Energy- and Endurance-Aware Design of Phase Change Memory Caches

Yongsoo Joo *, Dimin Niu *, Xiangyu Dong *, Guangyu Sun *, Naehyuck Chang †, and Yuan Xie *
* Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802
† Department of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea
Email: *{ysjoo, dun118, xydong, gsun, yuanxie}@cse.psu.edu, †naehyuck@elpl.snu.ac.kr

***Abstract*—Phase change memory (PCM) is one of the most promising technology among emerging non-volatile random access memory technologies. Implementing a cache memory using PCM provides many benefits such as high density, non-volatility, low leakage power, and high immunity to soft error. However, its disadvantages such as high write latency, high write energy, and limited write endurance prevent it from being used as a drop-in replacement of an SRAM cache. In this paper, we study a set of techniques to design an energy- and endurance-aware PCM cache. We also modeled the timing, energy, endurance, and area of PCM caches and integrated them into a PCM cache simulator to evaluate the techniques. Experiments show that our PCM cache design can achieve 8% of energy saving and 3.8 years of lifetime compared with a baseline PCM cache having less than a hour of lifetime.**

I. Introduction

Various non-volatile memory devices have emerged over the last few decades, such as a phase change memory (PCM), a magneto-resistive random access memory (MRAM), and a ferro-electric random access memory (FeRAM). Among them, PCM is known to be one of the most promising technologies.

PCM has several attractive features such as in-place update, byte accessibility, high density, and higher write endurance compared with NOR and NAND flash memories. Particularly, the current PCM technology is already superior to NOR flash in all aspects. For example, Samsung Electronics has already announced a 512 Mb PCM chip [1], which can be directly used as a drop-in replacement of conventional NOR flash memories. Some researchers [2][3] also exploit the in-place update capability of PCM to devise hybrid PCM and NAND flash storages, where a small size of PCM is used as a metadata storage. As PCM consumes no leakage current and needs no refresh operation like DRAM, PCM is now even considered to be used as a main memory. Several previous work [4][5][6][7] confirm that a PCM main memory can achieve significant energy saving with comparable performance to that of a DRAM main memory.

The natural next step would be using PCM to build a cache memory. By replacing a conventional SRAM cache with a PCM cache, we can achieve many benefits such as high density, non-volatility, no leakage power, and high immunity to soft error. However, PCM has the following problems to be tackled:

• Write latency: PCM has much slower write speed than SRAM, causing performance degradation. However, we can put a larger PCM cache to the same area occupied by a SRAM cache thanks to its high density, which can help mitigate the performance degradation. Also, well-designed write buffers may hide the slow write speed of PCM.

• Write energy: A PCM write operation consumes up to ten times more energy than that of a PCM read operation. We can reduce the number of PCM write operations by increasing the cache size as it can reduce cache line fetches due to capacity misses. However, it cannot reduce the write traffic from the upper cache layer, which can dominate cache energy consumption if the cache size is large enough to hold the working set of running applications.

• Write endurance: PCM has more than 1000 times higher write endurance than NOR and NAND flash memories. However, it is still not enough to endure the write traffic to a cache memory. In order to prolong the lifetime of a PCM main memory, researchers suggested a set of wear-leveling methods [4], a new row buffer design [5], and a hybrid DRAM and PCM memory architecture [6], achieving from 5.6 to 22 years of lifetime. However, as the write traffic to a cache memory is much heavier than that to a main memory, we need more aggressive endurance-enhancing techniques for PCM caches.

In order to overcome such disadvantages of PCM cache, some previous work introduces a hybrid cache memory combining two or more memory types such as SRAM, PCM, and embedded DRAM [8][9]. To the best of our knowledge, however, there have been no research on how to implement a pure PCM cache as a drop-in replacement of a SRAM cache. Especially, the write endurance limitation of a PCM cache has not been analyzed in detail so far.

We study a set of design techniques to reduce the write energy consumption of a pure PCM cache and to prolong its lifetime. We found from simulation that a pure PCM L2 cache is feasible with our proposed techniques; compared with the baseline 4 MB PCM L2 cache having less than a hour of lifetime, we achieved 8% of energy saving and 3.8 years of lifetime.

The contribution of this work is summarized as follows:

• We modeled the timing, energy, and endurance of PCM caches.

• We developed a trace-driven PCM cache simulator that reports not only performance and energy but also the accumulated number of writes of each PCM cell.

• We studied a read-before-write, a data inverting, and a set of wear leveling schemes to reduce the write energy of a PCM cache and to enhance its write endurance.

• We presented guidelines to design energy- and endurance-aware PCM caches.

II. Background

The phase-change material used for PCM is called GeSbTe (GST), which is alloys of germanium, antimony, and tellurium. GST has two phases, known as an amorphous phase and a crystalline phase, representing the high and low electrical resistivity, respectively. The schematic view of a PCM cell with a MOSFET access device is shown in Figure 1. Every PCM cell contains a storage region and a selector transistor. The active storage region is implemented at the intersection of a vertical thin-film metallic layer and a thin layer of GST. The GST can be crystallized in about 150 ns, which ensures the high speed write operation of PCMs compared with flash

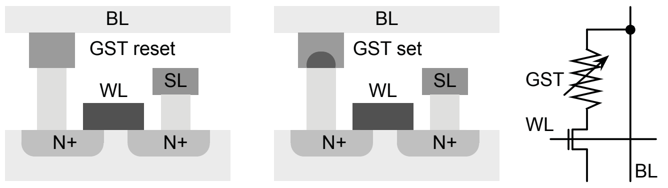

Fig. 1. PCM cell structure (BL: bit line, WL: word line, SL: source line).

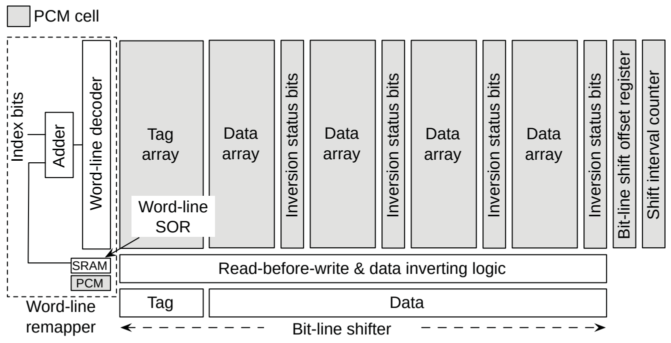

Fig. 2. Block diagram of the suggested PCM cache design.

memories. The phase-change material is crystallized by heating it above its crystallization temperature (a SET operation), and is melt-quenched to make the material amorphous (a RESET operation). These operations are controlled by electrical current: (1) a RESET operation uses a high amplitude pulse to place a PCM cell into a high-resistance state; (2) a SET operation uses a moderate amplitude but longer duration pulse, returning a PCM cell to a low-resistance state; and (3) a read operation uses a very low amplitude pulse and senses the cell resistance. PCM has many attractive features such as fast read access, high density, multi-bit storage capability, zero standby leakage, and non-volatility [10]. However, high write energy consumption and finite write endurance are the major challenges for PCM cache implementation.

III. PCM Cache Design Techniques

Figure 2 shows our proposed PCM cache architecture, where a read-before-write logic, a data inverting logic, a bit-line shifter, and a word-line remapper are integrated with the PCM tag and data array. The following subsections describe each component in detail.

A. Reducing the Amount of Writes

1) Read-Before-Write: The simplest implementation of a PCM write logic is to apply either a SET pulse or a RESET pulse according to the new data value to be written. However, such a write logic obviously wastes energy and the number of writes allowed for a PCM cell because it mostly performs redundant writes, i.e., the new value written to a PCM cell is the same to its previous value. Figure 3 shows that the amount of redundant writes accounts for 85% of the entire writes on average, similar to the result for a PCM main memory [4].

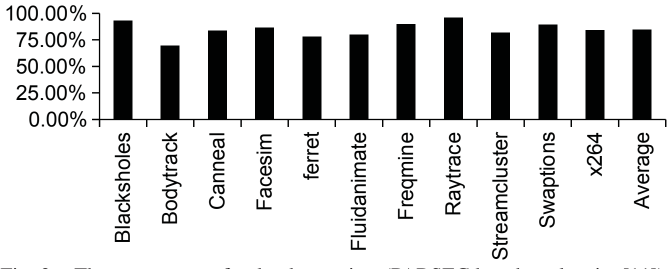

Fig. 3. The percentage of redundant writes (PARSEC benchmark suite [11]).

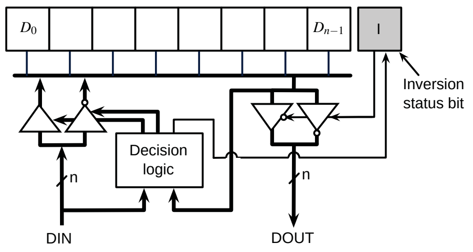

Fig. 4. Integration of the read-before-write and the data inverting scheme. The bold lines are n-bit width buses, where n is the size of a sub-block.

We can eliminate these redundant writes by performing a pre-read operation, which we call a read-before-write scheme. After reading the current data value of a target cache block, we compare it with the new data value to be written and apply either a SET or a RESET current pulse only if the two values are different.

2) Data Inverting: To further reduce the number of writes to PCM cells, we suggest employing a data inverting scheme in the PCM cache write logic. When we write a new data value to a cache block, we first read its current data value, and compute the Hamming distance (HD) between the two values. If the calculated HD is larger than the half of the cache block size, we invert the new data value before store it. Also, we set the inversion status bit to 1 to denote that the stored value has been inverted. When we read the stored data value, we refer to the inversion status bit to recover the original data values; we invert the read data value if its inversion status bit is 1.

We can increase the efficiency of the data inverting scheme by splitting a cache block into multiple sub-blocks and performing data inversion for each sub-block. However, the sub-block inverting scheme incurs area overhead because each sub-block requires its own inversion status bit. Hence, we need to carefully choose the size of a sub-block, considering the tradeoff between the efficiency and the area overhead, which will be analyzed in detail later.

3) Decision Logic: Both the read-before-write scheme and the data inverting scheme perform a pre-read operation, so we can efficiently merge these two schemes into a decision logic, as shown in Figure 4. The decision logic has two inputs: one is the new data value and the other is the current value read from the target cache block. It has a HD comparator, of which the output is 1 if the HD is larger than the half of its cache block (or its sub-block) size, and 0 if not. It refers to the two input data value and the output of the HD comparator to generate the enable signals of the buffer and the inverter.

B. Wear Leveling

1) Bit-Line Shifting: We can significantly reduce the total amount of writes to the PCM cache using the read-before-write and the data inverting technique. However, we found that the writes show an extremely uneven distribution in a cache block, as shown in Figure 5. This means that only a few hot PCM cells are worn out soon, making the entire cache useless even though most PCM cells are still functional. Hence, we suggest using a bit-line shifter to spread out the writes over the whole PCM cells in a cache block. As the tag array shows a similar uneven distribution to Figure 5, we use a tag shifter as well.

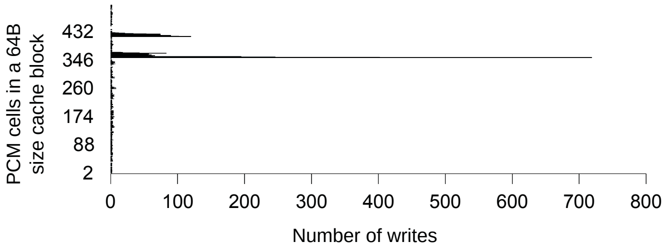

Fig. 5. Write distribution of the cache block containing the hottest PCM cell (tag bits are not shown here). Freqmine was run for 1037 ms on a 32 KB I/D cache and 4 MB PCM L2 cache, and the read-before-write scheme is applied.

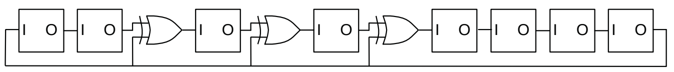

Fig. 6. Implementation of an 8-bit Galois LFSR. Each PCM flip flop has an XOR gate to implement the read-before-write scheme, which is not shown here.

In our design, each cache block has its own shift offset register (SOR) and shift interval counter (SIC). An SOR stores the shift offset of its cache block, and the bit-line shifter refers to it to determine how many bits to shift the incoming data before store it to the cache block. An SIC records the number of writes performed to its cache block, and when it reaches to a predefined threshold, we update the corresponding SOR value. This architecture makes hot cache blocks to change their shift offsets frequently, while keeping cold cache blocks from excessive updates of their shift offsets. Hence, we can achieve the balance between the wear-leveling performance and the additional bit changes caused by changing shift offsets.

2) Shift Offset Register (SOR): We implemented bit-line SORs using PCM cells because (1) SRAM SORs may incur significant area overhead as each cache block requires an SOR, and (2) the shift offset values in the SORs should be preserved during power failure in order to recover the original data from the stored data in the PCM cache.

We should consider the followings while implementing PCM SORs:

- Performance: PCM SORs may increase the cache access latency because reading a PCM SOR is required for every cache access. In fact, however, it can be hidden for both the cache read and write operations. First, for the cache read operation, it can be performed in parallel with the read operation of the cache block. Second, for the cache write operation, we also perform a cache block read operation because we use the read-before-write scheme and the data inverting scheme.
- Energy: PCM SORs consume more write energy than SRAM SORs. We used the read-before-write scheme for PCM SORs to reduce their write energy consumption.
- Endurance: As an SOR is updated only once per the period of its corresponding SIC, it does not limit the cache lifetime. We confirmed by experiments that the lifetime of SORs are longer than the PCM data array for the shift interval we used.

For every shift interval of an SIC, we need to choose a new value to update the corresponding SOR. One possible implementation is to add a predefined constant to the previous value, but its optimal value may change depending on the application running. Instead, we suggest using a Galois linear feedback shift register (LFSR), which can efficiently generate a pseudo random number sequence by using only a few exclusive or (XOR) gates. Figure 6 shows the circuit implementation of an 8-bit Galois LFSR, of which the period is 255 and the generated value ranges from 1 to 256.

TABLE I
SIC CODING CANDIDATES

Sequence	Binary	Gray	Balanced Gray	Proposed
1	0000	0000	0000	00000000
2	0001	0001	1000	00000001
3	0010	0011	1100	00000011
4	0011	0010	1101	00000111
5	0100	0110	1111	00001111
6	0101	0111	1110	00011111
7	0110	0101	1010	00111111
8	0111	0100	0010	01111111
9	1000	1100	0110	11111111
10	1001	1101	0100	11111110
11	1010	1111	0101	11111100
12	1011	1110	0111	11111000
13	1100	1010	0011	11110000
14	1101	1011	1011	11100000
15	1110	1001	1001	11000000
16	1111	1000	0001	10000000
N_{max}	16	8	4	2
N_{total}	30	16	16	16

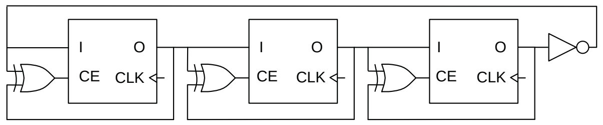

Fig. 7. Implementation of a 3-bit SIC with the proposed coding. An XOR gate is used for each PCM flip-flop to implement the read-before-write scheme.

3) Shift Interval Counter (SIC): For the similar reason to SORs, we implemented SICs using PCM cells. Updating PCM SICs can be performed in parallel with the cache block write operation, and thus it does not incur any performance degradation. However, their write energy consumption may not be ignorable because they are updated at every write to their associated cache blocks. Also, PCM SICs need appropriate endurance-enhancing techniques, or they will wear out before the cache blocks end their lifetime.

Through employing a proper coding, we can reduce the write energy of SICs as well as achieve sufficient lifetime. We present some SIC coding candidates for the shift interval of 16 writes in Table I, where N_{max} is the maximum number of bit changes for a PCM cell during a period, and N_{total} is the total number of bit changes occurred in a SIC. Note that N_{max} determines the lifetime of a SIC, and N_{total} is proportional to its write energy consumption. Using a binary code (the second column of Table I) shows that the LSB bit toggles for every writes, which is not acceptable. Instead, we may consider using the Gray code [12] to reduce both N_{max} and N_{total} because it changes only one bit between any two adjacent values. The third column of Table I shows that N_{max} is reduced to 8 and N_{total} to 16. We can again reduce N_{max} to 4 using the balanced Gray code [13], which perfectly distributes the number of bit changes among bits.

However, the combinatorial logics to implement the Gray code or the balanced Gray code are complicated, which may incur additional delay and area overhead. Hence, we suggest a new coding as shown in the fifth column of Table I, which can reduce N_{max} to 2 at the cost of increasing the number of PCM cells of the SIC. We can implement the proposed n-bit SIC using only an inverter, n XOR gates, and n PCM flip flops, as shown in Figure 7. The disadvantage of the suggested coding is that its maximum period is reduced from 2^n to $2n$. However, for small periods (e.g., 16 as in Table I), it can be compensated by the benefit of eliminating complicated endurance-enhancing techniques.

4) Word-Line Remapping: After we achieve wear leveling of the PCM cells in each cache block, there still exists uneven distribution of writes between cache blocks. Therefore, we use a word-line remapper to spread out writes over all cache blocks. The word-line remapper consists of an adder and a word-line SOR to keep the current word-line shift offset, through which a word-line decoder logic can determine what word-line should be enabled.

Like bit-line SORs, we need to store the word-line shift offset in a PCM SOR. However, we have to read it before every cache access, and thus its access latency is added to the critical path delay. Hence, we use a pair of an SRAM SOR and a PCM SOR as a word-line SOR to avoid performance degradation. When the word-line shift offset is changed, both the SRAM and PCM SORs are updated, but only SRAM SOR is accessed during normal operation. In the case of power failure, the word-line shift offset value can be recovered from the PCM SOR.

After changing the value of the word-line SOR, we should invalidate the contents in the cache because the word-line mapping is changed. Therefore, the word-line remapping period should be long enough (e.g., a second) to avoid excessive remapping overhead. We assume that an operating system periodically updates the word-line SOR rather than using a dedicated SIC.

IV. PCM Cache Modeling

A. Performance Model

The write latency of a PCM cache, T_{wr}, is expressed as

$$\begin{aligned} T_{wr} \quad = \quad & T_{wsor} + T_{add} + T_{wd} + T_{wl\_d} + T_{pcm\_rd} + T_{sh\_d} + \\ & T_{dec\_d} + T_{set}, \end{aligned} \qquad (1)$$

where T_{wsor}, T_{add}, T_{wd}, and T_{wl_d} are the delay of the word-line SOR, the adder, the decoder, and the word line of the cache block. T_{pcm_rd} is the latency of reading PCM cells in the data array, which includes the delay of the current-voltage converter and the sense amplifier. The read latency of the PCM bit-line SOR is not counted because it can be done simultaneously with reading the PCM cells in the data array. T_{sh_d}, T_{dec_d} and T_{set} are the delay of the bit-line shifter, the decision logic, and the SET operation of a PCM cell. Note that Equation (1) does not include any delays related to the tag write operation because it can be done at the same time to the data write operation, and its delay is smaller than that of the data part.

The read latency of a PCM cache, T_{rd}, is expressed as

$$T_{rd} = T_{wsor} + T_{add} + T_{wd} + T_{data} + T_{omux}, \qquad (2)$$

where T_{omux} is the delay of the output multiplexer and T_{data} is the time to read the tag part and the data part of a cache block, which is given by

$$T_{data} = T_{wl\_d} + T_{pcm\_rd} + T_{sh\_d}. \qquad (3)$$

B. Energy Model

We define E_{wr} to be the dynamic energy of a cache block write operation, which is expressed as

$$E_{wr} = E_{wsor} + E_{add} + E_{wd} + E_{wr\_d} + E_{wr\_t}, \qquad (4)$$

where E_{wsor}, E_{add}, and E_{wd} are the dynamic energy of the word-line SOR, the adder, and the word-line decoder, respectively. E_{wr_d} and E_{wr_t} respectively are the dynamic energy of a write operation to the data and tag part of a cache block, and are expressed as follows:

$$\begin{aligned} E_{wr}\_\mathrm{x} \quad = \quad & E_{wl}\_\mathrm{x} + (N_\mathrm{x} + N_{bsor}\_\mathrm{x})(E_{pcm\_rd} + E_{sa}) + E_{ht\_rd} + \\ & E_{sh}\_\mathrm{x} + E_{dec}\_\mathrm{x} + N_{set}\_\mathrm{x} E_{set} + N_{rst}\_\mathrm{x} E_{rst} + \\ & E_{ht\_wr}, \end{aligned} \qquad (5)$$

where E_{wl}_x, E_{sa}, E_{sh}_x, E_{dec}_x are the dynamic energy of the word line, the sense amplifier, the bit-line shifter, and the decision logic, respectively. E_{ht_rd} and E_{ht_wr} respectively are the dynamic energy of the cache read and write operation on the H-tree route wire. E_{pcm_rd} is the read energy of a PCM cell, and E_{set} and E_{rst} are the energy for the SET and RESET operations of a PCM cell, respectively. N_{bsor}_x is the width of the bit-line SOR, and N_d (N_t) is the number of PCM cells in the data part (the tag part) of a cache block, which includes the PCM cells used for storing inversion status. N_{set}_x and N_{rst}_x are the number of PCM cells that need the SET and RESET operation, respectively, which include the PCM cells used for storing inversion status and for bit-line SORs and SICs. Note that the last term E_{wr_t} in Equation (4) is counted only for cache block fetch operations; it is not counted for normal cache write operations.

We also define E_{rd} to be the dynamic energy of a cache block read operation, which is expressed as

$$E_{rd} = E_{wsor} + E_{add} + E_{wd} + N_{way}(E_{rd\_t} + E_{rd\_d}) + E_{comp} + E_{omux}, \qquad (6)$$

where N_{way} is the number of ways in the cache, and E_{comp} and E_{omux} are the dynamic energy of the tag comparator and the output multiplexer. E_{rd_d} and E_{rd_t} respectively are the dynamic energy of a read operation to the data and tag part of a cache block, and are expressed as follows:

$$\begin{aligned} E_{rd}\_\mathrm{x} \quad = \quad & E_{wl}\_\mathrm{x} + (N_\mathrm{x} + N_{bsor}\_\mathrm{x})(E_{pcm\_rd} + E_{sa}) + E_{ht\_rd} + \\ & E_{sh}\_\mathrm{x}. \end{aligned} \qquad (7)$$

We define P_{leak} to be the leakage power of a PCM cache, which can be calculated by summing the leakage power of all components in the cache, and is expressed as

$$P_{leak} = P_{wsor} + P_{add} + P_{wd} + P_{wl} + P_{sa} + P_{ht} + P_{dec} + P_{sh}, \quad (8)$$

where P_{wsor}, P_{add}, P_{wd}, P_{wl}, P_{sa}, P_{ht}, P_{dec}, and P_{sh} are the leakage power of the word-line SOR, the adder, the word-line decoder, the word line, the sense amplifier, the H-tree line, the decision logic, and the bit-line shifter, respectively.

C. Endurance Model

The lifetime of a PCM cache (in seconds) is expressed as follows:

$$L_{actual} = \frac{w_{pcm}}{w_{max}}, \qquad (9)$$

where w_{pcm} is the number of writes allowed to a PCM cell and w_{max} is the number of writes per second of the hottest PCM cell. We can estimate the upper bound of L_{actual} assuming that a wear-leveling scheme achieves a perfectly even distribution of writes, which is expressed as follows:

$$L_{upper} = \frac{w_{pcm} N_{pcm}}{\sum_{i=1}^{N_{pcm}} w_i}, \qquad (10)$$

where N_{pcm} is the total number of PCM cells in the PCM cache and w_i is the number of writes per second of ith cell.

TABLE II
TARGET SYSTEM SPECIFICATION

Component	Description
CPU core	Number of cores: 1, frequency: 2.0 GHz
L1 Cache	32/32 KB SRAM I/D cache, write-back and write-allocate policy Set-associativity: 1-way (directly mapped) Cache line size: 32 bytes, access latency: 3 cycles
L2 Cache	4 MB PCM unified cache, write-back and write-allocate policy Set-associativity: 4-way, cache line size: 64 bytes Access latency (read/write): 31/311 cycles
Main memory	DDR SDRAM, size: infinite (assuming no page fault) Access latency: 190 cycles, data bus width: 64 bits

D. Area Model

The area of a PCM cache is expressed as follows:

$$A = N_{pcm}A_{pcm} + A_{wsor} + A_{add} + A_{wd} + A_{sa} + A_{sh} + A_{dec} + A_{omux}, \quad (11)$$

where N_{pcm} is the total number of PCM cells used and A_{pcm} is the area of a PCM cell. A_{wsor}, A_{add}, A_{wd}, A_{sa}, A_{sh}, A_{dec}, and A_{omux} are the area of the word-line SOR, the adder, the word-line decoder, the sense amplifier, the bit-line shifter, the decoder, and the output multiplexer, respectively.

V. EVALUATION

A. Experimental Setup

We assumed 10^8 of PCM cell write endurance, which are used for recent previous work [4][5]. We used the PCM memory simulator [14], which is a PCM-supporting variant of the CACTI tool, to estimate the word-line and bit-line delay and their energy consumption of a given size of PCM cache memory. We used 45*n*m technology with that tool. The tool reports also the delay, energy, and area values of the PCM cache sub-components such as a current-voltage converter and a sense amplifier. We synthesized the logics described in Section III using the Synopsis Design Compiler with 45*n*m technology to get their delay, energy, and area values.

We developed a PCM cache simulator and integrated the obtained PCM cache model to evaluate our PCM cache design. It consists of a CPU trace fetch unit, SRAM-based L1 instruction and data cache memories, a PCM-based unified L2 cache, and a DDR SDRAM main memory. The CPU trace fetch unit supports the variable-length instruction set of Intel CPUs. All the cache modules and SDRAM modules are implemented so as to simulate not only address references but also actual data values. In addition, the PCM L2 cache module supports tracking the exact number of writes at a bit-level granularity, enabling accurate estimation of its lifetime. The target system specification used for our experiments is shown in Table II, where the PCM cache latency is obtained from Equations (1) and (2). For the input of the simulator, we used the Pin tool [15], as it supports capturing not only their address values but also data values through customized instrumentation. We selected 11 applications from the PARSEC benchmark suite [11]: blacksholes, bodytrack, canneal, facesim, ferret, fluidanimate, freqmine, raytrace, streamcluster, swaptions, and x264[1]. We captured 200 million instructions and their relevant data accesses of each application.

B. Experimental Results

We compared the lifetime upper bound and the energy consumption of a PCM cache between the baseline configuration, the read-before-write (RBW) scheme, and the data inverting scheme with various sub-block sizes. Figure 8 shows that applying RBW can dramatically increase L_{upper} and reduce the total energy consumption from 8% to 25% for three benchmarks. The area of RBW is 8% larger than that of the baseline cache. Figure 8 also shows that the data inverting scheme can further increase L_{upper}. Its efficiency is dependent on the sub-block size; a smaller sub-block size can achieve longer lifetime at the cost of area. The sub-block size of 16 (denoted by DI16) is a good choice for all applications considering the tradeoff between the achieved lifetime and the area overhead. Intuitively, the data inverting scheme is expected to also reduce the cache energy consumption, but it rather increases energy consumption compared to the RBW, as shown in Figure 8(b). It is because the additional read operations for the inversion status bits resulted in the increase of cache read energy, overwhelming the cache write energy reduction achieved from using the data inverting scheme. If read operations dominate the cache access, the energy overhead of the data inverting scheme becomes worse (e.g., raytrace), but in the opposite case, such an overhead can be mitigated (e.g., freqmine).

We performed experiment to decide the proper bit-line shift interval. The third column of Table III shows that a too short shift interval can noticeably reduce the value of L_{upper} because it generates additional bit changes. On the other hand, the last column shows that a too long period incurs considerable area overhead because of the increased SIC size. We chose the value of 16 because it is a good compromise between these two extremes (9.5% loss of L_{upper} and 3.6% of area overhead).

[1]We did not include vips and dedup because they were not compiled or caused a segmentation fault on our machine.

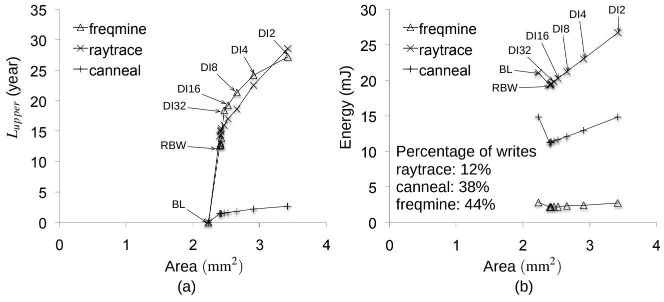

Fig. 8. The effect of data inverting scheme on the lifetime and the energy consumption (BL: baseline cache, RBW: read-before-write scheme, DI*n*: data inverting scheme with the sub-block size of *n*).

TABLE III
THE EFFECT OF CHANGING BIT-LINE SHIFT INTERVAL

Shift interval (# of writes)	SIC size (# of cells)	L_{upper} (year)	Energy (mJ)	Cache area (mm^2)
0 (no shift)	0	14.71	11.35	2.53
1	0	9.25	11.49	2.53
4	2	12.49	11.4	2.55
16	8	13.32	11.37	2.62
64	32	13.58	11.37	2.91

Note: freqmine was run five times, and DI16 is used.

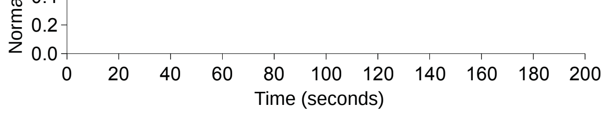

Fig. 9. The change of L_{actual} over time. Ferret was run repeatedly, and the y-axis is normalized to the value of L_{upper}.

TABLE IV
COMPARISON BETWEEN VARIOUS CACHE CONFIGURATION

App.	Baseline (Area: 2.23 mm^2)		RBW (Area: 2.49 mm^2)		DI16 (Area: 2.62 mm^2)	
	Energy (mJ)	Lifetime (day)	Energy (mJ)	Lifetime (year)	Energy (mJ)	Lifetime (year)
Blackscholes	8.91	0.22	8.56	20.41	8.95	24.04
Bodytrack	5.73	0.78	4.67	2.03	4.80	2.74
Canneal	14.88	1.83	11.28	1.19	11.72	1.34
Facesim	2.58	1.66	2.09	10.29	2.15	12.12
Ferret	18.04	0.69	14.69	1.00	15.22	1.30
Fluidanimate	7.81	2.02	7.15	4.31	7.43	5.48
Freqmine	2.85	1.39	2.20	11.47	2.26	16.96
Raytrace	21.07	0.01	19.42	8.54	20.33	10.70
Streamcluster	16.90	0.94	16.62	6.52	17.40	8.36
Swaptions	3.54	0.02	2.92	11.80	3.02	14.26
x264	23.54	0.10	21.54	2.30	22.53	2.78
Average	11.44	0.04	10.10	3.07	10.53	3.80

Note: arithmetic mean is used for energy values, and harmonic mean is used for lifetime values.

The preferable value of a word-line remapping period would be larger than a second considering the word-line remapping overhead. However, such a long time interval slows down the converging speed of L_{actual} to L_{upper}, preventing us to observe the convergence in a reasonable simulation time. As a workaround, we accelerated the converging speed of L_{actual} by reducing the PCM cache size to 128KB and setting the period to 100 ms. Figure 9 shows the simulation result for 200 seconds of execution time, which corresponds to about eight hours of simulation time on our machine. For the first ten seconds, the value of L_{actual} rapidly increased to that of L_{upper}, and then its converging speed is slowed. After 100 seconds of simulated time, L_{actual} becomes 92.7% of the value of L_{upper}. Although we expect that we can eventually achieve a pretty good efficiency (e.g., more than 99%) for a few days of running time in real systems, we conservatively took this result as the word-line remapping overhead in this paper.

Table IV shows the energy consumption and the lifetime of three cache configurations, where we estimated the lifetime value using $0.927L_{upper}$ as mentioned before. Note that the bit-line wear leveling overheads are already included in the value of L_{upper}. Using the RBW scheme with the wear-leveling schemes could achieve 12% of energy saving on average and 3.0 years of lifetime, and its area overhead was 12%. By investing another 5% of the baseline cache area, we could apply the data inverting scheme with the wear-leveling schemes, which resulted in 24% improvement of lifetime (3.8 years) compared with the RBW, while still achieved 8% of average energy saving. We also found strong relation between the energy reduction and the read-write ratio of each application; if the write operations are dominant, the energy saving could be up to 24% for RBW and 21% for DI16 configuration.

VI. Related Work

There are many previous work using the same principle of operation with our data inverting scheme to reduce the energy consumption of off-chip buses or bit- and word-lines in memory components. The bus invert coding scheme [16] is proposed to reduce bus switching activity by adding an additional bus line, which transfers the inversion status. The inverted data store scheme [17] selectively inverts the data to be stored in the ROM so that the difference between the precharging value and the data value is minimized. Both the schemes aim at reducing the dynamic energy consumption due to the high capacitance of bus lines or bit- and word-lines in a memory component.

As PCM consumes pretty high write energy compared to its read energy, many architectural researches on PCM suggest using various methods to reduce the write traffic to PCM. Some researchers [4][18] used a similar approach to the read-before-write scheme, while another researcher [6] proposes a line-level write scheme, which writes only the dirty lines. Especially, a PCM write logic [18] combining both the read-before-write scheme and the bus invert coding scheme is suggested to aggressively reduce the write energy of PCM-based memory, which is similar to the decision logic described in Figure 4. However, our work is differentiated from the previous work [18] in that we focused on extending the lifetime of a PCM cache using the data inverting scheme.

VII. Acknowledgment

This work was supported by Korea Research Foundation Grant funded by the Korean Government (KRF-2008-357-D00208). This work was also supported in part by NSF grants (0905365, 0643902, 0916887, 0903432) and SRC grant.

VIII. Conclusion

PCM caches have many benefits such as high-density, non-volatility, low leakage power, and high immunity to soft error. However, its slow write speed, high write energy, and limited write endurance prevent it from replacing SRAM caches. In this paper, we suggested guidelines to design a PCM cache; we studied several techniques to mitigate such a limitation of a PCM cache, and provided detailed timing, energy, endurance, and area models for PCM caches. We also developed a PCM cache simulator integrated with the PCM cache models to evaluate our PCM cache design. By properly using energy- and endurance-aware techniques, we could achieve 8% of energy saving and 3.8 years of lifetime with 17% of area overhead compared with the baseline PCM cache.

References

[1] K.-J. Lee *et al.*, "A 90nm 1.8V 512Mb diode-switch PRAM with 266MB/s read throughput," in *ISSCC* 2007, pp. 472–616.
[2] J. K. Kim *et al.*, "A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems," in *EMSOFT* 2008, pp. 31–40.
[3] Y. Park *et al.*, "PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash," in *SAC* 2008, pp. 1498–1503.
[4] P. Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," in *ISCA* 2009, pp. 14–23.
[5] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *ISCA* 2009, pp. 2–13.
[6] M. K. Qureshi *et al.*, "Scalable high performance main memory system using phase-change memory technology," in *ISCA* 2009, pp. 24–33.
[7] G. Dhiman *et al.*, "PDRAM: A hybrid PRAM and DRAM main memory system," in *DAC* 2009, pp. 664–669.
[8] P. Mangalagiri *et al.*, "A low-power phase change memory based hybrid cache architecture," in *GLSVLSI* 2008, pp. 395–398.
[9] X. Wu *et al.*, "Hybrid cache architecture with disparate memory technologies," in *ISCA* 2009, pp. 34–45.
[10] S. Raoux *et al.*, "Phase-Change Random Access Memory: A Scalable Technology," *IBM J. Res. Develop.*, vol. 52, no. 4/5, 2008.
[11] C. Bienia *et al.*, "The PARSEC benchmark suite: characterization and architectural implications," in *PACT* 2008, pp. 72–81.
[12] F. Gray, "Pulse code communication," US Patent 2 632 058, March 15, 1953.
[13] G. Bhat *et al.*, "Balanced Gray codes," *Electr. J. Comb.*, vol. 3, pp. 2–5, 1996.
[14] X. Dong *et al.*, "PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM," in *ICCAD* 2009, pp. 269–275
[15] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI* 2005, pp. 190–200.
[16] M. R. Stan *et al.*, "Bus-invert coding for low-power I/O," *IEEE TVLSI*, vol. 3, no. 1, pp. 49–58, 1995.
[17] Y.-S. Chang *et al.*, "Conforming inverted data store for low power memory," in *ISLPED* 1999, pp. 91–93.
[18] W. Xu *et al.*, "Data manipulation techniques to reduce phase change memory write energy," in *ISLPED* 2009, pp. 237–242.