A Partition-based Mechanism for Reducing Energy in Phase Change Memory

Yunjiao Peng, Qingfeng Zhuge, Duo Liu, Jian Li, Edwin H.-M. Sha, Key Laboratory of Dependable Service Computing in Cyber Physical Society, College of Information Science and Engineering, Hunan University, Changsha 410082, China
Email: pengyunjiao@hnu.edu.cn
College of Computer Science, Chongqing University, Chongqing 400044, China
Email: qzhuige@gmail.com, liu_duo93@126.com, leejian@cqu.edu.cn, edwinsha@gmail.com
Department of Computer Science, University of Texas at Dallas, Richardson, Texas 75080, USA

Abstract—Phase Change Memory (PCM) is a promising candidate of universal memory due to its non-volatility, high density and bit-alterability. However, high write energy consumption and limited endurance are two major challenges for its adoption in memory hierarchy. To address these challenges, one of the effective methods is to reduce write activities on PCM. This paper targets at PCM-based secondary storage and presents a partition-based mechanism, named Bipartite Matching Write (BMW), to reduce unnecessary write activities on PCM. Based on this mechanism, we propose two kinds of write algorithms, BMW-Greedy and BMW-KM, respectively. Based on greedy algorithm, BMW-Greedy obtains an approximating optimum result by allocating each input sub-block to an optimal available target sub-block. Based on kuhn-munkras algorithm, BMW-KM obtains an optimum result by finding a perfect matching between input sub-blocks and target sub-blocks. Experimental results show that, the proposed BMW-Greedy algorithm achieves 11.29% average energy saving (up to 22.93%) compared to the Flip-N-Write algorithm; the proposed BMW-KM algorithm achieves 12.80% average energy saving (up to 20.86%) compared to the DCW algorithm and 14.81% average energy saving (up to 22.93%) compared to the Flip-N-Write algorithm.

Index Terms—Phase Change Memory, endurance, write energy reduction.

I. INTRODUCTION

Phase Change Memory (PCM), a type of non-volatile memory, is attracting increasing attention as a promising candidate for next generation memories [1], [2], [3], [4], [5], [6], [7], [8], [9]. In contrast to conventional memory technology, PCM combines many advantages such as non-volatility, high density, scalability [10], [11], [12], byte addressability, large cycling endurance [13], [14], [15] and power-economy [16], [17] in one. However, high write energy consumption is the Achilles heel of PCM. This is mainly due to the intrinsic property that, the state switching of the memory element requires large electric current to go through a heating element to produce PCM heat for a long time. As shown in Table I, compared to read operation, both write latency and write energy consumption are much higher. Moreover, although PCM has much higher endurance than flash memory, it is still quite limited. A PCM cell can only sustain $10^7$ to $10^8$ rewrites [18], [19], [20], [21]. The importance of reducing unnecessary write activities on PCM cannot be overemphasized. Therefore, in this paper, we focus on reducing unnecessary write activities on PCM-based secondary storage.

Some previous work on reducing write activities on PCM-based memory has been proposed. Data Comparison Write (DCW) [22], proposed by Yang et al., directly reduces write activities on PCM by comparing the target data with the input data bit-by-bit and only writes the input data bits which are different from the corresponding target data bits. Cho et al. [23] proposed Flip-N-Write to improve the DCW algorithm by adding an additional flip bit for each PCM word. Each flip bit indicates whether the associated input data word has been flipped or not when the input data are written to PCM target place. On a write, after quick bit-by-bit comparison between the target data and input data, the input data word is either written to PCM directly or flipped before written according to how many bits are different. Xu et al. [7] proposed XOR-masked write which adds XOR pattern cell for each PCM word. Most of these techniques, however, focus on PCM-based memory.

In this paper, we target at PCM-based secondary storage and propose a partition-based write mechanism, called Bipartite Matching Write (BMW), to increase the flexibility of write operation. In the BMW, both the target data block and input data block are partitioned into

<table>
<thead>
<tr>
<th>Table I. Comparison of energy consumption for SET, RESET and READ operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>Set</td>
</tr>
<tr>
<td>Reset</td>
</tr>
</tbody>
</table>
several sub-blocks, such that we can flexibly allocate the input data sub-blocks to target data sub-blocks. Based on this mechanism, we further propose two kinds of write techniques known as **BMW-Greedy** and **BMW-KM**, respectively, to reduce unnecessary write activities on PCM. The main difference between these two techniques is the block allocation policy. **BMW-Greedy** adopts the greedy algorithm to allocate the input data sub-blocks, starting from the first input data sub-block. In **BMW-Greedy**, we always choose an optimal target data sub-block to write from all available target data sub-blocks for each input data sub-block. Different from **BMW-Greedy**, **BMW-KM** uses the kuhn munkras algorithm [24] to obtain a perfect matching with minimum number of bits to update at first. To further reduce write activities on PCM, we also propose a basic algorithm called **Block-Flip** which is applied in both of the above two algorithms. **Block-Flip** adds a flip bit for a block to indicate this block has been flipped or not. Different from **Flip-N-Write**, **Block-Flip** is specially designed for block write mechanism such as BMW.

Experimental results show that, **BMW-Greedy** reduces the total write energy of PCM by an average of 11.29% (up to 18.63%) over the existing DCW algorithm and 13.30% (up to 20.71%) over the existing **Flip-N-Write**. **BMW-KM** reduces the total write energy of PCM by an average of 12.80% (up to 20.86%) over the existing DCW algorithm and 14.81% (up to 22.93%) over the existing **Flip-N-Write**.

In summary, this paper makes the following contributions:

- We present a partition-based mechanism for PCM-based secondary storage to increase the flexibility of write operation. Based on this mechanism, we propose two kinds of write algorithms to reduce unnecessary write activities on PCM.
- We propose a basic algorithm used in the above two algorithms to further reduce unnecessary write activities on PCM. This basic algorithm is specially designed for block write mechanism.
- We demonstrate the effectiveness of our techniques by comparing with a number of representative techniques using a set of reality-based benchmarks.

The remainder of this paper is organized as follows. Section II first introduces the background of PCM and the PCM-based system architecture. Section III reviews related work. Section IV gives the motivation of this work. Section V presents our techniques in detail. Experimental results are presented and discussed in Section VI. Finally, we discuss future work and conclude in Section VII.

## II. BACKGROUND AND SYSTEM ARCHITECTURE

In this section, we first introduce the basic concepts of phase change memory. Then, we show a PCM-based system architecture, where PCM is used as secondary storage.

### A. Phase Change Memory

Phase change memory (PCM) is an attractive non-volatile memory. One common PCM cell is shown in Figure 1(a). The electrical current passes through the phase change material ($Ge_{2}Sb_{2}Te_{5}$ or GST) [25], [21] between the top electrode and heater. PCM utilizes the large resistivity contrast between crystalline (low resistivity) and amorphous (high resistivity) phases of the phase change material [26]. Set and reset state of PCM refers to low and high resistance state, respectively. As shown in Figure 1(b), the GST has low resistance with logical data 0 at SET state, and has high resistance with logical data 1 at RESET state.

![Figure 1. Illustration of phase change memory. (a) Basic structure of conventional phase change memory cell. (b) Typical GST R-I curve. (c) READ, SET, and RESET operations of PCM.](image)

**Figure 1.** Illustration of phase change memory. (a) Basic structure of conventional phase change memory cell. (b) Typical GST R-I curve. (c) READ, SET, and RESET operations of PCM.

The remainder of this paper is organized as follows. Section II first introduces the background of PCM and the PCM-based system architecture. Section III reviews related work. Section IV gives the motivation of this work. Section V presents our techniques in detail. Experimental results are presented and discussed in Section VI. Finally, we discuss future work and conclude in Section VII.

## II. BACKGROUND AND SYSTEM ARCHITECTURE

In this section, we first introduce the basic concepts of phase change memory. Then, we show a PCM-based system architecture, where PCM is used as secondary storage.

### A. Phase Change Memory

Phase change memory (PCM) is an attractive non-volatile memory. One common PCM cell is shown in Figure 1(a). The electrical current passes through the phase change material ($Ge_{2}Sb_{2}Te_{5}$ or GST) [25], [21] between the top electrode and heater. PCM utilizes the large resistivity contrast between crystalline (low resistivity) and amorphous (high resistivity) phases of the phase change material [26]. Set and reset state of PCM refers to low and high resistance state, respectively. As shown in Figure 1(b), the GST has low resistance with logical data 0 at SET state, and has high resistance with logical data 1 at RESET state.

![Figure 2. Illustration of PCM-based system architecture.](image)

**Figure 2.** Illustration of PCM-based system architecture.
The operations of PCM are controlled by electrical current, as shown in Figure 1(c). To set the PCM cell into the crystalline phase, a medium electrical current pulse referred to as SET current pulse is applied to crystallize the phase change material during SET time. In contrast, to make RESET state, the phase change material is melt-quenched by heating it using RESET current pulse which is a shorter current pulse with a higher peak value. The temperature of GST first rises above the melting temperature ($T_{\text{reset, min}}$) which is about 620°C and then cools down quickly [27], [7]. To read the state of the PCM cell, we need sense the resistance of the cell by passing an electrical current low enough not to disturb the current state.

From the Figure 1(c), we can see that SET current has a much longer persistent period than RESET current, and READ current is almost negligible. In fact, the energy consumption produced by SET operation can be several times higher than that of RESET operation [7]. The energy consumption of READ operation is the least. Therefore, replacing a write operation with a read-modify-write operation is an efficient way to reduce the energy consumption on PCM.

### B. System Architecture

A typical PCM-based system architecture is shown in Figure 2. In this system, PCM is used as secondary storage. Currently, some PCM-based storage array has emerged such as Samsung’s 8-Gb array [28].

PCM is used as secondary storage for the following reasons. First, PCM is a non-volatile memory. Next, PCM is superior to other secondary storages for its advantages such as increased speed, better scaling, and better density [29]. Moreover, compared to flash memory, PCM is much faster, as single bits can be changed to either 1 or 0 without first erasing an entire block of cells.

However, PCM has its own characteristics and challenges. Compared to conventional memory technology, higher write energy consumption and limited PCM endurance are the main challenges we faced. Therefore, reducing the unnecessary write activities on PCM is a significant work.

### III. RELATED WORK

PCM has recently attracted a large interest as a new generation of products in memory hierarchy. Vast PCM research has put focus on properties of phase change materials [27], [30], [10], [26], [31], [12] etc. Recently, the research about the feasibility of multi-level cell (MLC) [9], [32], [19], [7], etc, and architecture level optimization for PCM systems [22], [23], [7] has emerged. We find that [22], [23], [7] are the most relevant related work to ours. Therefore, we will give a detailed introduction for them below.

Yang et al. [22] proposed the Data-Comparison Write (DCW) which replaces the write operation with the read-before-write operation. In DCW algorithm, the corresponding target data word will be read at first before writing the input data word on a write. Each bit which starting from the first bit to the last bit in the read buffer is compared with the corresponding input data bit, if the input data bit is different from the corresponding memory bit, then this memory bit is updated. DCW is a simple and effective scheme to reduce unnecessary write activities on PCM without memory overhead.

Cho et al. [23] proposed the Flip-N-Write algorithm to improve the DCW algorithm at the expense of additional storage overhead. In Flip-N-Write algorithm, N is assumed to be the write bandwidth. On a write, the target data word will be quickly compared with the input data word bit-by-bit to get the number of bits which differ between target data word and input data word. If the number of bits to update excels $N/2$, then this input data word will be flipped at first. Thus a flip bit is needed to distinguish whether this input data word has been flipped or not, and this flip bit will be written to PCM with the new input data word at the same time. In the Flip-N-write algorithm, the maximum number of bits to update never exceed $N/2$ on a write, and the memory overhead is $1/N$ per a memory word.

Xu et al. [7] proposed the XOR-masked write algorithm. In the XOR-masked write algorithm, given the input word $d_1, d_2, \cdots, d_n$ to write and target word $d'_1, d'_2, \cdots, d'_n$ stored in PCM, an optimal XOR pattern data $d_p$ will be found to make the overall word write en-
energy be minimal when \( \{d_1 \oplus d_p, d_2 \oplus d_p, \ldots, d_n \oplus d_p\} \)
is written into the memory.

Different from the previous work, our work focuses on reducing unnecessary write activities on PCM-based secondary storage, instead of PCM-based main memory. Moreover, we try to reduce total PCM writes, namely sum of PCM updates and storage overhead, instead of PCM updates or storage overhead. In other words, the problem of this paper is a multi-objective optimization problem [33], [34] or energy-balanced problem [35]. Although both of updates on PCM and overhead in our work are the tradeoff between DCW and Flip-N-Write, total write activities on PCM in our work is the least.

Experimental results show that, the BMW-Greedy algorithm achieves 11.29% average energy saving (up to 18.63%) compared to the DCW algorithm and 13.30% average energy saving (up to 20.71%) compared to the Flip-N-Write algorithm; the BMW-KM algorithm achieves 12.80% average energy saving (up to 20.86%) compared to the DCW algorithm and 14.81% average energy saving (up to 22.93%) compared to the Flip-N-Write algorithm.

IV. MOTIVATIONAL EXAMPLE

In this section, we provide a miniature example to show the problem and motivation of our work.

In this example, we assume that an input data block is written to a target data block stored in PCM. The size of input data block and target data block are both 64 bits, as shown in Figure 3. In traditional write mechanism, the write activities on PCM is 38 due to there are 38 bits differing between the input data block and target data block. This means that, if a bit in input data block is equal to the corresponding bit in target data block, this bit will be skipped. Therefore, the more the bits are skipped, the less the write activities on PCM.

To maximize the number of bits skipped, we change the write mechanism in sacrifice of overhead. In this example, the number of bits skipped is 26 which is less than 64 = 2^6. If we flip all bits in input data block before writing, the number of bits skipped will be 38. The overhead is one flip bit to indicate whether this input data block has been flipped or not. Therefore, the write activities on PCM is 27.

To further reduce unnecessary write activities on PCM, we adjust the relation between input data block and target data block by using modeling and analyzing method [36]. Our problem is modeled as an bipartite matching by partitioning the input data block and target data block into several sub-blocks. Moreover, we determine the partitioning method by analyzing the relationship of memory overhead and PCM updates. After partitioning, each input data sub-block can be written to any one of target data sub-blocks. Therefore, we can apply some energy saving mapping algorithms to assign these input data sub-blocks [37].

In this example, the input data block and target data block are partitioned into 4 sub-blocks, respectively, as shown in Figure 3. Each sub-block consists of 4 PCM words, the size of which is assumed to be 4 bits. Moreover, four target data sub-blocks (denoted by \( y_i, i = 1, 2, 3, 4 \)) are available for four input data sub-blocks (denoted by \( x_i, i = 1, 2, 3, 4 \)). We present all of the above information effectively using a complete bipartite graph \( G = (X, Y, U) \), as shown in Figure 4 (a). Each edge has a nonnegative updates \( U(i, j)(0 < i \leq 4, 0 < j \leq 4) \) to indicate how many bits in PCM are updated when input data sub-block \( Sub_X[i] \) is written to target data sub-block \( Sub_Y[j] \) using Block-Flip algorithm. To be clear,
the bipartite graph is translated to the form of a matrix, as shown in Figure 4(b).

Then we apply the greedy algorithm to assign all input data sub-blocks. Starting from the first input data sub-block, we choose an optimal target data sub-block for each input data sub-block to write. For example, from Figure 4(b), we can see that U[1][4] is the lowest value in the first row, therefore the forth target data sub-block is chosen to write the first input data sub-block, and the number of bits to update is just 2 bits. Then the forth target sub-block is marked to be unavailable. We write the second input data sub-block in the same manner, but only those available target sub-blocks can be chosen. Therefore, the first target sub-block is chosen to write. Similarly, the third input sub-block is written to the second target data sub-block and the fourth input data sub-block is written to the third target data sub-block. The total number of bits to update is 12 bits, and the overhead is 12 bits. Therefore, the total number of bits to write is 24 bits. Except for the greedy algorithm, we can also choose other algorithms to assign all input data sub-blocks according to the demand of performance. An optimal algorithm will be introduced in V-C.

V. ALGORITHMS BASED ON BIPARTITE MATCHING WRITE (BMW) MECHANISM

In this section, we present our algorithms in detail. First, we explain the correlation of our algorithms, illustrated in Figure 5. Our object is to reduce the write energy consumption in PCM-based secondary storage system, and one of the most effective ways is to reduce the unnecessary write activities on PCM. As previously mentioned, if a input data bit is the same with the corresponding target data bit, then we skip this input data bit. Therefore, our aim is to skip the input data bits as many as possible in some ways. To increase the flexibility of write operation, we present a partition-based mechanism called Bipartite Matching Write (BMW). In BMW, we partition the input data block and the target data block into several sub-blocks in the same manner and each input data sub-block can be written to any one of target data sub-blocks. On this basis, there are two algorithms called BMW-Greedy and BMW-KM respectively are proposed. Their only difference is the way to assign all input data sub-blocks. The BMW-Greedy algorithm uses a greedy fashion to assign all input data sub-blocks, and the BMW-KM algorithm uses the kuhn-munkras algorithm to find an optimum matching at first, then assign these input data sub-blocks according to the optimum matching. Moreover, we propose a basic algorithm called Block-Flip which is applied in both of the above two algorithms. Therefore, we present the Block-Flip algorithm at first. Then the BMW-Greedy algorithm and BMW-KM algorithm are presented in Section V-B and Section V-C, respectively.

A. Block-Flip algorithm

In this subsection, we will present a basic algorithm called Block-Flip. The Block-Flip algorithm is a compromise of the DCW algorithm and Flip-N-Write algorithm by adding a flip bit for each input data block, and will be used in the following two algorithms. In the Block-Flip algorithm, as shown in Algorithm 1, the number of bits to update never excels half of the number of bits in a block by adding a flip bit for each block. The input of the Block-Flip algorithm is input data block X[] and target data block Y[]. The output is the number of bits to update update_time and extra overhead Overhead. The first step is to compare the input data block and the target data block bit by bit quickly to determine how many bits differ. The second step is to flip the input data block or write it directly according to the result of the first step. If the number of bits differing between the input data block and target data block excels half of the number of bits in a block, the input data block will be flipped before written, and set flip bit to be 1. Otherwise, the input data block will be written to PCM directly and set flip bit to be 0. In Block-Flip, although the updates on PCM is more than Flip-N-Write, the overhead is much less. Different from Flip-N-Write, Block-Flip is specially designed for block write algorithm.

Algorithm 1 Block-Flip

Input: The input data block, X[]; The target data block, Y[]

Output: The number of bits to update, update_time; The extra overhead, Overhead;
1: \text{update\_time} \leftarrow 0.
2: \text{Overhead} \leftarrow \text{length}[X].
3: \text{for } j = 0 \text{ to } \text{length}[X] \text{ do}
4: \text{if } X[i] \neq Y[i] \text{ then}
5: \text{update\_time} + + .
6: \text{end if}
7: \text{end for}
8: \text{if } \text{update\_time} > \text{length}[Y]/2 \text{ then}
9: \text{Flip} \leftarrow 1.
10: \text{update\_time} \leftarrow \text{length}[Y] - \text{update\_time}.
11: \text{Flip the input data block, and then write the new input data block and flip bit into PCM.}
12: \text{else}
13: \text{Flip} \leftarrow 0.
14: \text{Write the input data block into PCM.}
15: \text{end if}

B. BMW-Greedy Algorithm

In this subsection, we will present an enhanced algorithm called BMW-Greedy, as shown in Algorithm 2. The input is input data block X[] and target data block Y[]. The output is the number of bits to update (update_time) and extra overhead (Overhead). In Algorithm 2, the first step is to partition the input data block and the target data block into several sub-blocks the size of which is predetermined. Let array Sub_X[][] each input data sub-blocks, and Sub_Y[][] record all target data sub-blocks. The second step is to write these input data sub-blocks using a greedy fashion. Starting from the first element of array Sub_X[], each input data sub-block is compared with all available target data sub-blocks recorded in array Sub_Y[][] bit by bit to find an optimal target data sub-block to write, then we mark this target data sub-block to be unavailable and write this input data sub-
block into the optimal target data sub-block using the Block-Flip algorithm. In this algorithm, the overhead is $(\log_2 \text{length}[\text{Sub}_X] + 1) \times \text{length}[\text{Sub}_X]$ since we need $\log_2$ bits to record where each input data sub-block is written and 1 bit to indicate whether this input data block has been flipped or not. BMW-Greedy is easy to implement and obtains a suboptimal result.

Algorithm 2 BMW-Greedy

Input: The input data block, $X[]$; The target data block, $Y[]$.
Output: The number of bits to update, $\text{update\_time}$; The extra overhead, $\text{Overhead}$.
1: $\text{update\_time} \leftarrow 0$.
2: $\text{Sub}_X[] \leftarrow$ Partition the input data block.
3: $\text{Sub}_Y[] \leftarrow$ Partition the stored data block.
4: $\text{Overhead} \leftarrow (\log_2 \text{length}[\text{Sub}_X] + 1) \times \text{length}[\text{Sub}_X]$.
5: for $i = 0$ to $\text{length}[\text{Sub}_X]$ do
6: Choose an optimal target data sub-block $\text{Sub}_Y[j]$ from all available target data sub-blocks.
7: Mark the chosen target block $\text{Sub}_Y[j]$ as unavailable.
8: $\text{update\_time} \leftarrow \text{update\_time} + \text{Block} – \text{Flip}(\text{Sub}_X[i], \text{Sub}_Y[j]).$
9: end for

C. BMW-KM Algorithm

In this subsection, we will present another enhanced algorithm called BMW-KM, as shown in Algorithm 3. The input is input data block $X[]$ and target data block $Y[]$. The output is the number of bits to update $\text{update\_time}$ and extra overhead $\text{Overhead}$. Same with Algorithm 2, the input data block and target data block are first partitioned into several sub-blocks which are recorded in array $\text{Sub}_X[]$ and array $\text{Sub}_Y[]$, respectively. Then the assignment problem of input data sub-blocks can be formulated using a complete bipartite graph $G = (X, Y, U)$ with $n$ input data sub-block vertices ($X$) and $n$ target data sub-block vertices ($Y$), and each edge has a non-negative update $U(i, j)(0 < i < \text{length}[\text{Sub}_X], 0 < j < \text{length}[\text{Sub}_Y])$. Finally, we use the kuhn-munkras algorithm to find a perfect matching with minimum updates. The matrix interpretation of the kuhn-munkras algorithm is as follows.

1. The first step is to perform row operations on the matrix. To do this, the lowest of all $U(i, j)(0 \leq j < \text{length}[\text{Sub}_X])$ is taken, and is subtracted from each element in the first row. This will lead to at least one zero in that row. This procedure is repeated for all rows. Then a matrix $B[][]$ with at least one zero per row is obtained. Now we try to assign input data sub-blocks to target data sub-blocks such that each input data sub-block is written to only one target data sub-block. This is illustrated in the matrix shown in Figure 6(a). The zeros shown in boldface are the assigned tasks.

Sometimes it may turn out that the matrix at this stage cannot be used for assigning, as is the case shown in Figure 6(b). In this case, no complete matching can be obtained. Note that both input data sub-block $X_0$ and $X_2$ are written to target data sub-block $Y_0$. It’s obvious that both can’t be written to the same target data sub-block. Also note that no input data sub-block is written to target data sub-block $Y_2$. The following procedure is a way to overcome this.

2. The second step is to repeat the above procedure for all columns. The minimum element in each column is subtracted from all elements in that column and then check if an assignment is possible. In most situations this will give the result, but if it is still not possible to assign, then proceed with the steps below.

3. The third step is to cover all zeros in the matrix by marking as few rows and columns as possible. We can draw lines through all marked columns and marked rows, as shown in Figure 6(c). From the elements uncovered, find the lowest value. Subtract this from every unmarked element and add it to every element covered by two lines.

Repeat step (3) until an assignment is possible; this is when the minimum number of lines used to cover all the zeros is equal to the number of target data sub-blocks.

Algorithm 3 BMW-KM

Input: The input data block, $X[]$; The target data block, $Y[]$.
Output: The number of bits to update, $\text{update\_time}$; The extra overhead, $\text{Overhead}$.
1: $\text{update\_time} \leftarrow 0$.
2: $\text{Sub}_X[] \leftarrow$ Partition the input data block.
3: $\text{Sub}_Y[] \leftarrow$ Partition the target data block.
4: $\text{Overhead} \leftarrow (\log_2 \text{length}[\text{Sub}_X] + 1) \times \text{length}[\text{Sub}_X]$.
5: Create the bipartite graph and it’s corresponding matrix.
6: Perform row operation and check if an assignment is possible.
7: if no assignment can be made then
8: Perform column operation and check if an assignment is possible.
9: if no assignment can be made then
10: repeat
11: \( r \leftarrow \) the minimum number of lines used to cover all the 0.
12: \( \delta \leftarrow \) find the lowest value from all elements uncovered.
13: Subtract $\delta$ from every uncovered element.
14: Add $\delta$ to every element covered by two lines.
15: until \( r = \text{length}[\text{Sub}_Y] \)
16: end if
17: end if
18: Perfecit matching with minimum updates $\leftarrow$ find length[Sub_X] zeros which exist in different row and column.
19: for $i = 0$ to $\text{length}[\text{Sub}_X]$ do
20: $\text{update\_time} \leftarrow \text{update\_time} + \text{Block} – \text{Flip}(\text{Sub}_X[i], \text{Sub}_Y[j])$ where $\text{Sub}_X[i]$ and $\text{Sub}_Y[j]$ are matched.
21: end for

The BMW-KM algorithm is applied in previous example. The detailed calculating process of this algorithm is as follows.

1. Performing row operations on the matrix. As shown in Figure 7(a), the lowest value of the first row is 2, and the first row turn into (4, 2, 3, 0) after this row operation. This procedure is repeated for all rows. Finally, we have a matrix with at least one zero per row, as shown in Figure 7(b).

2. Checking if a complete matching is possible. In the above matrix, the zeros represent assignments. However, no complete matching can be made. Note that both the first input data sub-block and the third input data sub-block are written to the forth target data sub-block. Also note that no input data sub-block is written to the second
is superior to BMW-Greedy solution. However, in terms of complexity of assignment, BMW-Greedy is simpler to implement. The complexity of BMW-Greedy and BMW-KM is $O(n^2)$ and $O(n^3)$, respectively. Therefore, BMW-Greedy should be used when write speed is as important as PCM lifetime; otherwise, BMW-KM should be used to prolong the lifetime of PCM to an extreme at the sacrifice of write speed.

VI. EVALUATION

A. Experiment Setup

We have evaluated our algorithms based on a simulation using realistic workloads. In this section, we will introduce our experiment setup, including PCM configuration, benchmark and system configurations.

In our experiment, we assume the size of a block is 4096 bytes and the write bandwidth is 2 bytes, then employ four simulation programs: DCW, Flip-N-Write, BMW-Greedy and BMW-KM. These programs are used to evaluate the conventional write algorithms and our algorithms.

Experiments are conducted on a set of benchmarks, including “Flicker, Aria, Gary B, 7-BGM, 2-VSV, HDCV, PA, Fly”, as shown in Table III. “Flicker” are a set of photos (jpeg format) taken from Flicker which is a photo sharing Website. We selected fourteen latest photos which is available for downloading from “The comments” board. These photo have different pixels ranging from 850 × 850 to 4000 × 3000. “Aria and Gary B” are both albums (x-ape, x-wavpack, flac and x-mpegurl format) from Cafe del Mar. “7-BGM” are seven background music (mpg format) from movie. “2-VSV” are variety shows (mp4 format) taken from a Forum using IPV6. “HDCV” is a concert video (vnd.rn-realmedia format). “PA” is a photo.
### TABLE II.
**COMPARISON OF RESULTS IN THE MOTIVATIONAL EXAMPLE FOR DCW, Flip-N-Write, BMW-Greedy AND BMW-KM.**

<table>
<thead>
<tr>
<th></th>
<th>DCW</th>
<th>Flip-N-Write</th>
<th>Block-Flip</th>
<th>BMW-Greedy</th>
<th>BMW-KM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bits to update</td>
<td>38</td>
<td>20</td>
<td>26</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Overhead</td>
<td>0</td>
<td>16</td>
<td>1</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Total number of bits to write</td>
<td>38</td>
<td>36</td>
<td>27</td>
<td>24</td>
<td>20</td>
</tr>
</tbody>
</table>

### TABLE III.
**EXPERIMENT TEST CASES**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Size</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flicker</td>
<td>Photos downloaded from flicker</td>
<td>251592704</td>
<td>7678</td>
</tr>
<tr>
<td>Aria</td>
<td>A album belonging to Cafe del Mar</td>
<td>11975884800</td>
<td>365475</td>
</tr>
<tr>
<td>Gary B</td>
<td>A album belonging to Cafe del Mar</td>
<td>5501353984</td>
<td>167888</td>
</tr>
<tr>
<td>7-BGM</td>
<td>Background Music</td>
<td>259719168</td>
<td>7926</td>
</tr>
<tr>
<td>2-VSV</td>
<td>Variety show videos</td>
<td>4253253632</td>
<td>129799</td>
</tr>
<tr>
<td>HDCV</td>
<td>A high definition concert video</td>
<td>14447920</td>
<td>440</td>
</tr>
<tr>
<td>PA</td>
<td>A photo album</td>
<td>66453504</td>
<td>2028</td>
</tr>
<tr>
<td>Flv</td>
<td>Flash videos</td>
<td>708345856</td>
<td>21617</td>
</tr>
</tbody>
</table>

album of my candid photos (jpeg format). “Flv” are flash videos (x-flv format) taken from YouKu.

The experiment is done on a DELL(R) Dimension 3010DT workstation with an Intel(R) Core(TM) i3-3200 CPU @ 3.30GHz processor, a 4.00GB main memory, and a 500GB 7,200 RPM Seagate(R) hard disk drive. We use Ubuntu 12.10 with Ext4 file system.

### B. Evaluation Results

Before comparing our algorithms with DCW and Flip-N-Write, we have obtained the optimal partitioning method for the BMW-Greedy algorithm and BMW-KM algorithm. We have experimented on them with six partitioning method including partitioning a block into 8, 16, 32, 64, 128 and 256 sub-blocks, respectively.

The experimental results for BMW-Greedy and BMW-KM with six partitioning methods are shown in Figure 8(a) and 8(b), respectively. The x-coordinate represents six partitioning methods. For example, “8” represents partitioning the block into 8 sub-blocks. The y-coordinate represents the ratio of bits to write which includes the PCM updates and storage overhead. From this figure, we can see that, from “8” to “16” and “16” to “32”, the results of most cases change less. But from “32” to “64” and from “64” to “128”, the results of most cases decrease significantly. However, this trend doesn’t keep up. From “128” to “256”, the results of most cases begin to increase. That is to say, the overall performance of our algorithms maximizes when the number of sub-blocks is 128. The reason for this trend is that when the number of sub-blocks increases, the number of bits to update decreases; however, the storage overhead to record where each sub-block has written increases.

The experimental results for DCW, Flip-N-Write, BMW-Greedy and BMW-KM are shown in Table IV. In this table, we compare the number of bits to update, storage overhead and the total number of bits to write. Columns starting from the third column to the tenth column represent different benchmarks respectively. Column “Average” shows the average result of all benchmarks. Row “Bits to update” shows the number of bits to update without considering storage overhead, and Row “Overhead” presents how many extra bits need to be written to PCM with the input data. Row “Total writes” shows the total number of bits to write. Row “Ratio” shows the percentage of bits written to the size of input data. From this table, we can see that the overhead of DCW is 0 but the number of bits to update is the largest. The Flip-N-Write algorithm reduces the number of bits to update effectively but the overhead is too much, thus the total number of bits to write in PCM is even larger then DCW. Our algorithms are based on the partitioning strategy, and their difference is how to allocate these sub-blocks effectively. Since that the number of bits to update...
all sub-blocks are allocated using the greedy algorithm, starting from the first sub-block, BMW-Greedy

We have obtained the optimal number of blocks which, the total PCM writes reaches its minimum value. In PCM decreases with the increasing of the number of blocks, but overhead increases at the same time, in PCM decreases with the increasing of the number of blocks, but overhead increases at the same time, therefore there exists a critical number of sub-blocks, at which, the total PCM writes reaches its minimum value. We have obtained the optimal number of blocks which is 128 by experimentation as mentioned before. In the BMW-Greedy algorithm, starting from the first sub-block, all sub-blocks are allocated using the greedy algorithm, where the current choice is optimal. However, the final result is just near optimal in most case. In the BMW-KM algorithm, we use the kuhn-munkras algorithm to find a perfect matching at first, so that the final result is optimal. From Column“Average”, we can see that the average ratio of BMW-KM is minimum. BMW-Greedy reduces the total write energy consumption of PCM by an average of 11.29% (up to 18.63%) over the existing DCW algorithm and 13.30% (up to 20.71%) over the existing Flip-N-Write. BMW-KM reduces the total write energy of PCM by an average of 12.80% (up to 20.86%) over the existing DCW algorithm and 14.81% (up to 22.93%) over

TABLE IV.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Performance parameter</th>
<th>Flicker</th>
<th>Aria</th>
<th>Gary B</th>
<th>7-BGM</th>
<th>2-VSV</th>
<th>HDCV</th>
<th>PA</th>
<th>Flv</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCW</td>
<td>Bits to update</td>
<td>110342326</td>
<td>523275648</td>
<td>240146849</td>
<td>113285008</td>
<td>1806561313</td>
<td>63121000</td>
<td>29099002</td>
<td>309115657</td>
<td>1257979378</td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>113285008</td>
<td>1806561313</td>
<td>63121000</td>
<td>29099002</td>
<td>309115657</td>
<td>1257979378</td>
</tr>
<tr>
<td></td>
<td>Total writes</td>
<td>110342326</td>
<td>523275648</td>
<td>240146849</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>43.86%</td>
<td>43.70%</td>
<td>43.66%</td>
<td>43.62%</td>
<td>43.74%</td>
<td>43.78%</td>
<td>43.79%</td>
<td>43.64%</td>
<td>45.72%</td>
</tr>
<tr>
<td>Flip-N-Write</td>
<td>Bits to update</td>
<td>115269605</td>
<td>5478234077</td>
<td>2515745685</td>
<td>118255999</td>
<td>1679387687</td>
<td>5702848</td>
<td>26281288</td>
<td>279197261</td>
<td>1130723754</td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>115269605</td>
<td>5478234077</td>
<td>2515745685</td>
<td>118255999</td>
<td>1679387687</td>
<td>5702848</td>
<td>26281288</td>
<td>279197261</td>
<td>1130723754</td>
</tr>
<tr>
<td></td>
<td>Total writes</td>
<td>115269605</td>
<td>5478234077</td>
<td>2515745685</td>
<td>118255999</td>
<td>1679387687</td>
<td>5702848</td>
<td>26281288</td>
<td>279197261</td>
<td>1130723754</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>45.82%</td>
<td>45.74%</td>
<td>45.73%</td>
<td>45.53%</td>
<td>45.73%</td>
<td>45.80%</td>
<td>45.80%</td>
<td>45.67%</td>
<td>45.73%</td>
</tr>
<tr>
<td>BMW-Greedy</td>
<td>Bits to update</td>
<td>76346877</td>
<td>3715710944</td>
<td>204757283</td>
<td>80844906</td>
<td>1131308202</td>
<td>4462087</td>
<td>20793249</td>
<td>220628427</td>
<td>8071660504</td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>76346877</td>
<td>3715710944</td>
<td>204757283</td>
<td>80844906</td>
<td>1131308202</td>
<td>4462087</td>
<td>20793249</td>
<td>220628427</td>
<td>8071660504</td>
</tr>
<tr>
<td></td>
<td>Total writes</td>
<td>76346877</td>
<td>3715710944</td>
<td>204757283</td>
<td>80844906</td>
<td>1131308202</td>
<td>4462087</td>
<td>20793249</td>
<td>220628427</td>
<td>8071660504</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>33.47%</td>
<td>34.15%</td>
<td>25.02%</td>
<td>34.25%</td>
<td>29.78%</td>
<td>34.07%</td>
<td>34.41%</td>
<td>34.27%</td>
<td>32.43%</td>
</tr>
<tr>
<td>BMW-KM</td>
<td>Bits to update</td>
<td>7862272</td>
<td>374246400</td>
<td>171917312</td>
<td>8116224</td>
<td>132914176</td>
<td>450560</td>
<td>2076672</td>
<td>22135808</td>
<td>89964928</td>
</tr>
<tr>
<td></td>
<td>Overhead</td>
<td>7862272</td>
<td>374246400</td>
<td>171917312</td>
<td>8116224</td>
<td>132914176</td>
<td>450560</td>
<td>2076672</td>
<td>22135808</td>
<td>89964928</td>
</tr>
<tr>
<td></td>
<td>Total writes</td>
<td>7862272</td>
<td>374246400</td>
<td>171917312</td>
<td>8116224</td>
<td>132914176</td>
<td>450560</td>
<td>2076672</td>
<td>22135808</td>
<td>89964928</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>32.89%</td>
<td>31.89%</td>
<td>22.80%</td>
<td>33.69%</td>
<td>27.07%</td>
<td>33.13%</td>
<td>33.49%</td>
<td>32.41%</td>
<td>30.92%</td>
</tr>
<tr>
<td>Improvement</td>
<td>BMW-Greedy vs DCW</td>
<td>10.39%</td>
<td>9.55%</td>
<td>18.63%</td>
<td>9.37%</td>
<td>13.96%</td>
<td>9.71%</td>
<td>9.37%</td>
<td>9.37%</td>
<td>11.29%</td>
</tr>
<tr>
<td></td>
<td>BMW-Greedy vs Flip-N-Write</td>
<td>12.35%</td>
<td>11.59%</td>
<td>20.71%</td>
<td>11.28%</td>
<td>15.95%</td>
<td>11.73%</td>
<td>11.38%</td>
<td>11.38%</td>
<td>13.30%</td>
</tr>
<tr>
<td></td>
<td>BMW-KM vs DCW</td>
<td>10.97%</td>
<td>11.80%</td>
<td>20.86%</td>
<td>9.93%</td>
<td>16.67%</td>
<td>10.65%</td>
<td>10.30%</td>
<td>11.23%</td>
<td>12.80%</td>
</tr>
<tr>
<td></td>
<td>BMW-KM vs Flip-N-Write</td>
<td>12.93%</td>
<td>13.85%</td>
<td>22.93%</td>
<td>11.85%</td>
<td>18.66%</td>
<td>12.68%</td>
<td>12.31%</td>
<td>13.26%</td>
<td>14.81%</td>
</tr>
</tbody>
</table>

Figure 8. Comparison of experimental results for BMW algorithms with six partitioning method including partitioning a block into 8, 16, 32, 64, 128 and 256 sub-blocks, respectively.
the existing Flip-N-Write.

Figure 9 gives us a visual comparison of the total number of bits that is written among these four algorithms. Where, the x-coordinate represents different benchmarks and the y-coordinate represents the ratio of bits to write which includes the PCM updates and overhead. Different from Figure 8, where the comparison is among the BMW algorithms with different partitioning methods, the comparison in this figure is among DCW, Flip-N-Write, BMW-Greedy and BMW-KM algorithms.

VII. CONCLUSION AND FUTURE WORK

In this paper, we target at PCM-based secondary storage and proposed a partition-based mechanism called Bipartite Matching Write (BMW) to increase the flexibility of write operation. Based on this mechanism, we further proposed two kinds of write algorithms called BMW-Greedy and BMW-KM, respectively, to reduce unnecessary write activities on PCM. Moreover, we proposed a basic algorithm named as Block-Flip which is designed specially for block write mechanism such as BMW. The Block-Flip algorithm which improves the DCW algorithm by adding a flip bit for each block is applied in BMW-Greedy and BMW-KM. The BMW-Greedy algorithm and BMW-KM algorithm base on BMW where blocks are partitioned into several sub-blocks. What their difference is the allocation strategy. The BMW-Greedy algorithm uses the greedy algorithm to choose a target sub-block which is optimal currently to write. The BMW-KM algorithm uses the kuhn-munkras algorithm to find an optimal matching at first, so that to the final result is optimal. Although the BMW-Greedy algorithm cannot achieve an optimal result at last, but it can obtain a near-optimal result and the computation overhead is less to the BMW-KM algorithm.

Experimental results show that, the BMW-Greedy algorithm and BMW-KM algorithm perform best when the block is partitioned to 128 sub-blocks. Compared to DCW, BMW-Greedy achieves 11.29% average energy saving (up to 18.63%) and BMW-KM achieves 12.80% average energy saving (up to 20.86%). Compared to Flip-N-Write, BMW-Greedy achieves 13.30% average energy saving (up to 20.71%), and BMW-KM achieves 14.81% average energy saving (up to 22.93%).

Currently this work only targets on prolonging the lifetime of PCM-based secondary storage through reducing write activities at architecture level. In fact, PCM is one of the most promising DRAM alternatives with commercial products available in the market, it is essential to prolong the lifetime of PCM-based main memory. Therefore, how to apply our techniques in PCM-based main memory is our future work.

ACKNOWLEDGMENT

This work is partially supported by National 863 Program 2013AA013202, Chongqing csc2012ggC40005, NSFC 61173014, NSF CNS-1015802, Texas NHARP 009741-0020-2009.

REFERENCES


papers in refereed conferences and journals. He has served as an editor for many journals, and as program committee and Chairs for numerous international conferences. He received Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar Award, Chang Jiang Honorary Chair Professorship and China Thousand-Talent Program.