# Energy-Efficient Multi-Level Cell Phase-Change Memory System with Data Encoding

Jue Wang, Xiangyu Dong, Guangyu Sun, Dimin Niu, Yuan Xie
Department of Computer Science and Engineering
Pennsylvania State University
University Park, Pennsylvania, USA 16802
Email: {jzw175,xydong,gsun,dun118,yuanxie}@cse.psu.edu

***Abstract*—Phase-change memory (PCM) is one of the most promising technologies among emerging non-volatile memories. Recently, the technology of multi-level cell (MLC) for PCM has been developed and a high capacity memory system can be implemented by storing multiple bits in a cell. However, programming MLC PCM involves the program-and-verify scheme. Thus, the energy of programming intermediate states in MLC PCM is considerably larger than that of single-level cell (SLC) PCM. To mitigate the MLC energy overhead, we propose an energy-efficient PCM architecture using data encoding write based on the observation that there are significant value-dependent energy variations in programming MLC PCM. In addition, data comparison write (DCW) is adopted to enhance the effectiveness of the proposed data encoding architecture for MLC PCM. Simulation results show that this encoding architecture achieves 9.6% average energy saving (up to 19.8%) on the plain MLC PCM system, and 12.9% average energy saving (up to 26.7%) on the DCW-adopted MLC PCM system[1].**

## I. Introduction

Phase-change memory (PCM), one of the future non-volatile memory technologies, is based on the phenomenon that the phase of chalcogenide materials can be switched between amorphous and crystalline states by using electric currents. PCM has many attractive features including non-volatility, negligible standby leakage, fast read access, high cell density, and superior scalability [1]. PCM has made rapid progress in the recent years, and it is considered to have the read access latency that is comparable to those of SRAM and DRAM and the non-volatility like NAND flash [2], [3]. Therefore, there has been extensive research of using PCM at various levels of the memory hierarchy, for instance in the last-level caches [4]–[6] and the main memory [2], [7], [8].

Most recently, the feasibility of multi-level cell (MLC) for PCM including programming into two and four bits per cell has been shown [9]–[11]. Although MLC increases the PCM bit density, the energy of programming MLC PCM is considerably larger than that of single-level cell (SLC) PCM [3], [8], [12]. The extra energy consumed by MLC PCM comes from the necessity of program-and-verify (P&V) scheme which causes multiple programming steps per MLC write operation for intermediate states, and this effect is similar to the well-known energy consumption difference in MLC and SLC NAND flash designs [13]. Considering the general estimation that PCM consumes 2.2 times more energy than DRAM does [2], it is necessary to design an energy-efficient architecture for the MLC PCM system and reduce the programming energy by manipulating the data stored in MLC PCM.

[1]This work is supported in part by SRC grant, NSF 1147388, 0903432 and by DoE under Award Number DE-SC0005026.

## II. MLC PCM Background and Energy Model

PCM technology is based on the phase-change behavior of chalcogenide alloys (GST). The data storage capability is achieved by the resistance differences between the amorphous (high-resistance) and the crystalline (low-resistance) phase of GST. To SET the cell into its low-resistance state, an electrical pulse is applied to heat a significant portion of the cell above the crystallization temperature. This SET duration mainly depends on the crystallization speed of GST. Although SET pulses shorter than 10ns have been demonstrated [14], the typical value of the SET pulse duration is around 150ns [15]. On the other side, in the RESET operation, a larger electrical current is applied in order to melt the central portion of the cell. After this pulse is cut off abruptly, the molten material quenches into the amorphous phase. The RESET operation has shorter duration but tends to be current-hungry [15]. Generally, the energy consumptions of full RESET and SET operations are on the same order of magnitude.

Thanks to the large resistance contrast between the RESET and SET states (e.g. $10^2 - 10^3$), MLC PCM becomes feasible. However, the degree of success of such an MLC write depends on the resistance distributions over a large ensemble of PCM cells. Unlike SLC write, where the bit write quality can be ensured by over-SET or over-RESET, the intrinsic randomness associated with each write attempt and the inter-cell variability make it impractical to have a universal pulse shape for writing an intermediate state. In order to address this issue, resistance distribution tightening techniques have been developed based on the program-and-verify (P&V) technique [3]. P&V is a common programming technique for multi-bit writing and is widely used in MLC NAND flash products [13] and MLC PCM prototypes [9], [10]. In order to achieve non-overlapping resistance distributions of different bit levels, P&V needs to iteratively apply partial set pulses and then verify that a specified precision criterion is met, which leads to much longer write latency and hence the much larger programming energy.

978-1-4577-1954-7/11/$26.00 ©2011 IEEE

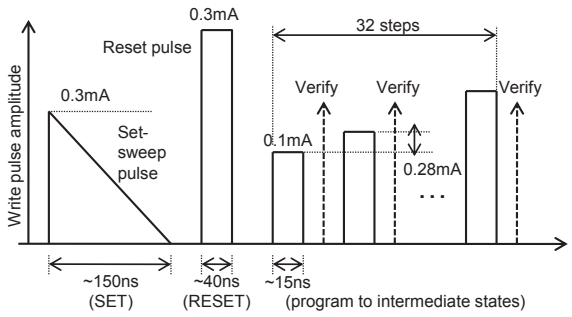


Fig. 1. The pulse shapes of complete SET and RESET operations and the modeled program-and-verify scheme by using partial SET pulses.

Similar to NAND flash, the MLC PCM programming energy can be more than 10 times of the SLC one.

In this work, we evaluate 2-bit MLC PCM, of which four states can be stored within each cell. Bit patterns "00" and "11" are stored using complete RESET and complete SET state, respectively. Intermediate resistance states in the order of increasing resistance are used to represent combinations "10" and "01". Fig. 1 demonstrates the concept of the P&V programming technique for MLC PCM. Instead of applying a full SET-sweep pulse to securely set the cell to SET state or applying a powerful RESET pulse to set the cell to RESET state, intermediate MLC states have to be programmed in an iterative manner. In this work, we refer to an MLC PCM prototype design from Bedeschi *et al.* [9], and we use the SET-sweep pulse with $0.2mA$ peak current and $150ns$ duration and the RESET pulse with $0.3mA$ amplitude and $40ns$ duration. For the partial SET pulses, we assume that there are 32 possible P&V steps in total. To program intermediate states, the cell has to experience a full SET-sweep pulse and a full RESET pulse as the initialization sequence to improve the programming quality of following steps [9]. After that, a sequence of partial SET pulses with $15ns$ duration are applied until the intermediate resistance level is reached. For programming intermediate state "10", the the partial SET pulse amplitude starts from $0.1mA$ with $28\mu A$ stepping; for programming "01", the partial SET pulse amplitude starts from $0.576mA$ with $28\mu A$ stepping[2].

Based on our energy model assumption, the energy consumption values of programming every state are listed in Table I. It should be noticed that process variation affects the MLC PCM behavior and the number of P&V rounds is different from cells to cells. Thus, the energy estimations in Table I are all average values.

From Table I, we can see that there are significant value-dependent programming energy variations in multi-level cell PCM. Programming RESET state "00" using full RESET pulse and SET state "11" using SET-sweep pulse need one order of magnitude less energy than programming other states. Thus, this phenomenon motivates us to propose a new MLC PCM architecture based on data encoding to increase the "11" and "00" states in the writing data and thereby reduce the programming energy.

[2]These parameters are scaled from an MLC PCM prototype with $90nm$ process node and bipolar-selected cells [9].

TABLE I
WRITE ENERGY OF PROGRAMMING EVERY STATE [9]

| State | Average current | Pulse duration | Average energy consumption |
|---|---|---|---|
| 00 | $300\mu A$ | $40ns$ | $36pJ$ |
| 01 | $100\mu A$ - $548\mu A$ | $220ns$ - $520ns$ | $307pJ$ |
| 10 | $576\mu A$ - $996\mu A$ | $220ns$ - $520ns$ | $547pJ$ |
| 11 | $200\mu A$ | $150ns$ | $20pJ$ |

**Algorithm 1** Data Encoding Algorithm

```
// N: memory line width
Write(A: address, D: data)
MT := Mapping Type (D); // get a mapping type
D_m := Map(D, MT); // map the data
for all i = 0, i ≤ N, i++ do
    write MLCs as D_m
end for
```

## III. Implementation of Energy-Efficient MLC PCM Data Encoding

In this section, the MLC PCM data encoding architecture is described. At first, we propose a data encoding algorithm that maximizes the frequencies of the "11" (i.e. full SET) and the "00" (i.e. full RESET) stored in MLC PCM. Then, the advantage and overhead of this algorithm are analyzed. In the end, the circuit architecture of such energy-efficient MLC PCM data encoding is discussed.

### A. Data Encoding Algorithm

The following terms are defined for the discussion of the data encoding algorithm:

- *Low Power States (LPS)*: the states which need less energy in programming, representing states "00" and "11" in 2-bit MLC PCM;
- *High Power States (HPS)*: the states which need more energy in programming, representing states "01" and "10" in 2-bit MLC PCM;
- *Original Data*: the data that are not encoded;
- *Encoded Data*: the data that are encoded according to the data encoding algorithm and are stored in PCM in an encoded form.

Basically, the key concept of the algorithm is to increase the total percentage of LPS in writing data by encoding the input data to ensure that the two most frequent states are mapped to "11" and "00". Therefore, the total writing energy can be reduced since programming "11" (i.e. full SET) and "00" (i.e. full RESET) states need less energy. Algorithm 1 captures the process of such data encoding.

In the encoding algorithm, the selection of *Mapping Type* is the critical part. For the special feature of MLC PCM programming, the mapping rules for this algorithm are designed as follows:

TABLE II
ENCODING LOOK-UP TABLE

| Mapping of two most frequent states | | Mapping of the other two states | | Mapping type |
|---|---|---|---|---|
| 00→00 | 11→11 | 01→01 | 10→10 | 0000 |
| 00→00 | 01→11 | 10→10 | 11→01 | 0001 |
| 00→00 | 10→11 | 01→01 | 11→10 | 0011 |
| 01→00 | 10→11 | 00→10 | 11→01 | 1100 |
| 01→00 | 11→11 | 00→01 | 10→10 | 1101 |
| 10→00 | 11→11 | 01→01 | 00→10 | 1111 |

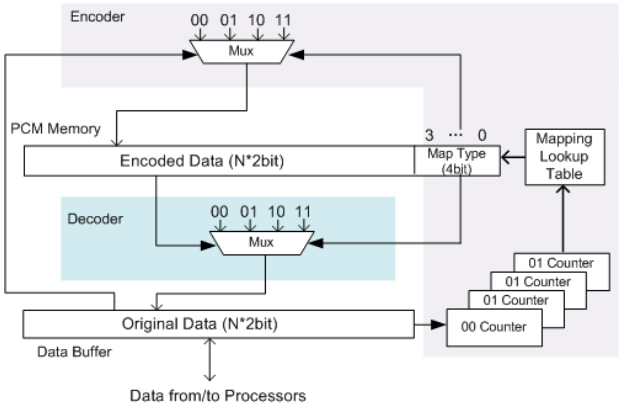


Fig. 2. The control and data flow of encoder and decoder.

- *Mapping Rule 1*: Map the two most frequent states to "11" and "00".
- *Mapping Rule 2*: Maintain the original data in the encoding operation as much as possible.

Rule 1 ensures that the states "11" and "00" are the tow most frequent states in the encoded data. Rule 2 is devised to reserve the effectiveness of data comparison write (DCW), which we discuss later in Section IV.

According to the mapping rules, there are $C_4^2 = 6$ different ways of mapping the four states. A look-up table is used to implement the data encoding algorithm, as shown in Table II.

For every memory line, the information of *Mapping Type* needs to be added to decode the encoded data. This mapping type needs 4 extra bits or 2 extra cells to present since there are 6 different mapping types and every memory cell has two bits. Because mapping types also need to be written to MLC PCM cells, we use as much as states of "00" and "11" to present the mapping type, as shown in Table II.

The control and data flow of the encoder and decoder is shown as Fig. 2. For every memory line, the original data is stored in a data buffer. Then, the percentage of every state is counted by the encoder to choose the mapping type using the mapping lookup table. For example, if the two largest frequency states of one memory line data are "01" and "11", the encoder choose mapping type as "1101" according to the mapping lookup table. Then, the encoded data is written as the rules: "00", "01", "10", "11" are encoded to "01", "00", "10" and "11", respectively. Thereby, the states "11" and "00" are the two most frequent states in the encoded data.

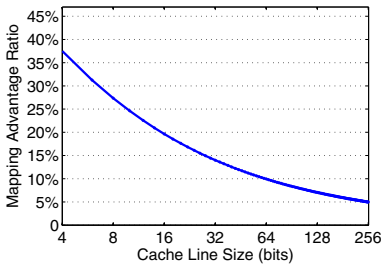


Fig. 3. The data encoding algorithm's advantage ratio (which indicates how many extra LPS have been added) versus the width of memory lines.

### B. Advantage and Overhead of Data Encoding Algorithm

For the original data, assume there are N 2-bit cells in a memory line and the probability of an MLC is LPS (i.e. "00" or "11") is $1/2$. Therefore, the probability of having $i$ LPS among the original data is $(1/2)^N C_N^i$ based on the binomial distribution. So the average percentage of LPS in $N$ cells is:

$$P_{LPS} = \sum_{i=0}^{N} \frac{i}{N} \frac{1}{2^N} C_N^i = \frac{1}{2} \tag{1}$$

Similarly, the average percentage of LPS in N cells after encoding is calculated as follows:

$$P'_{LPS} = \sum_{i=0}^{N/2} \frac{(N-i)}{N} \frac{1}{2^N} C_N^i + \sum_{i=N/2+1}^{N} \frac{i}{N} \frac{1}{2^N} C_N^i \tag{2}$$

In Equation 2, the result is split into two cases, of which the first part is mapping the HPS to LPS case since the LPS percentage is smaller ($i < N/2$) and the percentage of LPS is changed to $(N-i)/N$ after encoding. Therefore, we define the *data encoding algorithm's advantage ratio*, $R$, as:

$$R = \frac{P'_{LPS} - P_{LPS}}{P_{LPS}} = \sum_{i=0}^{N/2} \frac{(N-2i)}{N} \frac{1}{2^{N-1}} C_N^i \tag{3}$$

Compared with the original data, the percentage of LPS in the encoded data is increased by 37% (for a 4-bit memory line) or by 5% for a 64-byte memory line. The algorithm's advantage ratio is different depending on the width of memory lines as shown in Fig. 3. Moreover, this relative advantage is based on the assumption of random multi-bit value distribution. With realistic application, the benefits are different depending on different application workloads, which are shown in Section V.

Moreover, this encoding algorithm is a fix-length encoding, which can ease the memory accessing management. As shown in Fig. 2, the encoded data is the mapped data in addition of the mapping type. Therefore, if the width of original data is $N \times 2$ bits, the width of encoded data is $N \times 2 + 4$ bits, which equals to the width of one memory line in our design. For a 64-byte-per-line MLC PCM, the overhead of the data size is about 1.5%.

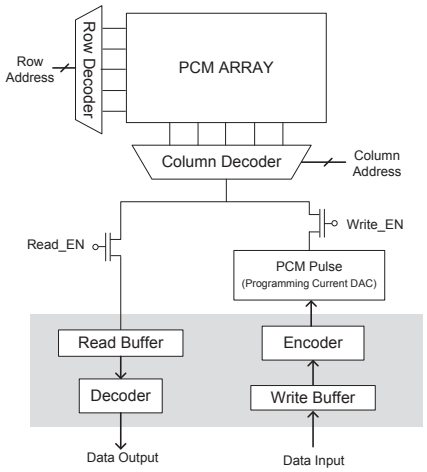


Fig. 4. PCM memory architecture including the extra components for data encoding and decoding.

### C. Data Encoding Hardware

The traditional memory structure needs to be modified in order to apply the data management function efficiently. A new architecture of MLC PCM is designed which has the capability to encode the input data before writing, and decode output data after reading. Therefore, write buffer, read buffer, encoder component, and decoder component are all required in this system. As shown in Fig. 4, the components in gray are the extra hardware components that are required by the purposed MLC PCM data encoding architecture.

The data access pattern in such an architecture is described as follows:

- *Write Operation*: Store a memory line in the write buffer before writing to the PCM. When data in buffer is ready, the encoder transforms the input data to the encoded data which needs less writing energy. Then, the programming component writes the result to PCM. Usually, processors' speed is much faster than writing to PCM's MLC, which requires about 270ns in the worst case according to our estimations as listed in Table I. Therefore, processor just need to store the memory line data to write buffer in this architecture and *Direct Memory Access* (DMA) makes the performance overhead negligible.
- *Read Operation*: When a memory line is read out, it is stored in the read buffer at first. Then decoder maps the encoded data to original data with the mapping type information, which is the least significant 4 bits of the memory line data.

## IV. Combine Data Encoding with DCW

Data comparison writes (DCW) is a common scheme for PCM system, which has been implemented in some PCM prototypes [16]. Therefore, in this section we study how to use the proposed encoding architecture in MLC PCM system with adopting DCW scheme.

### A. Introduction of DCW

DCW technique [17] is designed to remove the redundant bit writes. For MLC-2, the statistical bit-write redundancy is 25% if writing every state is equally likely.

The basic concept of DCW is preceding a write with a read. Write the cell only if the data is changed after writing. It will reduce the unnecessary write operations, which improve the endurance of PCM, and reduce the writing energy consumption. In PCM operations, reads are much faster and consume much less energy than writes. It is this asymmetry can we benefit from to improve the PCM endurance and reduce the write energy.

### B. Modified Data Encoding Algorithm for DCW

When DCW is adopted in this memory system, the question is how to combine the data encoding and DCW techniques efficiently. We propose two techniques to solve this problem: *Maximum Maintained Mapping* and *Energy Hamming Distance Comparison*.

- *Maximum Maintained Mapping:*

In the encoding algorithm, Rule 2 is to maintain the original data in the encoding operation as much as possible. It ensures that most of the data in the mapping will not be changed. From the encoding table, we can see that the Hamming distance between original data and encoded data is $N/2$ if writing every state is equally likely.

- *Energy Hamming Distance (EHD) Comparison:*

When encode one memory line, we can choose from two mapping types: the old mapping type which is read from the old memory data and the new mapping type which is calculate from the new data. The more similar the new data and old data are, the more efficient DCW is. In this case, the old mapping type should be chosen to save more energy using DCW. Otherwise we should choose the new mapping type to save energy through data encoding. To decide which case every data line belongs, we define Energy Hamming Distance (EHD) to quantify this problem. EHD is the Hamming Distance between two data with the weights which are set as the states' writing energy. Before writing the data, calculate the EHD between the old data and the new data encoded as the old mapping type, as well as the old data and the new data encoded as the new mapping type. Then the one which has smaller EHD is chosen to write to memory through DCW scheme, so that the effectiveness of encoding and DCW both can be maximally reserved. Algorithm 2 captures the scheme combing data encoding and DCW.

### C. Modified Architecture of MLC PCM for Combining Data Management and DCW

Some modifications are needed to implement on the PCM architecture to combine data encoding management and DCW. The modified PCM memory read/write data path architecture is shown in Fig. 5, which is based on Fig. 4. The gray part in Fig. 5 is added for DCW and the other part is the same as the architecture in Fig. 4. The *Diff* and *EHD Count* are used

**Algorithm 2** Combining Data Encoding with DCW

// N: memory line width
Write($A$: address, $D$: data)
$D_{old\_m(old)}$, $MT_{old}$ := Read($A$)
// Map new data as old mapping type
$D_{new\_m(old)}$ := Map($D$, $MT_{old}$);
// Get the new mapping type
$MT$ := Mapping Type ($D$);
// Map new data as new mapping type
$D_{new\_m(new)}$ := Map($D$, $MT$);
**if** EHD($D_{old\_m(old)}$, $D_{old\_m(old)}$) $\leq$ EHD($D_{new\_m(new)}$, $D_{old\_m(old)}$) **then**
  $MT$ := $MT_{old}$; // Keep the old mapping type
**end if**
**for all** $i = 0, i \leq N, i++$ **do**
  **if** Map($D$, $MT$) $\neq$ $D_{old\_m(old)}$ **then**
    update MLCs to Map($D$, $MT$)
  **end if**
**end for**

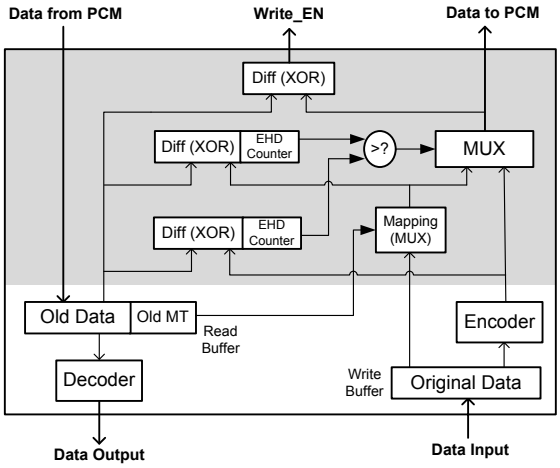


Fig. 5. Modified PCM memory read/write data path architecture combining the data encoding components and DCW.

to calculate the Energy Hamming Distance of two input data. Depending on which distance is smaller, the mapping type with smaller EHD are selected to save more energy. It should be noticed that the only difference of these two choices is to use the new mapping type or the old one which is stored in PCM. As a result, the decoder part does not need to change since it can decode all the data as encoded data using mapping type information.

The performance overhead of adding this part is small because the gates and reading operation is much smaller than the MLC PCM write latency. On the other hand, only several XOR gates, counters, and multiplexers are needed and only one block is added in PCM data path. Thus, the overhead of hardware is also negligible. We discuss more details about the hardware overhead in Section V.

TABLE III
BASELINE CONFIGURATIONS

| | |
|---|---|
| Processor | 4-core, 3.2GHz, SPARCV9-like cores |
| I-L1/D-L1 caches | private, 64KB/64KB, 2-way, 64-Byte cache line |
| L2 cache | shared, 4MB, 16-way, 64-Byte cache line |
| PCM module | 4GB, 128-entry write buffer |

## V. EXPERIMENTAL RESULTS

In this section, we evaluate the energy improvement after applying the data encoding-based MLC PCM architecture.

### A. Experiment Methodology

Our experiment simulates a 3.2GHz chip-multiprocessor with four SPARCV9-like cores. Each core has their private 2-way-associative I-L1 and D-L1 caches with the identical capacity of 64KB. The 16-way-associative L2 cache is shared by all the cores and has a capacity of 4MB. The size of write buffer before PCM memory is set to 128 entries. The write buffer is large enough to mitigate the impact of long write latency of PCM memory. For all the benchmarks, the write buffer is never full so that the read operation will not be blocked for a long time if there is a burst of write operations. The parameters of baseline configuration are listed in Table III.

For our 4-core system, our experimental evaluation makes use of 4-thread OpenMP version of workloads from PARSEC 2.1 [18] and SPEC OMP 3.2 [19] benchmark suites. The input size of the SPEC OMP benchmark is medium, and the native inputs are used for the PARSEC benchmark to generate realistic program behavior. We exclude 2 workloads from PARSEC and 1 workload from SPEC OMP[3], hence we evaluate 23 workloads in total. We collect the memory accesses to the PCM module by using the Simics full-system simulator [20]. Each Simics simulation run is fast forwarded to the pre-defined breakpoint at the code region of interest, warmed-up by 100 million instructions, and then simulated in the detailed timing mode for 1 billion cycles.

### B. Hardware Overhead

We evaluate the hardware overhead of the proposed encoding with DCW by using Design Compiler to estimate the area and energy consumption with $45nm$ TSMC CMOS library. According to the area analysis, the proposed encoder and decoder architecture incurs extra area of $0.025mm^2$, which is negligible compared to the area of PCM system. According to the energy analysis, the encoder and decoder incur extra energy consumption of $0.971pJ$ and $0.449pJ$ per memory line access, respectively. For different workloads, the read and write access numbers are different, which are show in Fig. 6. These results are used in our simulation to calculate the encoder and decoder's energy overhead.

[3] *blackscholes* and *swaptions* are excluded because these two workloads generate too few memory access traffic; *gafort* causes segmentation fault when executed in the parallel.

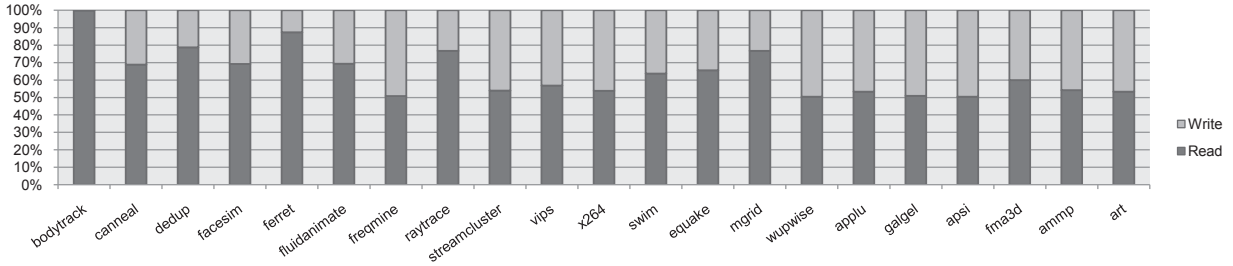


Fig. 6. Ratio of read and write memory access times

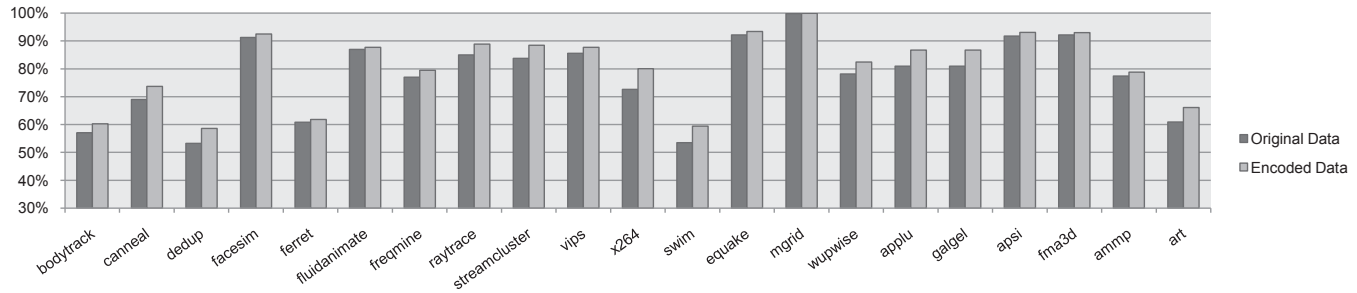


Fig. 7. Percentage of LPS in original data and encoded data.

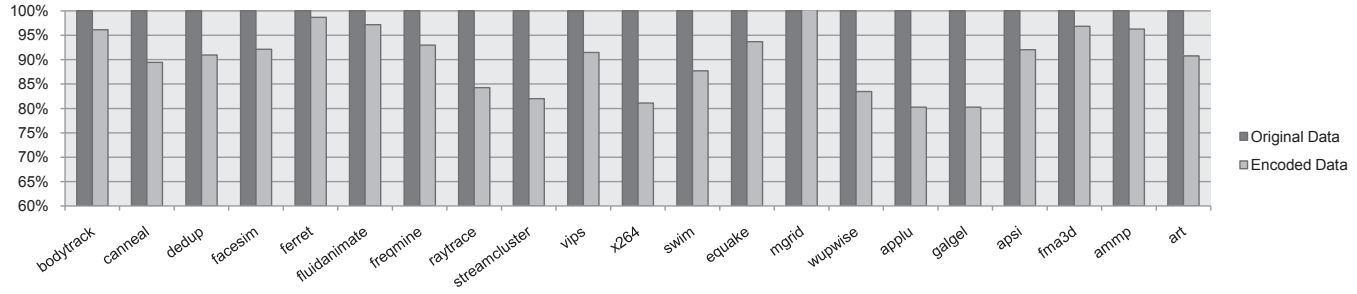


Fig. 8. Reduction of energy consumption after encoding.

### C. Evaluation of Data Encoding Algorithm

Fig. 7 shows the result of the LPS's percentage in the data. It shows that the LPS's percentage is increased by 4.6% on average (up to 11.1%). It can be seen that if the percentage of LPS in the workloads' original data is smaller, the advantage of the encoding technology is larger. It is because the improvement of LPS's percentage is limited when it is already large enough in the original data. Therefore, for *mgrid* from Fig. 7, the percentage of LPS is 99.9% in original data and is increased only 0.1% after encoding; for *swim*, the percentage of LPS is 53.5% in original data and is increased as much as 11.1% after encoding.

Fig. 8 compares the results of energy consumption for different applications with original data and encoded data, which include the energy of writing and reading, respectively. The energy consumption of writing every different state is listed in Table I. Besides, we also consider the energy of encoder and decoder for every write and read access. Fig. 8 shows that the total energy consumption is reduced by 9.6% on average of different workloads (up to 19.8%). Basically, if the increasing of LPS's percentage is larger, more energy is saved in the workload. However, if the number of memory read accesses is too large relative to that of write accesses, the overhead of decoder's energy will influence the effectiveness of the encoding technique since the decoder consumes energy for every reading access. From Fig. 7 and Fig. 8, we can see that for workload *swim* and *galgel*, the improvement of LPS's percentage is 11.1% and 7.1%. But, *galgel* saves 19.8% energy, which is more than *swim* of 12.3%, because decoder energy overhead of *galgel* is smaller since its ratio of write to read is larger than *swim*, which can be seen from Fig. 6.

### D. Evaluation of Combining Data Encoding and DCW

The benefits of combining data coding and DCW together are also evaluated. Fig. 9 shows the result of percentage of

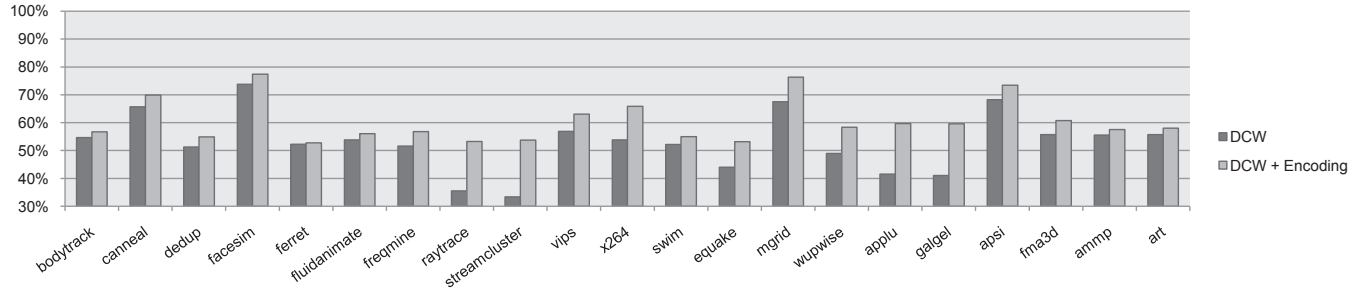


Fig. 9. Percentage of LPS in original data and encoded data after the DCW technique is adopted.

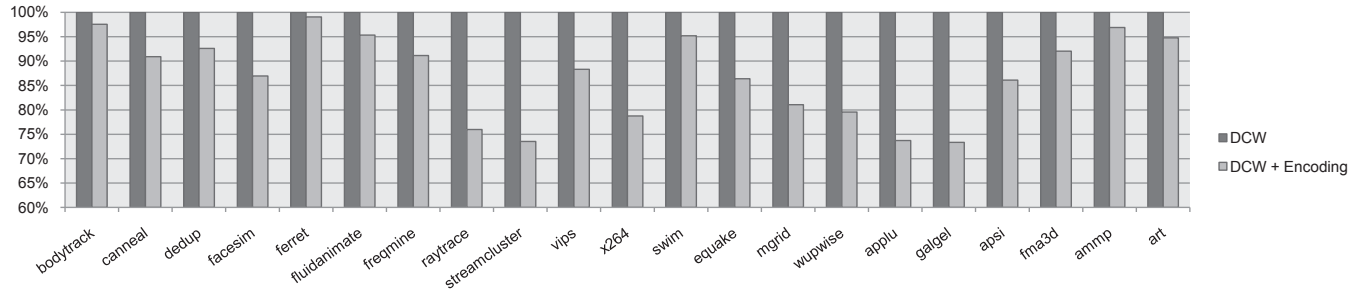


Fig. 10. Reduction of energy consumption after the combination of data encoding and DCW.

LPS in the writing data. In the first case, only the DCW technology is used. In the second case, encoding is used in together with DCW. The result shows that the percentage of LPS is increased by 16.8% on average (up to 60.9%). It can be noticed that generally the improvement of LPS's percentage is larger than the memory system without DCW. The reason is DCW technique eliminate the unnecessary writes, and most of them is writing state "00" repeatedly. So the percentage of LPS in the workloads' original data is smaller for the memory system with adopting DCW. Therefore, the effectiveness of data encoding becomes larger.

Fig. 10 shows that the total writing and reading energy of two different configurations: original data using DCW, encoded data using DCW. After adding the DCW scheme, every write access includes one read access. The energy consumption of these reading operations are also calculated in the simulation. Moreover, the energy overhead of the encoder and decoder components are also considered. The result shows that the total energy of writing and reading using the proposed architecture is reduced by 12.9% on average (up to 26.7%) compared to the memory system which only uses DCW.

It should be noticed that adopting data encoding saves much more energy in the memory system with DCW than the one without DCW for some workloads, such as *streamcluster*. There are two reasons: the first one is that LPS's percentage of original writing data is decreased from 83.8% to 33.4% after DCW scheme, so that data encoding algorithm can map much more writing data from HPS to LPS; the second reason is the ratio of read to write in *streamcluster* is small which is 1.17, thus the reading energy overhead is small. On the other hand, for some other workloads, such as *swim*, adopting data encoding saves less energy in the system with DCW than the one without DCW. Because of the character of the writing data in *swim*, DCW cannot decrease the LPS's percentage in original data too much, just from 53.5% to 52.2%. Moreover, the energy overhead is large which comes from two aspects: the decoder's energy for every reading access and the reading energy overhead in DCW since every write access includes one read access.

These simulation results shows that the proposed encoding architecture is effective to the MLC PCM system without or with adopting the DCW technique.

## VI. Related Work

Recent work has put the focuses on how to reduce the energy overheads of PCM write operations. Data comparison write [17], [21] was proposed to eliminate redundant bit-writes using a read-before-write operation, which can help identify such redundant bits and cancel those redundant write operations to save energy and reduce the impact on performance. Several data inverting schemes [6], [22] were proposed to further reduce the number of bit-writes. To take advantage of the asymmetric RESET and SET energy, Xu *et al.* [23] proposed selective-XOR operations to bias the data value distribution. However, most previous work focused on the SLC PCM without using the special feature of MLC PCM. *Mercury* [12], a fast and energy efficient MLC architecture, is designed to mitigate the MLC overhead by adaptively using

reset-to-set or set-to-reset write schemes. Qureshi *et al.* [8] designed an adaptive PCM management infrastructure to dynamically partition MLC PCM into SLC when the memory capacity is over-provisioned. Similarly, *AdaMS* [3] was designed as an adaptive MLC and SLC partitioning architecture for PCM file storages. Energy-aware data compression [24] was proposed to reduce the programming energy of MLC NAND flash, in which variable-length code reduces the total MLC programming energy, but such a sophisticated is implemented in the software layer.

In this paper, data encoding algorithm is designed to manipulate data in MLC PCM in an energy-efficient way. An energy-efficient data encoding algorithm is presented to manage and encode the input data for reducing the programming energy. In addition, our proposed PCM data encoding scheme can be supplementary to the already used DCW technique so that it can further improve the energy efficiency on the basis of DCW-adopted systems.

## VII. Conclusion

Phase-change memory (PCM) is considered as one of the most promising technologies among emerging non-volatile memories. Besides the common single-level cell (SLC) technology, recent PCM prototypic chips demonstrate that multi-level cell (MLC) is practical. Compared to SLC, MLC can store more than 1 bits on every PCM cell, thus it is one of the attractive properties of PCM that helps to achieve higher storage density. However, programming MLC PCM involves the program-and-verify scheme and incurs much more energy consumption. In this work, we propose a new MLC PCM architecture using data encoding before writing to implement an energy-efficient MLC PCM system based on the observation that there are significant value-dependent energy variations in programming MLC PCM. In addition, we adopt data comparison write (DCW) to enhance the effectiveness of the proposed data encoding architecture. The experimental results show that the total energy consumption of writing and reading using the proposed architecture is reduced by 9.64% on average (up to 19.8%) on plain MLC PCM systems, and by 12.9% on average (up to 26.7%) on DCW-adopted MLC PCM systems.

## References


[1] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner *et al.*, "Phase-change random access memory: a scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4/5, 2008.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proceedings of the International Symposium on Computer Architecture*, 2009, pp. 2–13.

[3] X. Dong and Y. Xie, "AdaMS: Adaptive MLC/SLC phase-change memory design for file storage," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2011, pp. 31–36.

[4] P. Mangalagiri, K. Sarpatwari, A. Yanamandra, V. Narayanan, Y. Xie *et al.*, "A low-power phase change memory based hybrid cache architecture," in *Proceedings of the Great Lakes Symposium on VLSI*, 2008, pp. 395–398.

[5] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proceedings of the International Symposium on Computer Architecture*, 2009, pp. 34–45.

[6] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang *et al.*, "Energy- and endurance-aware design of phase change memory caches," in *Proceedings of the Design, Automation and Test in Europe Conference*, 2010, pp. 136–141.

[7] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the International Symposium on Computer Architecture*, 2009, pp. 24–33.

[8] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montano, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proceedings of the International Symposium on Computer Architecture*, 2010, pp. 153–162.

[9] F. Bedeschi, R. Fackenthal, C. Resta, E. Donze *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, 2009.

[10] T. Nirschl, J. Phipp, T. Happ, G. Burr *et al.*, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Proceedings of the IEEE International Electron Devices Meeting*, 2007, pp. 461–464.

[11] D.-H. Kang, J.-H. Lee, J. Kong, D. Ha *et al.*, "Two-bit cell operation in diode-switch phase change memory cells with 90nm technology," in *Proceedings of the Symposium on VLSI Technology*, 2008, pp. 98–99.

[12] M. Joshi, W. Zhang, and T. Li, "Mercury: A fast and energy-efficient multi-level cell based phase change memory system," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2011, pp. 345–356.

[13] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi *et al.*, "Characterizing flash memory: anomalies, observations, and applications," in *Proceedings of the International Symposium on Microarchitecture*. New York, New York: ACM, 2009, pp. 24–33.

[14] D. Krebs, S. Raoux, C. T. Rettner, G. W. Burr *et al.*, "Characterization of phase change memory materials using phase change bridge devices," *Journal of Applied Physics*, vol. 106, no. 5, p. 054308, 2009.

[15] F. Bedeschi, E. Bonizzoni, G. Casagrande, R. Gastaldi *et al.*, "SET and RESET pulse characterization in BJT-selected phase-change memories," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2005, pp. 1270–1273.

[16] H. Chung, B. H. Jeong, B. Min, Y. Choi, B.-H. Cho *et al.*, "A 58nm 1.8v 1gb pram with 6.4mb/s program bw," in *Proceedings of the International Solid-State Circuits Conference*, 2011, pp. 500–502.

[17] Y. Byung-Do, L. Jae-Eun, K. Jang-Su, C. Junghyun, L. Seung-Yun *et al.*, "A low power phase-change random access memory using a data-comparison write scheme," in *Proceedings of the International Symposium on Circuits and Systems*, 2007, pp. 3014–3017.

[18] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 72–81.

[19] Standard Performance Evaluation Corporation, "SPEC OMP (OpenMP Benchmark Suite)," http://www.spec.org/omp/.

[20] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[21] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of the International Symposium on Computer Architecture*, 2009, pp. 14–23.

[22] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proceedings of the International Symposium on Microarchitecture*, 2009, pp. 347–357.

[23] W. Xu, J. Liu, and T. Zhang, "Data manipulation techniques to reduce phase change memory write energy," in *Proceedings of the International Symposium on Low power Electronics and Design*, 2009, pp. 237–242.

[24] Y. Joo, Y. Cho, D. Shin *et al.*, "Energy-aware data compression for multi-level cell (MLC) flash memory," in *Proceedings of the Design Automation Conference*, 2007, pp. 716–719.