Reducing Writes in Phase-Change Memory Environments by Using Efficient Cache Replacement Policies

Roberto Rodríguez-Rodríguez, Fernando Castro, Daniel Chaver, Luis Pinuel and Francisco Tirado
ArTeCS Group
Complutense University of Madrid
Email: {robalrr, fcastror, dani02, lpinuel, ptirado}@pdi.ucm.es

***Abstract*—Phase Change Memory (PCM) is currently postulated as the best alternative for replacing Dynamic Random Access Memory (DRAM) as the technology used for implementing main memories, thanks to its significant advantages such as good scalability and low leakage. However, PCM also presents some drawbacks compared to DRAM, like its lower endurance. This work presents a behavior analysis of conventional cache replacement policies in terms of the amount of writes to main memory. Besides, new last level cache (LLC) replacement algorithms are exposed, aimed at reducing the number of writes to PCM and hence increasing its lifetime, without significantly degrading system performance.**

I. Introduction

Although DRAM has been the prevalent building block for main memories during many years, current research is focused on exploring other technologies for designing future memory systems in response to the scaling constraints observed when DRAM is used with small feature sizes. Among these technologies, PCM is one of the prime contenders.

This kind of low-cost and non-volatile memory avoids the need of refreshing the content of the cell, reducing also the static power consumption. Furthermore, it provides higher density than DRAM and therefore much higher capacity for the memory system within the same budget. Nevertheless, several obstacles restrict the adoption of PCM as main memory: long write access latency, high write power and limited endurance. Endurance is related with the amount of writes that a cell is likely to sustain, and in PCM cells this number is significantly reduced compared to DRAM, so the write traffic must be cut in order to extend the lifetime of PCM-based systems. For this purpose, several architectural techniques have been proposed recently [1]–[4].

In this work, we deal with this problem by focusing on the LLC replacement policy. The goal is to efficiently design a policy that retains in the LLC the most frequently written blocks and therefore to reduce the amount of writebacks to main memory. Thus, in this context, the LLC replacement policy is not just oriented to provide a low miss rate and to increase system performance, but also to reduce the amount of writes to main memory. We analyze the behavior of classical and current cache replacement policies [5]–[7] regarding the amount of writes to main memory, and we propose new policies in order to reduce this write traffic without significantly impacting performance.

978-3-9815370-0-0/DATE13/

The rest of the paper is organized as follows: Section 2 describes the work related to our research. Section 3 presents the algorithms proposed to increase the lifetime of PCM-based memory systems. Sections 4 and 5 detail the experimental framework used and the obtained results respectively. Finally, Section 6 concludes.

II. Related work

A. Techniques for reducing writes to PCM

- Eliminating redundant bit writes [1], [8]: As reads are much faster and less power consuming than writes in PCM, every write is preceded by a read and a bitwise comparison, writing only those bits that change.
- *Flip-N-Write* [9]: Before performing a write to PCM, both the data to write and its bitwise inverse are compared with the data stored in the row, writing the one that involves less bit flips.
- *Wear Leveling* [8]: These techniques try to even out the number of writes performed to each PCM cell.
- Hybrid memory [4]: The idea here is to avoid writes to PCM by inserting an extra memory level that filters most of such accesses, and which is built on a memory technology not so sensible to writes.
- Write-back impact reduction [10], [11]: The amount of writebacks to PCM is reduced extending [12] by including this goal in the proposed LLC partitioning algorithm and changing the LRU policy in order to consider also dirtiness information [10]. The write-backs impact is also reduced by evenly distributing them among write queues.

B. Cache replacement policies used in this work

- LRU (Least Recently Used): It discards the least recently used block, under the philosophy that, due to temporal locality, it is also the block that will not be required for the longest time in the future.
- PeLIFO (Pseudo Last In First Out) [7]: It builds on a LIFO (Last In First Out) replacement policy [5], in which, making use of a Fill Stack, the block that entered the cache in the last place is the candidate for

replacement. In PeLIFO the bottom part of the Fill Stack is reserved for long-term reuses as LIFO does. However, unlike LIFO, which performs all the replacements at the head of the stack, PeLIFO dynamically selects for replacement intermediate positions (called Escape Points) that guarantee that short-term reuses are also fulfilled.

- SRRIP, BRRIP and DRRIP (Static, Bimodal and Dynamic Re-Reference Interval Prediction respectively) [6]: The recency stack is thought of as a Re-Reference Interval Prediction (RRIP) Stack that represents the order in which blocks are predicted to be re-referenced. The block at the head is predicted as *near-immediate* (i.e. the block will be re-referenced sometime soon) while the block at the tail is predicted as *distant* (i.e. the block will be re-referenced in the distant future). SRRIP uses M bits per cache block to store one of 2^M possible RRPVs (Re-Reference Prediction Values). On cache fills SRRIP predicts, following its insertion policy, that the missing block will have an intermediate state denoted as *long* re-reference (RRPV=2^M-2). On re-reference, SRRIP may employ two different promotion policies, HP (Hit Priority) and FP (Frequency Priority): SRRIP-HP updates the RRPV of the associated block to zero while SRRIP-FP just decrements it. On a cache miss, SRRIP randomly selects the block (victimization policy) with a *distant* RRIP (RRPV=2^M-1). If it does not exist, SRRIP increments the RRPVs of all blocks in the set and repeats the search.
  SRRIP can cause thrashing under some circumstances. To avoid it, the authors propose BRRIP, that inserts most cache blocks with a *distant* RRIP and a few of them with a *long* RRIP. In order to be robust across all cache access patterns, a third policy (DRRIP) proposes to use Set Dueling [13] to identify which insertion policy is best suited for the application, choosing between SRRIP and BRRIP as Fig. 1 illustrates.

- CLean-Preferred victim selection policy CLP [10]: It implements a modified LRU policy that gives preference to clean blocks (vs dirty ones) when choosing a victim. Among the different configuration options it offers, in this paper we evaluate the one reporting the highest write reduction.

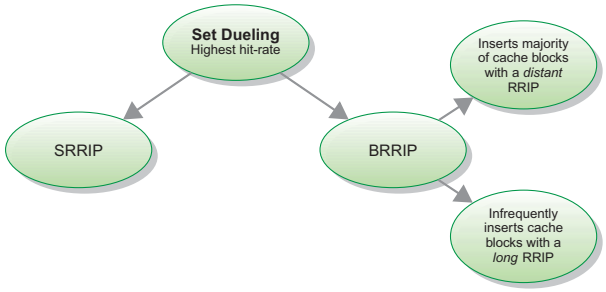


Fig. 1. Insertion of a new block under DRRIP.

III. Proposed Policies

The replacement policy determines how to manage insertion, promotion, and victimization of blocks in cache. Usually, decisions are only performed with the goal of increasing hit rate. However, when the LLC is backed up with a PCM, decreasing the amount of LLC writebacks may become even more important than reducing the number of misses. Hence, we propose several modifications to the insertion, promotion and victimization policies that incorporate this objective. Information such as the block dirtiness state or the kind of access performed (read or write), are used by our policies for predicting whether the block will generate future writebacks.

In some cases a hybrid mechanism is employed, that combines two or more replacement policies, and it selects in each replacement which one should be used. The most common mechanism employed for making this decision is the Set Dueling technique proposed in [13]. Usually, the decision is guided only by hit rate information. However, in the case of a PCM-based system, the Set Dueling mechanism should also consider information concerning the amount of writebacks that the LLC will generate.

Next, we describe the replacement policies proposed in order to reduce the amount of PCM writes. Although we have tried many different options, we only explain and evaluate those that report interesting results from any point of view. To denote them, we add a *W* to the acronym of policy they are based on:

- *DRRIPW4*: This policy modifies DRRIP as follows: the Set Dueling mechanism selects the replacement policy that reports the lowest number of writebacks to main memory, instead of the one that achieves the highest hit rate.
- *DRRIPW7*: This policy operates like DRRIPW4, but instead of using a static promotion policy it uses a dynamic one: when the Set Dueling mechanism selects the SRRIP component, HP is employed, whereas when BRRIP is selected, FP is used.
- *DRRIPW8*: Like the previous policy, it operates as DRRIPW4 and changes the promotion policy to a dynamic one: clean blocks are promoted with FP while dirty blocks use HP. This way, dirty blocks (which generate writebacks), are given more opportunities to stay in the cache than clean blocks.
- *DRRIPW9*: This algorithm combines the two previous policies. It operates like DRRIPW4, and changes the promotion policy as follows: when the Set Dueling mechanism selects the SRRIP component, HP is employed for dirty blocks and FP for clean blocks, whereas when BRRIP is selected, FP is always used.
- *DRRIPW10*: This algorithm modifies the victimization policy of DRRIPW8: Among all blocks with a *distant* RRIP, it selects a clean one; if not found, a dirty block is victimized. Like in the original DRRIP policy, if no blocks with a *distant* RRIP exist, it increments RRPV of all blocks and repeats the process.
- *DRRIPW11*: In this case, the victimization policy of original DRRIP is modified: When a replacement is

required, this algorithm looks for a candidate satisfying two criteria: 1) RRPV with the maximum value, 2) clean block. If such a block is not found, the RRPV of all blocks in the cache set is augmented by one and a new search starts. If the RRPV of all blocks are set to the maximum values and no clean block exists, the algorithm evicts the first block in the array.

- *DRRIPW12*: This algorithm is very similar to the previous one, but with an important difference: during the first stage, like before, the clean block with the highest RRPV is victimized, but in this case without changing the RRPV state of the blocks. During the second stage, a common search is performed.

IV. Experimental Framework

As simulation infrastructure we use Pin [14] for obtaining the memory trace and MultiCacheSim [15] to model the cache hierarchy. The cache hierarchy is based on an Intel Atom, which provides two cache levels, being the LLC a non-inclusive (also non-exclusive) cache. The size of the first level (L1) is 24KB, with 4-ways and 64-byte line size, whereas the size of the second level (L2, and in this case also LLC) is 512KB, with 8-ways and 64-byte line size. Since we target a processor in the frontier between embedded and general purpose, we use both the MiBench [16] and the SPEC CPU2006 [17] suites with *large* and *train* entries respectively. All applications are executed until completion.

Regarding the energy model employed, we use CACTI 6.5 [18] to determine the energy consumption per access in each cache level, whereas for computing the energy consumption associated with accesses to main memory we follow [11], employing 1J/GB and 6J/GB per PCM read and PCM write respectively. In Table I we show data regarding latencies and energy consumption per memory level.

TABLE I. Latencies and Energy Consumption

Level	Latencies (cycles)	Energy (Read/Write/Tag) (nJoules)
L1	1	0.092/0.066/0.001
L2	15	1.102/1.166/0.010
PCM read	200	59.604
PCM write	4000	357.627

Finally, it is worth to note that we employ the Average Memory Access Time (AMAT) in order to estimate the performance of each evaluated algorithm. Although AMAT just considers the latencies of each level and not the overlapping between memory accesses, it is commonly accepted as a valid metric to estimate system performance. Next we show the equations employed to determine the memory hierarchy energy consumption (1) as well as the AMAT (2):

$$\begin{aligned} Energy = \sum_{i=1}^{2} (RHL_i * REL_i + WHL_i * WEL_i + \\ +(RML_i + WML_i) * (TEL_i + WEL_i)) + \\ +RPCM * REPCM + WPCM * WEPCM \end{aligned} \quad (1)$$

where RHL_i and WHL_i denote read and write hits in cache level *i*, RML_i and WML_i denote read and write misses in cache level *i*, RPCM and WPCM correspond to the amount of reads and writes to PCM, REPCM and WEPCM denote the energy consumption per read and write to PCM and finally REL_i, WEL_i and TEL_i correspond to the energy consumption of a read, a write and a tag array consult in cache level *i*.

$$AMAT = \frac{HL_1 * LL_1 + RHL_2 * LL_2 + RPCM * LRPCM}{AccL_1} \quad (2)$$

where HL_1 and AccL_1 denote the total number of hits and accesses to L1, LL_i refers to the latency of *i* cache level and LRPCM denotes the latency in the access to PCM.

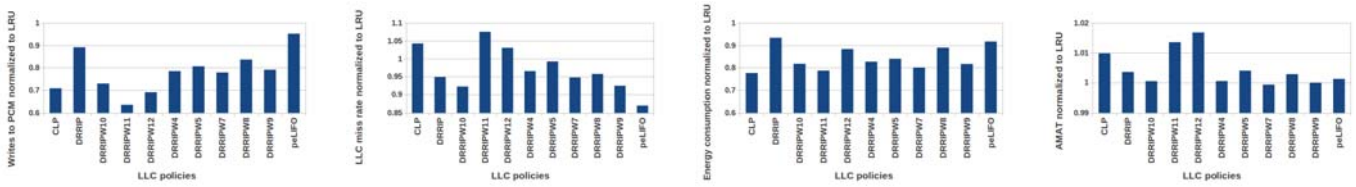
V. Evaluation

In this section we present a detailed experimental study of the different LLC replacement policies described. We report results about the number of writebacks, the LLC miss rate, the AMAT and the involved energy consumption in the memory hierarchy. Note that in all the figures and for each evaluated policy, we show the arithmetic mean of the normalized metrics respect to LRU, considering all the selected applications of each suite (MiBench and SPEC 2006). It is worth noting that L1 always employs the LRU algorithm, whereas the policies under evaluation are used in the L2. Note also that results shown for original DRRIP and for our proposed DRRIP-based algorithms that do not specifically change the promotion policy, are derived from using an *HP promotion policy*.

Fig. 2(a) and 3(a) illustrate the average number of writes from LLC to PCM that generate the different policies. Focusing on MiBench suite, we can state that the best results are achieved with DRRIP-based policies and CLP, being DRRIPW11 the one that gets the highest reduction with around 37%. In the SPEC 2006 scenario, DRRIPW12 exhibts the highest reduction (around 13%), followed by CLP (almost 12% reduction), and DDRIPW10 that achieves a 10% reduction. Note that the percentage of avoided writes is higher in the MiBench suite due to the lower amount of writes that take place in this scenario (between 2 and 3 orders of magnitude compared to SPEC 2006 scenario).

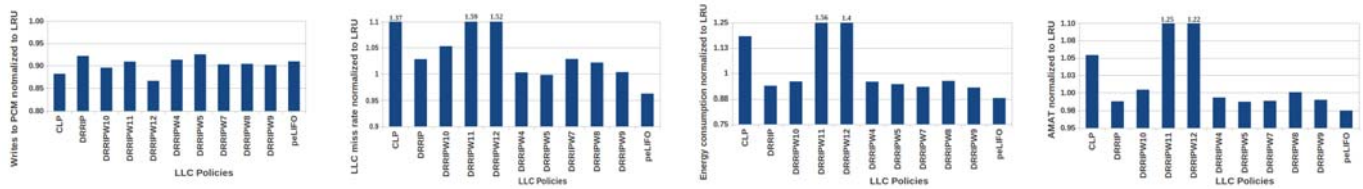
Being writeback reduction to PCM the main objective of this work, we should not neglect performance. Fig. 2(b) and 3(b) and Fig. 2(d) and 3(d) illustrate the LLC miss rate and the AMAT respectively for the different policies evaluated. In some policies a high write reduction comes at the expense of a significant miss rate increment and consequently a performance drop. Although in the MiBench scenario this AMAT degradation may be tolerable, like occurs with DRRIPW11 and CLP algorithms, in the SPEC 2006 scenario this performance drop turns unacceptable, invalidating this kind of proposals (DRRIPW11, DRRIPW12 and CLP). Nevertheless, DRRIPW10 (or other policies like DRRIPW4 and DRRIPW7), achieves a significant write reduction (28% and 11% for MiBench and SPEC 2006 respectively in DRRIPW10), maintaining a reduced miss rate and therefore without significantly impacting performance.

Finally, Fig. 2(c) and 3(c) illustrate energy consumption on the memory system for the different evaluated policies. In the case of MiBench, all policies reduce the energy consumption of an LRU, being CLP and DRRIPW11 the less consuming ones (around 20% energy reduction). Instead, in the case of SPEC 2006, these two policies and DRRIPW12 exceed



(a) (b) (c) (d)

Fig. 2. Results normalized to LRU for the MiBench suite: (a) Amount of writes, (b) LLC Miss Rate, (c) Memory energy consumption, (d) AMAT.



(a) (b) (c) (d)

Fig. 3. Results normalized to LRU for the SPEC CPU2006 suite: (a) Amount of writes, (b) LLC Miss Rate, (c) Memory energy consumption, (d) AMAT.

the consumption of the others due to the high miss rate they exhibit. For this suite, most remaining policies oscillate between 5 and 10% reduction.

In summary we can conclude that the optimal policy depends on the particular requirements of the system and the user. For our configuration, DRRIPW10 (and other DRRIP-based policies) constitutes a good trade-off, showing a considerable PCM endurance improvement without degrading performance in both scenarios. Besides, in the case of SPEC 2006, most DRRIP based and peLIFO policies achieve a good trade-off between impact on PCM write traffic and performance.

VI. Conclusions

In this paper we deal with the endurance constraint of phase-change memories –one of the most promising memory technologies to replace conventional DRAM– by means of the last level cache replacemenet policy. For this purpose of extending PCM lifetime, we evaluate the operation of classical replacement algorithms in terms of writes to main memory and we propose new policies with the main goal of minimizing this number without degrading performance significantly.

The obtained results show that most of our policies manage to significantly reduce the amount of writes to PCM. In the evaluated configurations, DRRIPW10 postulates as the best alternative for obtaining a good trade-off between write traffic reduction (28% and 11% for MiBench and SPEC 2006 respectively) and performance degradation (0.1% and 0.4% for MiBench and SPEC 2006 respectively) respect to LRU. Furthermore, the energy reduction achieved reaches 18% and 4% for MiBench and SPEC 2006 respectively.

Acknowledgment

This work has been supported in part by the Spanish government through the research contract CICYT-TIN 2008/508, TIN2012-32180, Consolider Ingenio-2010 CSD2007-0050, and the HIPEAC-3 European Network of Excellence.

References

[1] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, p. 14, Jun. 2009.

[2] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," *Proceedings of the Micro-42*, pp. 2–13, 2009.

[3] M. Qureshi and M. Franceschini, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," *Computer Architecture*, 2010.

[4] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using PCM technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, Jun. 2009.

[5] C. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.

[6] A. Jaleel, K. Theobald, S. Steely Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer*, 2010.

[7] M. Chaudhuri, "Pseudo-lifo: the foundation of a new family of replacement policies for last-level caches," in *MICRO*, 2009, pp. 401–412.

[8] B. C. Lee *et al.*, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, p. 143, 2010.

[9] S. Cho, "Flip-n-write: a simple deterministic technique to improve PRAM write performance, energy and endurance," *Proceedings of MICRO-42*, pp. 347–357, 2009.

[10] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing PCM main memory lifetime," *Proceedings of DATE*, pp. 914–919, 2010.

[11] M. Zhou, Y. Du, B. Childers, R. Melhem, and D. Mossé, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM TACO*, vol. 8, no. 4, pp. 1–21, Jan. 2012.

[12] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, 2006, pp. 423–432.

[13] M. Qureshi, A. Jaleel, Y. Patt, and S. Steely, "Adaptive insertion policies for high performance caching," *ACM SIGARCH*, 2007.

[14] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, no. 6, pp. 190–200, Jun. 2005.

[15] B. Lucia, "https://github.com/blucia0a/ MultiCacheSim."

[16] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of WWC*, 2001, pp. 3–14.

[17] "http://www.spec.org/cpu2006/."

[18] "http://www.hpl.hp.com/research/cacti/."