 20: Speedup of FPB for different LLC capacities.

6.4.3. Number of write queue entries Figure 21 shows FPB’s effectiveness for varying number of write queue entries. The writes in the queue are flushed when the queue is full. With more entries in the queue, the bursty flush tends to request more power tokens, which is sensitive to write throughput. On average, FPB improves performance by 75.6%, 85.2% and 88.1% for three write queues.

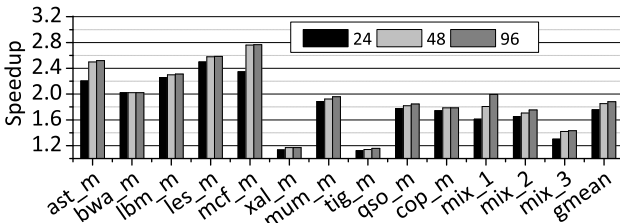


Figure 21: Speedup of FPB for different write queue sizes.

For benchmarks with large WPKI, such as *mum*, FPB has a large speedup. The overall performance improves significantly when the write entry count is increased from 24 to 48. It saturates at 48, and a 96-entry write queue does not exhibit notable improvement over a 48-entry queue.

6.4.4. Number of power tokens Figure 22 shows the performance impact of using 1/8 more or fewer power tokens. We chose this setting to study performance when the overall area change (increase or decrease) is about one LCP size, i.e., all eight chips each increase or decrease by 1/8 size.

From Figure 22, FPB does better with a tighter power budget. This phenomenon is due to FPB better using the power budget than DIMM+chip. If there is an abundant power budget, then wasting some tokens will not have a large performance impact and it is less critical to design advanced power budgeting schemes.

6.4.5. Integrating write pausing and write truncation Write cancellation, write pausing [20], and write truncation [10] are recent

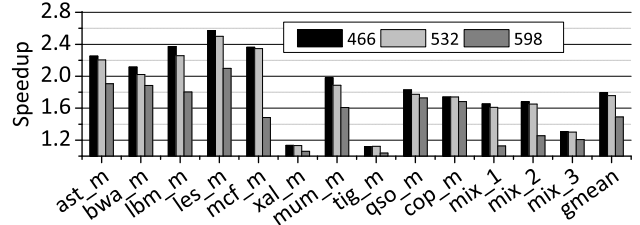


Figure 22: Speedup of FPM for different power token budgets.

effective read latency reduction schemes for MLC PCM. Although they address different issues than FPB, we examined their compatibility with FPB.

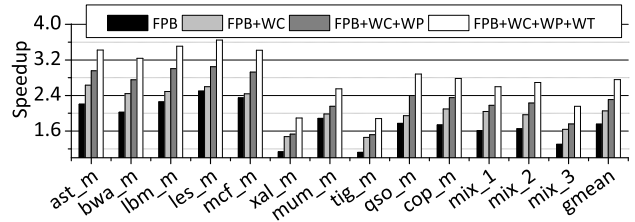


Figure 23: FPB with WC, WP and WT.

Figure 23 shows performance improvement when FPB is integrated with write cancellation (WC), write pausing (WP), and write truncation (WT). As WC needs a large write queue, we increased the entries in the read and write queues to 320 (40 R/W entries per bank, 8 banks). In our experiments, WC is always enabled with WP.

From Figure 23, we observe that FPB, WC, WT, and WT are orthogonal designs that target different performance opportunities. When all these designs are combined, on average, FPB+WP+WT achieves 175.8% improvement over DIMM+chip. This is a gain of 57% over FPB.

However, FPB+WP+WT has a smaller improvement over WP+WT. This happens because WC, WP and WT mitigate the importance of writes on performance, i.e., WC and WP move many writes off the critical path and WT reduces write latency. As discussed, FPB gains performance due to improved write throughput. Thus, when writes are less critical, the performance improvement from FPB is less.

7. Prior art

In addition to the related research discussed in the preceding sections, we discuss other related work in this section.

PCM power management. High write power is known as a major disadvantage of PCM. Schemes have been proposed to change only the cells that need to be changed [12, 31]. Cho *et al.* proposed *Flip-n-Write* that can pack two line writes with the power budget of writing the number of one line cells [4]. It has limited benefit for MLC PCM due to the additional states used in MLC PCM. Hay *et al.* proposed to track/estimate bit changes in the last-level cache and issue write operations as long as the DIMM power budget can be satisfied [8]. These power management schemes focus mainly on SLC PCM. The FPB schemes proposed in this paper address MLC PCM write by exploiting its characteristics.

To address write power in MLC PCM, Joshi *et al.* proposed a novel programming method that decreases write energy and latency by switching between two write algorithms: *single RESET multi-SET* and *single SET multi-RESET* [11]. The latter is often less re-

liable. Wang *et al.* proposed to reduce write energy by adopting different mappings between data values and resistance levels [28].

MLC PCM. MLC PCM can effectively reduce per bit fabrication cost. Schemes have been proposed to address its latency, write energy, and endurance issues [20, 21, 10]. MLC PCM differs from SLC PCM in that it has a non-negligible resistance drift problem. Zhang *et al.* proposed different encoding schemes to mitigate drift [30]. Awasthi *et al.* proposed lightweight scrubbing operations to prevent soft errors [1].

Asymmetric write. The RESET and SET operations have asymmetric characteristics in terms of latency and power [24]. For SLC PCM, Qureshi *et al.* proposed to perform SET operations before the memory line is evicted from last-level cache [22]. When a write operation comes, only the short-latency RESET needs to be performed. Applying PRESET on MLC PCM indicates the adoption of *single SET multi-RESET* write scheme, which tends to increase the demand for power tokens.

Industry chip demonstration. Samsung recently demonstrated a 20nm PCM chip [5] that splits the conventional local charge pump into sub-pumps to reduce voltage drop and current consumption along long wires. The prototype adds an external high voltage and current supplement interface. Our scheme is orthogonal to this design as IPM can be applied at the sub-pump level, while GCP can reuse the current supplement interface.

8. Conclusions and Future Work

In this paper, we proposed FPB, fine-grained power budgeting, that applies two new power management strategies: FPB-IPM enables iteration power management to reclaim unused power tokens as early as possible at the DIMM level and FPB-GCP uses a global charge pump to mitigate power restrictions at the chip level. Our experimental results showed that FPB is effective and robust for a broad range of MLC PCM settings. On average, FPB improves performance by 76% and write throughput by $3.4\times$ over previous power management technique [8].

In this paper, we used a range of power efficiencies to accommodate different device types and their associated parameters. For a more detailed evaluation and low level understanding of the circuit behavior, our future work will consider the circuit level design and conduct SPICE simulations.

9. Acknowledgments

We thank the anonymous reviewers for their constructive suggestions, and Prof. Moinuddin K. Qureshi for shepharding the paper. We also acknowledge the support from PCM@Pitt research group. This research is supported partially by National Science Foundation grants CNS CAREER-0747242 and CNS-1012070.

References

- [1] M. Awasthi, *et al.*, "Efficient Scrub Mechanisms for Error-Prone Emerging Memories," in *HPCA*, 2012.
- [2] F. Bedeschi, *et al.*, "A Bipolar-Selected Phase Change Memory Featuring Multi-Level Cell Storage," *JSSC*, 2009.
- [3] M. Boniardi, *et al.*, "Impact of Material Composition on the Write Performance of Phase-Change Memory Device," in *IMW*, 2010.
- [4] S. Cho and H. Lee, "Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance," in *MICRO*, 2009.
- [5] Y. Choi, *et al.*, "A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth," in *ISSCC*, 2012.
- [6] J. Dickson, "On-chip High Voltage Generation in NMOS Integrated Circuits using an Improved Voltage Multiplier Technique," *JSSC*, 1976.
- [7] K. Fang, *et al.*, "Memory Architecture for Integrating Emerging Memory Technologies," in *PACT*, 2011.
- [8] A. Hay, *et al.*, "Preventing PCM Banks from Seizing Too Much Power," in *MICRO*, 2011.
- [9] Y. Hwang, *et al.*, "MLC PRAM with SLC write-speed and robust read scheme," in *VLSIT*, 2010.
- [10] L. Jiang, *et al.*, "Improving Write Operations in MLC Phase Change Memory," in *HPCA*, 2012.
- [11] M. Joshi, *et al.*, "Mercury: A Fast and Energy-Efficient Multi-level Cell based Phase Change Memory System," in *HPCA*, 2011.
- [12] B. C. Lee, *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
- [13] K. T. Malladi, *et al.*, "Towards Energy-Proportional Datacenter Memory with Mobile DRAM," in *ISCA*, 2012.
- [14] D. Mantegazza, *et al.*, "Statistical Analysis and Modeling of Programming and Retention in PCM Arrays," in *IEDM*, 2007.
- [15] T. Nirschl, *et al.*, "Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory," in *IEDM*, 2007.
- [16] H. Oh, *et al.*, "Enhanced Write Performance of a 64-Mb Phase-Change Random Access Memory," *JSSC*, 2006.
- [17] G. Palumbo and D. Pappalardo, "Charge Pump Circuits: An Overview on Design Strategies and Topologies," *IEEE Circuits and Devices Magazine*, 2010.
- [18] M. K. Qureshi, *et al.*, "Enhancing Lifetime and Security of PCM-based Main Memory with Start-Gap Wear Leveling," in *MICRO*, 2009.
- [19] M. K. Qureshi, *et al.*, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *ISCA*, 2009.
- [20] M. K. Qureshi, *et al.*, "Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing," in *HPCA*, 2010.
- [21] M. K. Qureshi, *et al.*, "Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase Change Memories," in *ISCA*, 2010.
- [22] M. K. Qureshi, *et al.*, "PreSET: Improving Read Write Performance of Phase Change Memories by Exploiting Asymmetry in Write Times," in *ISCA*, 2012.
- [23] A. Raghavan, *et al.*, "Computational Sprinting," in *HPCA*, 2012.
- [24] S. Raoux, *et al.*, "Phase-change Random Access Memory: A Scalable Technology," *IBM J. RES. & DEV.*, 2008.
- [25] S. Schechter, *et al.*, "Use ECP, not ECC, for Hard Failures in Resistive Memories," in *ISCA*, 2010.
- [26] N. H. Seong, *et al.*, "Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-change Memory with Dynamically Randomized Address Mapping," in *ISCA*, 2010.
- [27] T. Sherwood, *et al.*, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, 2002.
- [28] J. Wang, *et al.*, "Energy-efficient Multi-Level Cell Phase-Change Memory System with Data Encoding," in *ICCD*, 2011.
- [29] D. H. Yoon, *et al.*, "BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs," in *ISCA*, 2012.
- [30] W. Zhang and T. Li, "Helmet: A Resistance Drift Resilient Architecture for Multi-level Cell Phase Change Memory System," in *DSN*, 2011.
- [31] P. Zhou, *et al.*, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *ISCA*, 2009.