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

Yunjiao Peng[a], Qingfeng Zhuge*[b,c], Duo Liu[b,c], Jian Li[b], Edwin H.-M. Sha[b,c,d]
[a] College of Information Science and Engineering, Hunan University, Changsha 410082, China
Email: pengyunjiao@hnu.edu.cn
[b] College of Computer Science, Chongqing University, Chongqing 400044, China
Email: qfzhuge@gmail.com , {liuduo, leejian}@cqu.edu.cn, edwinsha@gmail.com
[c] Key Laboratory of Dependable Service Computing in Cyber Physical Society,
Ministry of Education, Chongqing, 400044 China
[d] 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 called *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 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 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

TABLE I.
COMPARISON OF ENERGY CONSUMPTION FOR SET, RESET AND READ OPERATION.

	Time(ns)	Current(mA)	Voltage(V)	Energy/bit(pJ)
Read	10	2	0.2	4
Set	1000	4.5	0.6	2700
Reset	50	16	1.2	960

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 $10^7 \sim 10^9$ 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

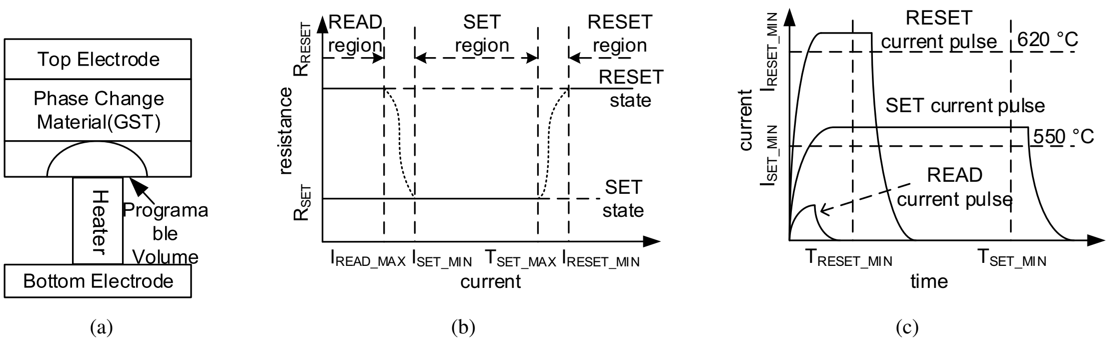


Figure 1. Illustration of phase change memory. (a) Basic structure of conventional phase change memory cell. (b) Typical GST R-I curve. (c) READ, SET, and RESET operations of PCM.

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 ($Ge_2Sb_2Te_5$ 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.

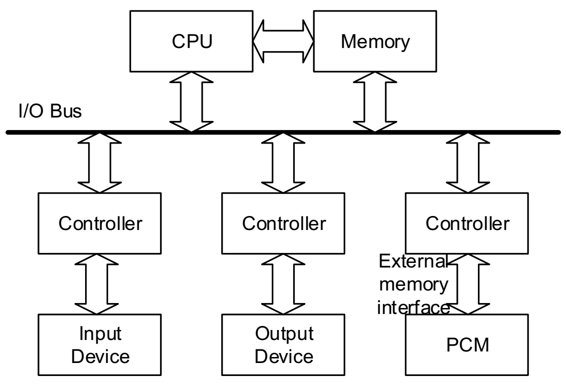


Figure 2. Illustration of PCM-based system architecture.

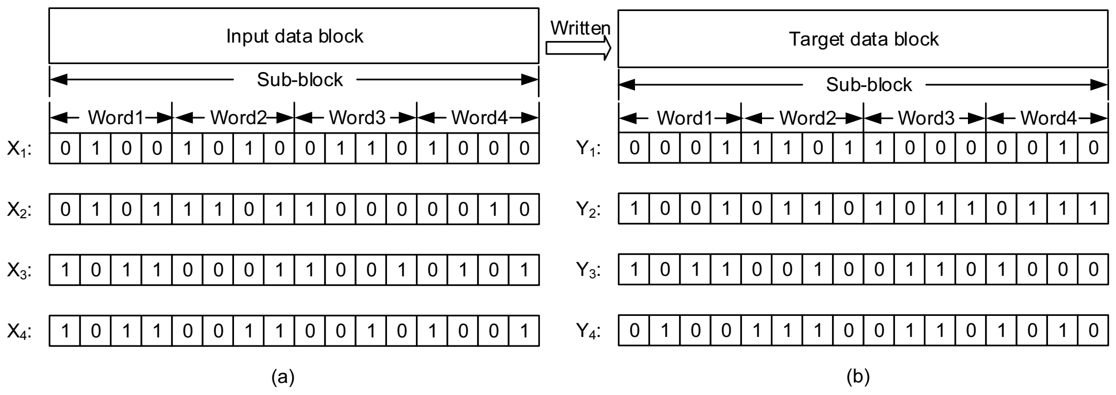

Input data block
Written
Target data block
Sub-block
Word1 Word2 Word3 Word4
X1: 0 1 0 0 1 0 1 0 0 1 1 0 1 0 0 0
X2: 0 1 0 1 1 1 0 1 1 0 0 0 0 0 1 0
X3: 1 0 1 1 0 0 0 1 1 0 0 1 0 1 0 1
X4: 1 0 1 1 0 0 1 1 0 0 1 0 1 0 0 1
Y1: 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0
Y2: 1 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1
Y3: 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0
Y4: 0 1 0 0 1 1 1 0 0 1 1 0 1 0 1 0
(a)
(b)

Figure 3. Input data block and target data block in motivational example. (a) Input data block partitioned into 4 sub-blocks. (b) Target data block partitioned into 4 sub-blocks.

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 (I_{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 needful 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 excel $N/2$ 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 $\{\hat{d_1}, \hat{d_2}, \cdots, \hat{d_n}\}$ stored in PCM, an optimal XOR pattern data d_p will be found to make the overall word write en-

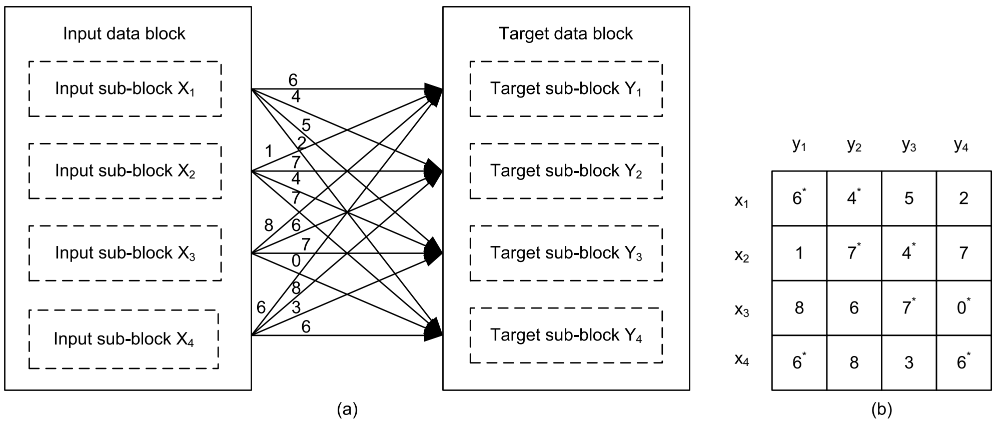


Figure 4. Illustration of Bipartite graph model. (a) Bipartite graph to formulate the write problem. (b) The matrix form of the bipartite graph. The elements with $*$ represent that there is a flip operation before write operation.

ergy be minimal when $\{d_1 \bigoplus d_p, d_2 \bigoplus d_p, \cdots, d_n \bigoplus d_p\}$ 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 $64/2$. 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,

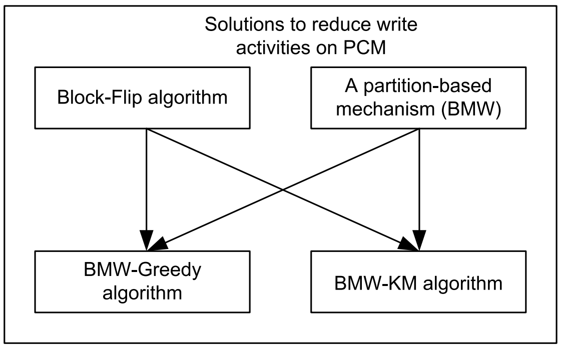

Solutions to reduce write activities on PCM
Block-Flip algorithm
A partition-based mechanism (BMW)
BMW-Greedy algorithm
BMW-KM algorithm

Figure 5. Illustration of the proposed algorithms.

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 way 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 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 *BWM-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: $update\_time \leftarrow 0$.
 2: $Overhead \leftarrow length[X]$.
 3: for $j = 0$ to $length[X]$ do
 4:     if $X[i] \neq Y[i]$ then
 5:         $update\_time++$.
 6:     end if
 7: end for
 8: if $update\_time > length[Y]/2$ then
 9:     $Flip \leftarrow 1$.
10:     $update\_time \leftarrow length[Y] - update\_time$.
11:     Flip the input data block, and then write the new input data block
        and flip bit into PCM.
12: else
13:     $Flip \leftarrow 0$.
14:     Write the input data block into PCM.
15: 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 $Sub\_X[]$ record all 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 $(log_2^{length[Sub\_X]}+1)*length[Sub\_X]$ since that we need $log_2^{length[Sub\_X]}$ 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, $update\_time$;
        The extra overhead, $Overhead$;
1: $update\_time \leftarrow 0$.
2: $Sub\_X[] \leftarrow$Partition the input data block.
3: $Sub\_Y[] \leftarrow$Partition the stored data block.
4: $Overhead \leftarrow (log_2^{length[Sub\_X]} + 1) * length[Sub\_X]$.
5: for $i = 0$ to $length[Sub\_X]$ do
6:     Choose an optimal target data sub-block $Sub\_Y[j]$ from all
       available target data sub-blocks.
7:     Mark the chosen target block $Sub\_Y[j]$ as unavailable.
8:     $update\_time \leftarrow$
       $update\_time + Block - Flip(Sub\_X[i], Sub\_Y[j])$.
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 $update\_time$ and extra overhead $Overhead$. Same with Algorithm 2, the input data block and target data block are first partitioned into several sub-blocks which are recorded in array $Sub\_X[]$ and array $Sub\_Y[]$, 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 nonnegative updates $U(i, j)(0 < i < length[Sub\_X]$, $0 < j < length[Sub\_Y])$. 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 $U(1, j)(0 \leq j < length[Sub\_Y])$ 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, $update\_time$;
        The extra overhead, $Overhead$;
1: $update\_time \leftarrow 0$.
2: $Sub\_X[] \leftarrow$Partition the input data block.
3: $Sub\_Y[] \leftarrow$Partition the target data block.
4: $Overhead \leftarrow (log_2^{length[Sub\_X]} + 1) * length[Sub\_X]$.
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 $r = length[Sub\_Y]$
16:    end if
17: end if
18: Perfect matching with minimum updates $\leftarrow$ find $length[Sub\_X]$
    zeros which exist in different row and column.
19: for $i = 0$ to $length[Sub\_X]$ do
20:     $update\_time \leftarrow update\_time+$
        $Block - Flip(Sub\_X[i], Sub\_Y[j])$ where $Sub\_X[i]$ and
        $Sub\_Y[j]$ 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

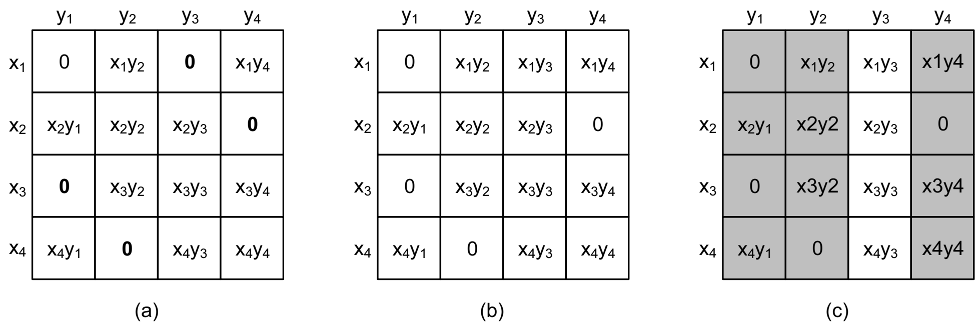


Figure 6. The matrix interpretation of the *kuhn- munkras* algorithm. (a) The matrix which can be used for assigning. (b) The matrix which cannot be used for assigning. (c) Draw the minimum number of lines to cover all zeros.

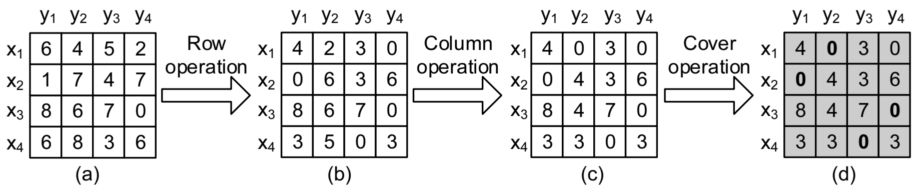


Figure 7. Computation steps of the *kuhn munkras* algorithm. (a) The original matrix created according to the bipartite graph. (b) The new matrix after row operations. (c) The new matrix after column operation. (d) Draw the minimum number of lines to cover all zeros, and the number of lines at this step is equal to the number of target data sub-blocks. The zeros shown in boldface are assignments.

target data sub-block. To overcome this, we repeat the above procedure for all columns.

(3) Performing column operations. The minimum element in each column is subtracted from all the elements in that column. After column operations, we have a new matrix as shown in Figure 7(c).

(4) Checking if a complete matching is possible. We can cover all zeros in the matrix by marking as few rows and columns as possible, as shown in Figure 7(d), draw lines through all marked columns and marked rows. If the minimum number of lines is equal to the number of target data sub-blocks, a complete matching is possible. From Figure 7(d), we can see that the minimum number of lines is 4, therefore, a complete matching is possible. The zeros shown in boldface are assignments which is $\{x_1y_2, x_2y_1, x_3y_4, x_4y_3\}$. Therefore, the total number of bits to write is 20 due to the number of bits to update is 8 and overhead is 12.

Table II summarizes the results in the example. Row "number of bits to update" shows the number of bit to update without considering overhead. Row "Overhead" shows additional storage overhead. Row "Total number of bits to write" shows the total number of bits to write, which is the sum of the number of bits to update and overhead.

D. Performance Analysis

The above two algorithms have their own advantages. In terms of result, *BMW-KM* is superior to *BMW-Greedy* since that *BMW-KM* generates global optimization solution but *BMW-Greedy* only generates approximate optimal solution. However, in terms of complexity of assignment, *BMW-Greedy* is superior to *BMW-KM* and 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, Flv", 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 (mpeg 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*.

	DCW	Flip-N-Write	Block-Flip	BMW-Greedy	BMW-KM
Number of bits to update	38	20	26	12	8
Overhead	0	16	1	12	12
Total number of bits to write	38	36	27	24	20

TABLE III.
EXPERIMENT TEST CASES

Benchmark	Description	Size(bits)	Blocks
Flicker	Photos downloaded from flicker	251592704	7678
Aria	A album belonging to Cafe del Mar	11975884800	365475
Gary B	A album belonging to Cafe del Mar	5501353984	167888
7-BGM	Background Music	259719168	7926
2-VSV	Variety show videos	4253253632	129799
HDCV	A high definition concert video	14417920	440
PA	A photo album	66453504	2028
Flv	Flash videos	708345856	21617

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,

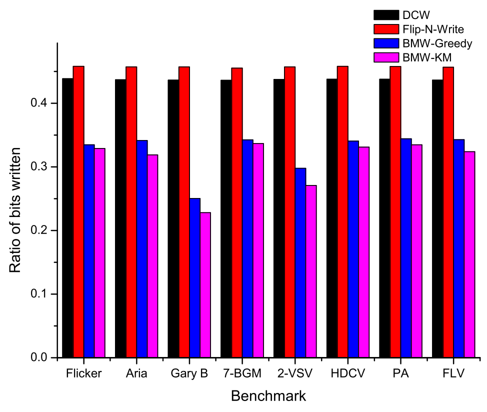
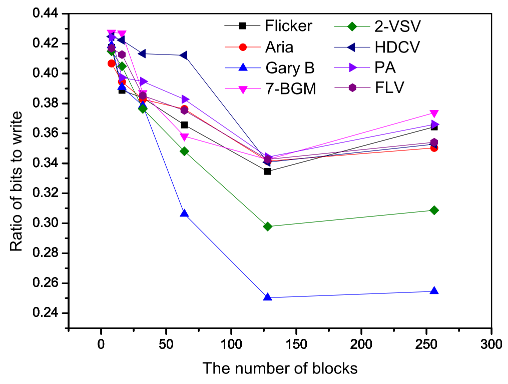
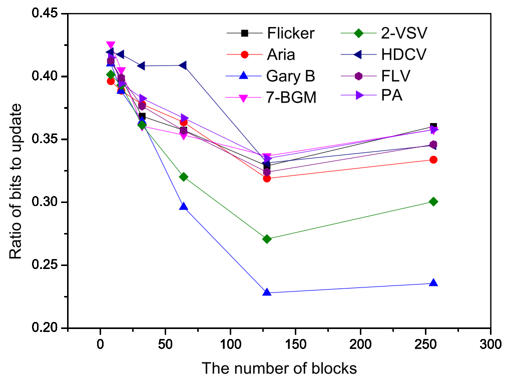


Figure 9. Comparison of experiment results for DCW, *Flip-N-Write*, *BMW-Greedy* and *BMW-KM*.

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



(a) Comparison of experimental results for *BMW-Greedy* with six partitioning method including partitioning a block into 8, 16, 32, 64, 128 and 256 sub-blocks, respectively.



(b) Comparison of experimental results for *BMW-KM* with six partitioning method including partitioning a block into 8, 16, 32, 64, 128 and 256 sub-blocks, respectively.

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.

TABLE IV.
EXPERIMENT RESULTS FOR DCW, *Flip-N-Write*, *BMW-Greedy* AND *BMW-KM*.

Algorithms	**Performance parameter**	**Flicker**	**Aria**	**Gary B**	**7-BGM**	**2-VSV**	**HDCV**	**PA**	**Flv**	**Average**
DCW	**Bits to update**	110342326	5233275648	2401846849	113285008	1860561313	6312100	29099002	309115657	1257979738
	Overhead	0	0	0	0	0	0	0	0	0
	Total writes	110342326	5233275648	2401846849	113285008	1860561313	6312100	29099002	309115657	1257979738
	Ratio	43.86%	43.70%	43.66%	43.62%	43.74%	43.78%	43.79%	43.64%	**43.72%**
Flip-N-Write	**Bits to update**	99545061	4729741277	2171911061	102023551	1679387687	5702848	26281288	279197261	1136723754
	Overhead	15724544	748492800	343834624	16232448	265828352	901120	4153344	44271616	179929856
	Total writes	115269605	5478234077	2515745685	118255999	1945216039	6603968	30434632	323468877	1316653610
	Ratio	45.82%	45.74%	45.73%	45.53%	45.73%	45.80%	45.80%	45.67%	**45.73%**
BMW-Greedy	**Bits to update**	76346877	3715710944	1204757283	80844966	1133708202	4462087	20793248	220628427	807156504
	Overhead	7862272	374246400	171917312	8116224	132914176	450560	2076672	22135808	89964928
	Total writes	84209149	4089957344	1376674595	88961190	1266622378	4912647	22869920	242764235	897121432
	Ratio	33.47%	34.15%	25.02%	34.25%	29.78%	34.07%	34.41%	34.27%	**32.43%**
BMW-KM	**Bits to update**	74887091	3445326584	1082340711	79371148	1018636830	4325723	20177742	207404939	741558846
	Overhead	7862272	374246400	171917312	8116224	132914176	450560	2076672	22135808	89964928
	Total writes	82749363	3819572984	1254258023	87487372	1151551006	4776283	22254414	229540747	831523774
	Ratio	32.89%	31.89%	22.80%	33.69%	27.07%	33.13%	33.49%	32.41%	**30.92%**
Improvement	**BMW-Greedy vs DCW**	10.39%	9.55%	**18.63%**	9.37%	13.96%	9.71%	9.37%	9.37%	**11.29%**
	BMW-Greedy vs Flip-N-Write	12.35%	11.59%	**20.71%**	11.28%	15.95%	11.73%	11.38%	11.39%	**13.30%**
	BMW-KM vs DCW	10.97%	11.80%	**20.86%**	9.93%	16.67%	10.65%	10.30%	11.23%	**12.80%**
	BMW-KM vs Flip-N-Write	12.93%	13.85%	**22.93%**	11.85%	18.66%	12.68%	12.31%	13.26%	**14.81%**

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 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 cstc2012ggC40005, NSFC 61173014, NSF CNS-1015802, Texas NHARP 009741-0020-2009.

REFERENCES

[1] W. Zhang and T. Li, "Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures," in *Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on*. IEEE, 2009, pp. 101–112.

[2] C. Lam, "Cell design considerations for phase change memory as a universal memory," in *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*. IEEE, 2008, pp. 132–133.

[3] M. Gill, T. Lowrey, and J. Park, "Ovonic unified memory-a high-performance nonvolatile memory technology for stand-alone memory and embedded applications," in *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, vol. 1. IEEE, 2002, pp. 202–459.

[4] L. Burcin, S. Ramaswamy, K. K. Hunt, J. D. Maimon, T. J. Conway, B. Li, A. Bumgarner, G. F. Michael, and J. Rodgers, "A 4-mbit non-volatile chalcogenide random access memory," in *Aerospace Conference, 2005 IEEE*. IEEE, 2005, pp. 1–8.

[5] S. Lai and T. Lowrey, "Oum-a 180 nm nonvolatile memory cell element technology for stand alone and embedded applications," in *Electron Devices Meeting, 2001. IEDM'01. Technical Digest. International*. IEEE, 2001, pp. 36–5.

[6] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 449–464, 2008.

[7] W. Xu, J. Liu, and T. Zhang, "Data manipulation techniques to reduce phase change memory write energy," in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 237–242.

[8] H. Li and Y. Chen, "An overview of non-volatile memory technology and the implication for tools and architectures," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09*. IEEE, 2009, pp. 731–736.

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

[10] R. Simpson, M. Krbal, P. Fons, A. Kolobov, J. Tominaga, T. Uruga, and H. Tanida, "Toward the ultimate limit of phase change in ge2sb2te5," *Nano letters*, vol. 10, no. 2, pp. 414–419, 2009.

[11] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y.-C. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, *et al.*, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 465–479, 2008.

[12] A. Pirovano, A. Lacaita, A. Benvenuti, F. Pellizzer, S. Hudgens, and R. Bez, "Scaling analysis of phase-change memory technology," in *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*. IEEE, 2003, pp. 29–6.

[13] N. Mielke, S. Hudgens, B. Johnson, and T. Lowrey, "Reliability of ovonic unified memory," in *Proceedings qf5'h Top Res. Conf. Reliability*, 2002.

[14] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. L. Lacaita, and R. Bez, "Reliability study of phase-change nonvolatile memories," *Device and Materials Reliability, IEEE Transactions on*, vol. 4, no. 3, pp. 422–427, 2004.

[15] K. Kim and S. J. Ahn, "Reliability investigations for manufacturable high density pram," in *IEEE 43rd Annual*

International Reliability Physics Symposium, San Jose, 2005, pp. 157–162.

[16] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H.-M. Sha, "Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 350–355.

[17] D. Roberts, T. Kgil, and T. Mudge, "Using non-volatile memory to save energy in servers," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 743–748.

[18] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 14–23.

[19] D. Kang, J.-H. Lee, J. Kong, D. Ha, J. Yu, C. Um, J. Park, F. Yeung, J. Kim, W. Park, *et al.*, "Two-bit cell operation in diode-switch phase change memory cells with 90nm technology," in *VLSI Technology, 2008 Symposium on*. IEEE, 2008, pp. 98–99.

[20] D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E. H.-M. Sha, "Curling-pcm: Application-specific wear leveling for phase change memory based embedded systems." in *ASP-DAC*, 2013, pp. 279–284.

[21] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: opportunities and challenges," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011, pp. 325–334.

[22] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*. IEEE, 2007, pp. 3014–3017.

[23] S. Cho and H. Lee, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 347–357.

[24] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[25] G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, *et al.*, "Phase change memory technology," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, vol. 28, no. 2, pp. 223–262, 2010.

[26] H. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[27] A. Lacaita, "Phase change memories: State-of-the-art, challenges and perspectives," *Solid-State Electronics*, vol. 50, no. 1, pp. 24–31, 2006.

[28] R. Neale, "Pcm progress report no. 7: A look at samsung's 8-gb array," Tech. Rep., September 1970.

[29] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: a protoype phase change memory storage array," *Proc. HotStorage*, p. 74, 2011.

[30] S. Lai, "Current status of the phase change memory and its future," in *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*. IEEE, 2003, pp. 10–1.

[31] D. Xu, "Carbon-containing interfacial layer for phase-change memory," May 20, uS Patent 6,566,700.

[32] T. Nirschl, J. Phipp, T. Happ, G. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, *et al.*, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*. IEEE, 2007, pp. 461–464.

[33] R. Venkata Rao and G. Waghmare, "A comparative study of a teaching-learning-based optimization algorithm on multi-objective unconstrained and constrained functions," *Journal of King Saud University-Computer and Information Sciences*, 2013.

[34] R. V. Rao and V. Patel, "An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems," *Scientia Iranica*, 2012.

[35] M. Zhang, "Energy-balanced distributed clustering algorithm in wireless sensor networks." *Journal of Software (1796217X)*, vol. 8, no. 8, 2013.

[36] G. Zhang, K. Zhang, X. Zhu, M. Chen, C. Xu, and Y. Shao, "Modeling and analyzing method for cps software architecture energy consumption." *Journal of Software (1796217X)*, vol. 8, no. 11, 2013.

[37] J. Yuan, X. Miao, L. Li, and X. Jiang, "An online energy saving resource optimization methodology for data center." *Journal of Software (1796217X)*, vol. 8, no. 8, 2013.

Yunjiao Peng received the BE degree in computer science and technology from the Tianjin Normal University, Tianjin, China, in 2011. She is currently working toward the ME degree in the institute of information science and engineering at the Hunan University.

Qingfeng Zhuge received her Ph.D. from the Department of Computer Science at the University of Texas at Dallas in 2003. She obtained her B.S. and M.S. degrees in Electronics Engineering from Fudan University, Shanghai, China. She is now a full professor at Chongqing University, China. She received Best Ph.D. Dissertation Award in 2003. She has published more than 60 research articles in premier journals and conferences. Her research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.

Duo Liu received the Ph.D. degree in computer science from the Department of Computing, The Hong Kong Polytechnic University in 2012. He received the B.E. degree in computer science from the Southwest University of Science and Technology, Sichuan, China, in 2003, and the M.E. degree from the Department of Computer Science, University of Science and Technology of China, Hefei, China, in 2006, respectively. He is currently an assistant professor with the College of Computer Science, Chongqing University, China. His current research interests include emerging memory techniques and high performance computing.

Jian Li received the BE degree in network engineering from the Chongqing University, Chongqing, China, in 2012. He is currently working toward the ME degree in the school of computing at the Chongqing University.

Edwin Hsing-Mean Sha (S88-M92-SM04) received Ph.D. degree from the Department of Computer Science, Princeton University, USA in 1992. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, USA. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas. Since 2012, he served as the Dean of College of Computer Science at Chongqing University, China. He has published more than 280 research 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.