This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

# Durable and Energy Efficient In-Memory Frequent Pattern Mining

Duo Liu, Yi Lin, Po-Chun Huang, Xiao Zhu, and Liang Liang

***Abstract*—It is a significant problem to efficiently identify the frequently-occurring patterns in a given dataset, so as to unveil the trends hidden behind the dataset. This work is motivated by the serious demands of a high-performance in-memory frequent-pattern mining strategy, with joint optimization over the mining performance and system durability. While the widely-used *frequent-pattern tree (FP-tree)* serves as an efficient approach for frequent-pattern mining, its construction procedure often makes it unfriendly for nonvolatile memories (NVMs). In particular, the incremental construction of FP-tree could generate many unnecessary writes to the NVM and greatly degrade the energy efficiency, because NVM writes typically take more time and energy than reads. To overcome the drawbacks of FP-tree on NVMs, this paper proposes *evergreen FP-tree (EvFP-tree)*, which includes a *lazy counter* and a *minimum-bit-altered (MBA) encoding scheme* to make FP-tree friendly for NVMs. The basic idea of the lazy counter is to greatly eliminate the redundant writes generated in FP-tree construction. On the other hand, the MBA encoding scheme is to complement existing wear-leveling techniques to evenly write each memory cell to extend the NVM lifetime. As verified by experiments, EvFP-tree greatly enhances the mining performance and system lifetime by 40.28% and 87.20% on average, respectively. And EvFP-tree reduces the energy consumption by 50.30% on average.**

***Index Terms*—Frequent pattern mining, FP-tree, FP-construct, nonvolatile memories (NVMs), phase-change memory (PCM), performance, energy efficiency, lifetime.**

## I. Introduction

Machine learning and data mining are the key technologies of data analytics, which reveal the hidden knowledge behind data [29], [45]. In the data mining area, it is a crucial problem to discover the frequently-occurring itemsets or patterns in a dataset, and there are many approaches such as Apriori [4] and frequent-pattern tree (FP-tree) [16]. While such in-memory frequent-pattern mining approaches assume that the to-be-mined data and the metadata structures are kept in byte-addressable memories, the energy efficiency and persistence could become a key design issue for the mining system, because dynamic random-access memory (DRAM) requires continuous energy supply to keep its data. As a result, emerging nonvolatile memories (NVMs)—such as phase-change memory (PCM) or spin-torque transfer random-access memory (STT-RAM)—are often considered powerful alternatives to replace DRAM for in-memory data analytics, due to their outstanding non-volatility and energy efficiency [38], [9], [19]. Unfortunately, when existing data mining and machine learning methods are used on NVMs, the mining performance or system lifetime might seriously suffer, due to the lacking of considerations over the special intrinsic characteristics of NVMs [6], [10], [35], [40]. This paper is thus motivated by making frequent pattern mining durable and efficient on NVM-based memory/storage systems by considering their characteristics. Our ultimate goal is to realize the ideals of high-performance, energy-efficient in-memory data analytics, while guaranteeing satisfactory durability of the mining memory/storage systems.

Several techniques have been proposed to find the frequently-occurring patterns in a dataset. For example, Apriori discovers the frequent patterns by candidate generation [4]. To avoid generating many candidates for frequent pattern discovery, Han et al. propose FP-tree [16] to compact the frequent pattern information in a space-efficient tree structure. CFP-tree [41] is a more compact data structure than FP-tree to further cut the space usage. There are also some work on the adaptivity enhancement of mining tree structures, e.g., AFPIM [20], CATS tree [12], CanTree [23], and CP-tree [42]. However, to our best knowledge, there are few frequent-pattern mining methods designed for NVMs up to now, and this work aims to fill up this gap.

NVMs have become competitive alternatives to DRAM, due to their comparable read/write performance and outstanding static energy consumption. However, many NVMs come with special characteristics that demand careful management to provide satisfactory access performance or memory lifetime. For example, as a popular NVM, PCM comes with two key characteristics: (a) A PCM write operation takes more time and energy than a read operation does [6], [40], [47]. How to reduce the unnecessary PCM writes is thus crucial for the performance [25], [44]; (b) A PCM cell wears and eventually becomes corrupted as it is repetitively written with data [10], [35]. It is thus necessary to equalize the writes to different PCM cells, so as to extend the PCM lifetime [10], [35], [26]. In summary, although NVMs are welcomed by high-performance data analytics due to the non-volatility and energy efficiency, existing machine learning or data mining algorithms cannot run seamlessly on NVMs, and new management designs are indispensable to enable durable and efficient (in terms of both time and energy) mining for frequently-occurring patterns on

A preliminary version of this paper was presented at the ACM/IEEE 2016 International Conference on Parallel Processing (ICPP 2016).

Duo Liu, Yi Lin, and Xiao Zhu are with the College of Computer Science, Chongqing University, Chongqing, 400044, China.

Liang Liang is with the College of Communication Engineering, Chongqing University, Chongqing, 400044, China.

Po-Chun Huang is with the Department of Computer Science and Engineering, Yuan Ze University, Taiwan.

Duo Liu is the corresponding author (E-mail: liuduo@cqu.edu.cn).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

NVMs.

This paper proposes a novel technique, named *evergreen FP-tree (EvFP-tree)*, to enable durable and efficient frequent pattern mining on NVMs. EvFP-tree consists of a *lazy counter* design and a *minimum-bit-altered (MBA)* encoding scheme to reduce and equalize the wearing of PCM cells, so as to enhance the mining performance and system lifetime. The lazy counter proposes to split the frequent-pattern mining process into two phases, so as to eliminate the redundant updates of the support counter of FP-tree nodes and to accelerate the mining process. On the other hand, the MBA scheme proposes to replace classical two's complement encoding scheme of integers, so as to prevent the least-significant bits (LSBs) from being quickly worn out and to extend the NVM lifetime. **Compared with previous work, the strength of this work is that we integrate the frequent pattern mining and NVM together to utilize the attractive property of persistent of NVM**.Please note that, although in this paper we take PCM as an example to design the EvFP-tree, the idea to exploit the intrinsic characteristics and operational constraints of memory/storage media to optimize the performance, energy efficiency, and durability of data mining or machine learning algorithms could be generalized to other NVMs, such as the spin torque transfer RAM (STT-RAM) [32] or resistive RAM (ReRAM) [28], [7], [8].

To evaluate the efficacy of the proposed EvFP-tree, we conduct a series of experiments based on publicly available realistic traces. The obtained results show that EvFP-tree could enhance the mining performance by 40.28% and system lifetime by 87.20% against the existing FP-tree approach. In addition, EvFP-tree reduces the energy consumption by 50.30% on average. (Although EvFP-tree focuses on PCM, it can be easily extended to other NVMs. Moreover, the design spirits of EvFP-tree are also preferred by other mining methods.)

Our major technical contributions include:

- We propose *EvFP-tree* to make frequent pattern mining durable, efficient and more friendly for NVM-based systems, by considering the specific features of NVMs.
- In EvFP-tree, we propose a lazy counter design and a minimum-bit altered (MBA) encoding technique to eliminate and distribute writes evenly across NVM cells.
- We conduct experiments based on a series of realistic traces, and the results demonstrate the effectiveness of our technique.

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III presents the proposed EvFP-tree scheme. Section IV reports the experimental results. Finally, we conclude this paper in Section VI.

## II. Background and Motivation

### A. Frequent-pattern Mining and FP-tree

*Frequent-pattern tree (FP-tree)* [16] is a very important data structure in *frequent pattern growth (FP-growth)* [16], which is proposed to solve the performance bottleneck of *Apriori* [4]. FP-growth and Apriori are both algorithms for digging out the key trends hidden in a massive dataset in the data mining area. By compressing the crucial information of frequent patterns into FP-tree structure, FP-growth can eliminate the costly candidate generation process of Apriori. The basic idea of FP-tree is to maintain each occurred pattern in a *node*, and keep a *support count* in the *counter* field of each node to represent how many times the pattern has occurred. All nodes in an FP-tree are then organized in a *prefix tree* or a *trie*, where two patterns with a common prefix will have a common ancestor node of the corresponding nodes of the two patterns in the FP-tree. The more the paths overlap with one another, the more compression we can achieve using a prefix-tree structure.

| TID | Data items |
|---|---|
| 1 | <a> |
| 2 | <b, c, d> |
| 3 | <e, f> |
| 4—5 | <a> |
| 6—7 | <f, g> |
| 8—10 | <f> |
| 11—14 | <b, c, d> |

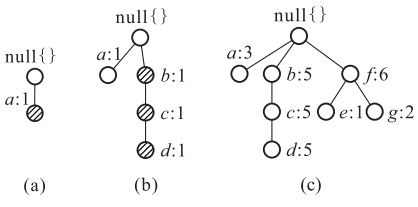



Fig. 1: Classical construction of an FP-tree [16]. (a) Transaction 1 $\langle a \rangle$ is discovered. (b) Transaction 2 $\langle b, c, d \rangle$ is discovered. (c) All 14 transactions are discovered.

To simplify the discussion on frequent-pattern mining, we will focus on the construction of FP-tree in this work because FP-tree is the main component of FP-growth, which is a widely used frequent-pattern mining method and was proposed to replace Apriori to achieve better performance on frequent-pattern mining.

To better understand the FP-tree, we will give some brief introduction of the definitions for FP-tree. Let $I = \{I1, I2, \cdots, Im\}$ be a set of items. $T$ is a transaction which consists of several items and *TID* is the transaction id which is an identifier of the transaction. A transaction database $D$ which consists of $T$ is a set of transactions. Let $P = \{P1, P2, \cdots, Pn\} \subseteq I$, where $n \in [1, m]$, be a set of items called a pattern. $P$ can be a subsequent itemset of $T$ when a pattern contained in $T$. The support of a pattern $P$ in a database is defined as the percentage of transactions in $D$ that contain $P$. If a pattern frequently appears in a database and the support of the pattern is no less than a given minimum support threshold ***filtering threshold*** $\delta$ is called a *frequent pattern*.

Fig. 1 shows the classical construction of an FP-tree. In the example dataset, each row represents a transaction with its *transaction ID (TID)* and the associated *data items* (called *itemset*); and transactions come in increasing order of TIDs, **as shown in Fig. 1, the column of TID.** In the beginning, only the root node of the FP-tree exists, and the corresponding pattern of the root node is simply an empty pattern, $\langle null \rangle$. As the first pattern $\langle a \rangle$ is discovered, the corresponding node for the pattern $\langle a \rangle$ will be created since no node other than the root node exists in the FP-tree. The support counters of the newly created nodes are initialized to 1, since the corresponding pattern $\langle a \rangle$ has occurred for the first time (Fig. 1(a)) (modified nodes are shadowed for clarity). Likewise, the discovery of the pattern $\langle b, c, d \rangle$ creates three new nodes for the patterns

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

$\langle b \rangle$, $\langle b, c \rangle$, and $\langle b, c, d \rangle$, where the support counter of each new node is also initialized to 1 (Fig. 1(b)). The construction then proceeds until all transactions in the dataset are scanned, and the FP-tree after the discovery of the whole dataset is shown in Fig. 1(c). Note that the items of a transaction will be sorted according to the descending order of their occurrence frequencies in the dataset before they are scanned, such as $e$ and $f$ are sorted as $\langle f, e \rangle$ in this example.

### B. Nonvolatile Memories

Recently, byte-addressable nonvolatile memories (NVMs), such as spin torque transfer RAM (STT-RAM) [32], magnetoresistive RAM (MRAM) [43], resistive RAM (ReRAM) [28], and phase-change memory (PCM) [6], [22], have become promising competitors of dynamic random-access memory (DRAM). While NVMs often provide comparable read and write performance against DRAM, their non-volatility makes them suitable for both the main memory and storage of power-constrained computing systems, such as smart phones and wireless sensor nodes. However, the extra operational constraints of NVMs have created a barrier from directly using NVMs as main memory: First, the writes to many NVMs are often much slower (often by 7–10 times) and consume much more energy than reads. It is therefore preferred to minimize the writes to NVMs to simultaneously enhance the performance, energy efficiency, and lifetime of NVMs. On the other hand, as an NVM cell is being written, it is gradually worn and eventually becomes corrupted. As a result, writes across all NVM cells should be made as even as possible (called *wear-leveling*). Such characteristic differences of NVMs often create barriers from directly using NVMs to replace DRAM, and extra management is often necessary to achieve acceptable performance and lifetime of NVMs. Thus this paper proposes a technique to reduce writes at algorithm level and prolong the lifetime when in-memory frequent-pattern mining approaches are applied on NVMs.

### C. Observations and Motivation

As observed by the example in Fig. 1, the classical FP-tree construction process is unaware of the read/write asymmetry and limited write endurance of NVMs when the main memory is replaced by NVMs in a computing system. Thus, several issues emerge when the FP-tree is created on NVMs like PCM. To better illustrate the problem, we collect the number of update operations in each node of FP-tree and the standard deviation of bit write counts for each support counter.

Fig. 2(a) shows the number of update operations in each node of chess dataset (A dataset from UCI Machine Learning Repository [1], [2], [5], please see the details in Section IV-A). From Fig. 2(a), it could be observed that the nodes with low index show much more frequent update features compared with the high-indexed nodes. This is because the tree is constructed from the root, and the support counter of all FP-tree nodes corresponding to *any* prefix of the pattern must be updated. So the nodes close to the root of the FP-tree will be frequently updated and keep larger counters compared with the others. As a result, it could seriously shorten the lifetime of NVM because of the great difference in the number of update operations among nodes in FP-tree. Moreover, writes to NVMs are often significantly slower than reads, repetitively updating the counters of FP-tree nodes is inefficient.

Fig. 2(b) shows the standard deviation of bit write counts for each support counter of chess dataset. It also exhibits the similar trend with the update operations that the FP-tree nodes with smaller index numbers often have the higher standard deviations of write counts. This is because that the nodes with smaller index numbers often have the larger support counter. With the classical two's complement, every increment operations will lead to the flip of the LSBs. In contrast, the MSBs are rarely flipped compared with the LSBs. This imbalance writes to NVM could also deteriorate it's endurance. Although existing wear-leveling techniques equally utilize the PCM words, they cannot effectively alleviate the biased wearing of bits/cells of the same PCM word. Moreover, although existing encoding schemes like the straight ring counter and **Johnson counter** [33] could replace the classical two's complement representation to equalize the wearing of bits/cells in a counter, the relative space efficiency is considerably harmed by 25%–50% (See Section III-C).

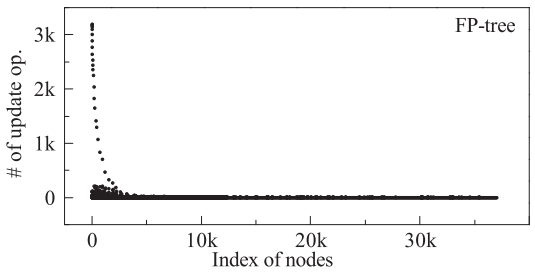
This work is motivated by the serious demand of a durable and efficient mining strategy for the frequently-occurring patterns in a dataset when the to-be-mined data are kept in NVM for in-memory data analytics. To do this, we propose promising solutions to frequent pattern mining problems for NVM-based computing systems, with the considerations over the read/write asymmetry and write endurance issues of NVMs. The technical problem is how to reduce the number of writes to NVMs because NVMs usually have better performance on reads, compared to writes. Another technical problem falls on how to minimize the number of bit flips on updating pattern counters and even out the bit flips of NVM cells for the discovery of frequent patterns. This work takes the FP-tree construction over PCM for example to demonstrate how our designs reduce writes, minimize the number of bit flips, and evenly distribute bit alterations/flips during the construction of frequent patterns. This is because FP-growth with FP-tree are widely adopted in frequent pattern mining and PCM is a popular alternative to replace DRAM as main memory especially when the latest 3D XPoint jointly released by Intel and Micron could be a way to realize PCM products. Besides, frequent pattern mining methods (including FP-tree) usually have the same problems over NVM-based systems, and the designs proposed in this paper could be easily extended to other mining methods.

## III. Evergreen FP-tree

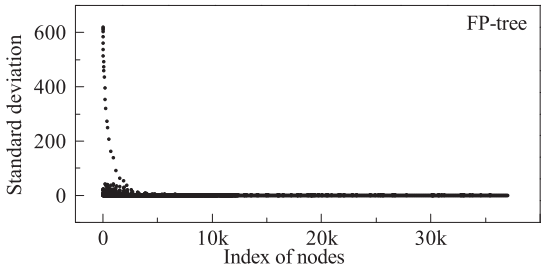
### A. Overview

This section presents the *evergreen FP-tree (EvFP-tree)*, which makes FP-tree friendly for NVMs. In EvFP-tree, we propose a novel construction strategy called *lazy counter* for FP-trees to greatly reduce the number of write operations generated during the tree construction process (Section III-B). To prevent the support counter field of FP-tree nodes from being worn out quickly, we also propose a *minimum-bit-altered (MBA)*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems




(a) The *chess* dataset ($\delta = 0.1$).




(b) The *chess* dataset ($\delta = 0.1$).

Fig. 2: Reduction of write operations and the standard deviation of bit write counts of each node in the FP-tree

*encoding* to evenly distribute the writes to NVM bits/cells in the same counter (Section III-C). Moreover, we proposed how to efficiently organize and manage the (potentially many) children of the same FP-tree node.

### *B. Lazy Counter*

This section presents the *lazy counter*, an effective two-phase construction strategy of FP-trees to reduce the writes to the PCM (See Fig. 1). In order to minimize the unnecessary writes, as a transaction is scanned, not the counter of every FP-tree node along the path between the root node and the node of the whole transaction (whose data items are sorted according to the descending order of their occurrence frequencies) is updated. Instead, only the counter of the FP-tree node of the whole transaction is updated. Once all transactions have been scanned, a linear-time *summarizing* process is carried out in a depth-first fashion, where the counter of each FP-tree node is updated as the sum of the counter of the node itself and the counter of all child nodes. After the summarizing process, the counter of each FP-tree node is fixed as the number of occurrences of the corresponding pattern of the FP-tree node, as in the original FP-tree design. This is based on the observation that *the support counter of a nonleaf node is the sum of the support counter of all offspring nodes and that of itself before the summarization*. With the lazy counter, the scanning of each transaction would modify the support counter of exactly one FP-tree node. After that, the summarization process would modify the support counter of each FP-tree node for at most once. In contrast, with existing FP-tree approach, scanning a transaction with $n$ items would modify the support counter of exactly $n$ FP-tree nodes. As can be observed, the number of overall counter modifications (in both phases) with the lazy counter technique is always less than

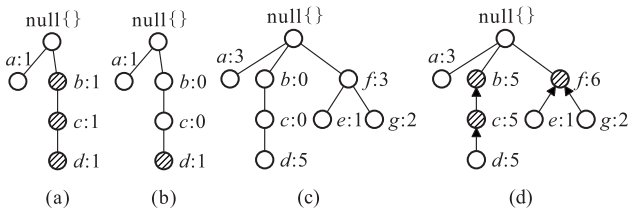



Fig. 3: Lazy counter and two-phase FP-tree construction strategy. (a) Classical construction of FP-tree. (b) The proposed construction strategy. (c) The FP-tree before summarizing. (d) Summarizing process.

that (in one phase) without the lazy counter technique. Note that such a performance difference will be exaggerated when transactions with many data items exist in the dataset.

In order to better clarify the lazy counter and two-phase construction of FP-trees, let us consider the dataset in Fig. 1 again. When a pattern is discovered for the first time, such as Transactions 2 $\langle b, c, d\rangle$, three corresponding FP-tree nodes will be created for the itemsets $\langle b\rangle$, $\langle b, c\rangle$, and $\langle b, c, d\rangle$ respectively, as in the classical FP-tree construction process. However, with the classical FP-tree construction process, the support counter of all three FP-tree nodes will be written as 1 (See Fig. 3(a)). However, with the proposed lazy counter technique, only the support counter of the node for the itemset of the whole transaction $\langle b, c, d\rangle$ is written to 1, leaving the support counters of the other three nodes as 0 **(See Line 8 of Algorithm 1 and Fig. 3(b))**. Likewise, when a pattern has been discovered, it will have a corresponding node in the FP-tree. For example, as Transaction 11 is scanned, each of the three prefixes $\langle b\rangle$, $\langle b, c\rangle$, and $\langle b, c, d\rangle$ already had a corresponding FP-tree node. In the classical FP-tree construction process, the support counter of all three nodes $\langle b\rangle$, $\langle b, c\rangle$, and $\langle b, c, d\rangle$ will be modified. In contrast, with the lazy counter technique, only the counter of the node $\langle b, c, d\rangle$ is modified **(See Line 6 of Algorithm 1).**

Once all transactions in the dataset have been scanned, a depth-first summarizing process is carried out to update the value of the support counter of each FP-tree node to the number of occurrences of the corresponding pattern in the dataset **(See Lines 11–15 of Algorithm 1 and Fig. 3(c), where arrow symbol represents the summarized process).** In the summarizing process, a depth-first traversal is carried out on the FP-tree, and the support counter of each traversed FP-tree node is updated as the sum of (a) the current counter value of the traversed node, as well as (b) the counter value of all children of the traversed node. For example, to summarize the counter of the node $\langle f\rangle$, we add the counters of all its child nodes, i.e., the counter of nodes $\langle f, e\rangle$ $(= 1)$ and $\langle f, g\rangle$ $(= 2)$, to the counter of the parent node $\langle f\rangle$ $(= 3)$. The counter of the FP-tree node $\langle f\rangle$ is thus fixed as $3 + 1 + 2 = 6$ **(See Line 15 of Algorithm 1 and Fig. 3(d))**. Detailed algorithm is shown as in Algorithm 1. **Lines 1–10 of Algorithm 1 show the first learning phase, and Lines 11–15 of Algorithm 1 show the second summarized phase. Lines 5–6 of Algorithm 1 show**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

**Algorithm 1** EvFP-tree with lazy counter.

**Input:** Dataset $D$; item list $L$ sorted in descending order of appearance frequency of items.
**Output:** EvFP-tree $T$ of the dataset $D$.

```
1: Create an EvFP-tree T with a root node labeled null.
2: for each transaction x ∈ D {p and ℓ are the first and the
   remaining items of x, respectively. N is the node in EvFP-tree.}
   do
3:    for each item p ∈ [p|ℓ] do
4:       if ℓ = null then
5:          if N.itemset = p.itemset then
6:             p.counter++;
7:          else
8:             Create a new node N with N.counter = 1;
9:       else
10:         Create a new node N with N.counter = 0;
11: Procedure EvFPtree_summarize(N).
12: if node N ≠ null then
13:    for each child node N_c of node N do
14:       Call EvFPtree_summarize(N_c);
15:    N.counter+= ∑_{∀children N_c of N} N_c.counter;
```

**the modification of the support counter of exist nodes in EvFP-tree. Lines 7–10 of Algorithm 1 show the creation of new nodes.**

Although the design of the lazy counter seems straightforward, its main idea can be extended to many other problems in diversified application domains. By relaxing an online problem to an offline one, we can have multiple inputs of the problem, instead of only one, at a time. As a result, one might be able to exploit the bulk information provided by the multiple inputs to optimize the algorithms that solve the problem. For our lazy counter design, by summarizing the support counters after processing the whole dataset (or just a part of it), the update costs of the PCM can be reduced, and the mining performance can be enhanced. In another example, a key can be inserted to or deleted from a red–black search tree with $n$ keys in $O(\lg n)$ time in the worst case. However, it is shown that $k$ consecutive keys can be inserted to or deleted from a red–black tree with $n$ keys in just $O(k+\lg n)$ time in the worst case, by exploiting the characteristics of red–black tree and the bulk information given by the multiple sequential keys [37]. Although the idea to exploit the bulk information provided by multiple inputs for performance optimization is straightforward, it remains an interesting problem on how to realize the idea on a given online algorithm in a practical fashion.

### C. Minimum-bit-altered (MBA) Encoding

To evenly distribute the writes over the bits/cells in the same word, we propose a novel scheme called *minimum-bit-altered (MBA)* encoding that can be incorporated into the support counter of FP-tree nodes. As TABLE I shows, the proposed MBA encoding scheme (shown as the bottommost row) perfectly equalizes the wearing of each bit of the counter, where each bit/cell is flipped for exactly four times as the counter is increased from 0 to 15 then back to 0. Although the MBA encoding scheme is a special case of Gray Code, it addresses a different key design problem to evenly distribute the bit writes in the same PCM word, so as to prolong the PCM lifetime. And the MBA is restricted to one points: Each bit/cell of the counter is flipped for exactly four times as the counter is increased.

Although the proposed construction strategy, EvFP-tree, effectively eliminates unnecessary updates to the counters of the FP-tree nodes, the imbalanced wearing of cells/bits still needs to be solved. If the counters of FP-tree nodes use classical binary encoding, the LSBs will be modified much more frequently and could be worn out much earlier than the other bits [27]. For example, the least significant bit of a counter will be flipped every time when the counter is increased. Moreover, the number of bits updated by an increment operation could be unpredictably large, due to the propagation of carry bits [27]. For example, when a counter $15_{10} = 00001111_2$ is increased, five bits of the counter must be altered, turning the value of the counter into $16_{10} = 000\underline{10000}_2$ (the altered bits are underlined). Thus, the time to update a counter could be unpredictably long, and the system performance is harmed. However, in the MBA encoding scheme, at most one bit is updated to update a counter. And most importantly, the bit flip in each update operation is in different location with MBA. In this way, the huge gap in the number of writes between LSBs and MSBs can be reduced.

There have been existing counter encoding schemes that reduce the number of bits altered as the counter is increased. In particular, a straight ring counter [30], [33] and a **Johnson counter** [30], [33] alter only two bits and one bit as the counter is increased, respectively. (We assume that *selected bit writing* [46] is available to avoid writing the unmodified bits to reduce unnecessary wearing of PCM cells.) However, for a four-bit counter, a straight ring counter and a **Johnson counter** can represent only four and eight different states/values, respectively, whereas the two's complement representation can hold $2^4 = 16$ states/values. As a result, the straight ring counter and **Johnson counter** could lead to considerable relative space wastes of $2/4 = 50\%$ and $(4-3)/4 = 25\%$, respectively, because more bits/cells are needed to hold the same range of the counter values as with classical binary encoding. When the counter length is long, such a considerable waste of PCM space could be unacceptable, and new, space-efficient encoding schemes are needed.

To summarize, the proposed MBA encoding scheme exhibits the advantages of straight ring or **Johnson counter**, but is still optimal in terms of space efficiency. For counters with more bits in MBA, such as 16 or 32 bits, the counter can be split into multiple nibbles (i.e., *half bytes*), where each nibble is a standalone digit that stores a hexadecimal value. When a lower digit overflows, the next higher digit will be updated. As a result, a 32-bit counter can hold eight nibbles and represent a value between 0 and $(2^{32}-1)$, just like the two's complement scheme. In the worst case, the number of bit alterations on each increment operation of the counter is reduced from 32 (with classical binary encoding scheme) to 8 (with the proposed encoding scheme). Table II shows an example of the proposed binary encoding scheme, it can be observed that a 16-bit counter with the MBA encoding scheme

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE I: Comparison of the counter encoding schemes with modified bits being underlined.

| Decimal value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2's complement | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Straight ring | 0000 | 0001 | 0010 | 0100 | 1000 | — | — | — | — | — | — | — | — | — | — | — |
| Johnson | 0000 | 0001 | 0011 | 0111 | 1111 | 1110 | 1100 | 1000 | — | — | — | — | — | — | — | — |
| MBA | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 1111 | 1011 | 1001 | 1101 | 0101 | 0100 | 1100 | 1110 | 1010 | 1000 |

TABLE II: A 16-bit counter in the MBA encoding.

| Decimal | MBA encoding | # Altered Bits |
|---|---|---|
| 0 | 0000 0000 0000 0000 | 4 |
| 1 | 0000 0000 0000 0001 | 1 |
| 2 | 0000 0000 0000 0011 | 1 |
| ... | ... | ... |
| 15 | 0000 0000 0000 1000 | 1 |
| 16 | 0000 0000 0001 0000 | 2 |
| ... | ... | ... |
| 65,535 | 1000 1000 1000 1000 | 1 |

stores a counter value up to $(2^{16}-1)$, and at most four bits (reduced from 16 with the classical two's complement scheme) are altered in each increment operation.

When the depth-first summarizing is done to fix the value of the counter of each FP-tree node, an issue might arise that addition cannot be directly done with the proposed encoding scheme. This could be easily solved by storing a small table in a preserved area of PCM or in ROM to keep the mapping between classical binary encoding and the proposed MBA encoding. Because we use a nibble to keep a hexadecimal digit, only 16 entries are needed to store the mapping table. To avoid extra reads to PCM during the depth-first summarizing process, this small mapping table could be loaded into SRAM-based registers of the memory controller on device. Alternatively, special hardware, such as *application-specific integrated circuits*, can be included in the device to further reduce the overheads of increment and add operations of the counters in our MBA encoding scheme.

Although there are a number of excellent wear-leveling techniques proposed for PCM [10], [35], the MBA scheme is still necessary to ensure a reasonable PCM lifetime. This is because that existing PCM wear-leveling techniques only emphasize the equalized wearing of different PCM words or bytes; they do not focus on how to equalize the wearing of different cells/bits in the same PCM word. As the proposed MBA scheme tries to equalize the wearing of different cells/bits in the same PCM word, it fills the gap of *intra-word wear-leveling*, and can be orthogonally used with existing *inter-word wear-leveling* techniques to maximize the PCM lifetime and to minimize the wear-leveling overheads.

### D. Implementation Remarks: Expansion of FP-tree Nodes

As an FP-tree is constructed, a time-costly *node expansion* operation might be triggered to increase the number of child node-links of an FP-tree node, so as to accommodate the newly discovered patterns. Due to the slow write speed and limited capacity of PCM, the node expansion shall be efficient in terms of both time and space. This section presents how to expand an FP-tree node to maintain the structure of an FP-tree.

In our proposed FP-tree node design, each node has four fields, namely:

- The key $k$ that represents the corresponding pattern of the FP-tree node.
- A node-link $p$ to the parent FP-tree node.
- A node-link $f$ to its first child FP-tree node.
- A pointer $v$ to the *overflow chain* that refers to an array of node-links to child FP-tree nodes. (See the explanation below for details.)
- A counter $c$ that keeps the support count of the corresponding pattern of the FP-tree node.

As an FP-tree node is first allocated, its node-link $p$ will be initialized to the parent of the newly created FP-tree node. (Since the root of an FP-tree always refers to the pattern $<$null$\{\}>$, we can safely claim that the node-link of a newly-created FP-tree node always refers to a valid FP-tree node.) The node-links $f$ and $v$ are initialized to $-1$, indicating that there is currently no children of the FP-tree node. In the meantime, the counter $c$ is then initialized to 1, indicating that the first support of the pattern is just encountered.

After a new FP-tree node is successfully allocated and initiated, the child node-link of its parent node must be updated to refer to the new node as well. In particular, an empty child node-link will be used to refer to the new FP-tree node. However, if the parent FP-tree node does not have any free child node-links available and its overflow chain pointer $v$ is unused ($=-1$), an *overflow structure* with an overflow chain pointer $v$ and an array $A$ with three child node-links $A[0]$, $A[1]$, and $A[2]$ will be allocated. As in an FP-tree node, the overflow chain pointer $v$ of the newly-allocated overflow structure is initialized to $-1$. The first free entry $A[0]$ is then used to store the node-link to the new child node, while the remaining node-links $A[1]$ and $A[2]$ are initialized to $-1$, waiting for the allocation of subsequent child nodes. Once all three node-links $A[0]$, $A[1]$, and $A[2]$ are used and yet another new child node needs to be inserted, an overflow structure with one overflow chain pointer and seven child node-links will be allocated in PCM. In general, suppose that a new child node is inserted to the FP-tree, and its parent node does not have any free child node-links. If the last overflow structure of the overflow chain of the parent node has $(2^k-1)$ child node-links, a new overflow structure with $(2^{k+1}-1)$ child node-links will be allocated. Because the overflow chain pointer $v$ is of the same size as a child node-link, the overflow structures are of the size of exponential of two, i.e., 2, 4, 8, ... node-links, and the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

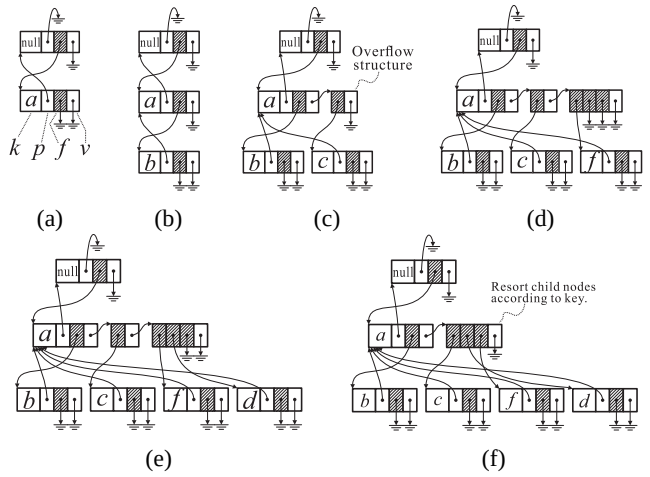



Fig. 4: The expansion procedure of a node: (a) Pattern $<a>$ discovered. (b) Pattern $<a, b>$ discovered. (c) Pattern $<a, c>$ discovered. (d) Patterns $<a, f>$ discovered. (e) Patterns $<a, d>$ and $<a, e>$ discovered. (f) After flattening the overflow chain. Shaded fields are child node-links in nodes or overflow structures. Irrelevant fields to the node expansion, such as the support count, are omitted for clarity.

TABLE III: Example for node expansion.

| TID | Data items | TID | Data items | TID | Data items |
|---|---|---|---|---|---|
| 1 | $<a>$ | 3 | $<a, c>$ | 5 | $<a, d>$ |
| 2 | $<a, b>$ | 4 | $<a, f>$ | 6 | $<a, e>$ |

space allocation of overflow structures can be easily realized by a *buddy system*.

In some application scenarios, an FP-tree node could expand seriously and have a considerable number of child nodes. In this case, the traversal overheads of the child node-links of the FP-tree during the depth-first summarizing process or mining process of the FP-tree might be unacceptably amplified. Thus, when the number of child node-links grows beyond a predetermined threshold $2^h$, we can "flatten" the child node-links of all existing overflow structures into a single overflow structure with an array of $(2^h - 1)$ node-links, where all but one (that has the smallest key) of the node-links are sorted by the ascending order of the key. As a result, binary searches can be applied to locate the to-be-traversed key in all but one of the child node-links of the same FP-tree node. Note that the threshold is set as an exponential of two to conform to the space allocation manner of a buddy system. To see how a node of FP-tree expands to accommodate more child nodes, an example is shown as in Table III and Figure 4.

## IV. Evaluation Studies

This section evaluates the efficacy of the proposed EvFP-tree on PCM, a popular flavor of NVMs. In particular, Section IV-A presents the experimental settings and the publicly-available datasets used throughout the experiments. After that, the experimental results in terms of the mining overheads and the memory lifetime are presented and discussed in Section IV-B.

TABLE IV: Characteristics of datasets for the experiments.

| Dataset | # Trans. | # Items | Max. Tran. Leng. | Avg. Tran. Leng. | Size (MB) |
|---|---|---|---|---|---|
| *T10I4D100K* | 100,000 | 870 | 29 | 10.10 | 4.00 |
| *T40I10D100K* | 100,000 | 942 | 77 | 39.61 | 15.5 |
| *chess* | 3,196 | 75 | 37 | 37.00 | 0.34 |
| *mushroom* | 8,124 | 119 | 23 | 23.00 | 0.57 |
| *pumsb** | 49,046 | 2,088 | 63 | 50.48 | 11.3 |
| *connect* | 67,557 | 129 | 43 | 43.00 | 9.3 |
| *pumsb* | 49,046 | 2,113 | 74 | 74.00 | 16.7 |
| *accidents* | 340,183 | 468 | 51 | 33.81 | 35.5 |
| *C73D10* | 10,000 | 1592 | 73 | 73 | 3.2 |
| *C20D10* | 2000 | 174 | 20 | 20 | 0.16 |

### A. Experimental Setup

*1) Experimental Environments and Configurations:* To evaluate the effective of the proposed EvFP-tree technique, we conducted a series of trace-driven simulation on a workstation featured with two CPUs clocked at 2.6 GHz and 64 GB DRAM, and the system is running RedHat 6.0 with kernel version 2.6.3. According to [18], the latencies of read, write, and reset operations of PCM are set to 6.82 ns, 152.20 ns, and 12.20 ns, respectively. Besides, the energy consumption of read, wirte, and reset operations of PCM are set to 0.064 nJ, 0.07 nJ, and 0.876nJ, respectively. To fairly assess the efficacy of EvFP-tree, we assume that there is a four-way associative cache [21], [34], [39] and the cache size ranges from 4 KB to 32 KB, which is managed with the least-recently-used (LRU) replacement policy. The size of each cache line is set to 64 bytes. For the experiments for the MBA encoding scheme, we assumed that there is an SRAM table to facilitate efficient conversion between MBA encoding and two's complement encoding. To fairly evaluate the overheads incurred by the table lookup, we assume that the read and write latencies of the SRAM table are both 1.41 ns, and energy consumption of the SRAM table is 0.004nJ [18].

In the experiments, we selected two synthetic sparse datasets *T10I4D100K* and *T40I10D100K* from IBM Almaden Quest research group [1].(A dataset is called "sparse" if it contains few high-frequency itemsets, and called "dense" otherwise.) In addition, to present the realistic scenario, we chose five realistic dense datasets *chess*, *mushroom*, *pumsb**, and *connect* from UCI Machine Learning Repository [1], [2], [5], a realistic dense dataset *accidents* from Karolien Geurts [1], [15], and two dense datasets *C73D10* and *C20D10* from the PUMS Kansas 1990 census data [3]. For convenience, the basic properties of each dataset are given by Table IV, where "# Trans." and "# Items" are the numbers of transactions and distinct data items in the dataset, respectively. The fields "Max. Tran. Leng." and "Avg. Tran. Leng." indicate the maximum and average length of the transactions in the dataset, respectively. Last but not least, the rightmost field indicates the file size of each dataset in megabytes. Here, please note that the performance benefits of the lazy counter design could be potentially magnified as the FP-tree grows higher, while the efficacy of the MBA encoding scheme is not seriously affected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

by the dataset size. Thus, the datasets we used are reasonably large for the evaluation of the performance and durability of the proposed EvFP-tree.

To properly assess the enhancement of the proposed schemes in terms of write reduction and bit-level wear-leveling effects, all items with occurrence probability less than a *filtering threshold* $\delta = 0.1$ will be removed before the FP-tree is constructed. However, since the dataset *T10I4D100K* contains only few high-frequency data items with occurrence frequency higher than 0.1, $\delta$ is set to 0.01 for this dataset. Moreover, $\delta$ is set to 0.2 for the dataset *accidents* and $\delta$ is set to 0 for the dataset *C20D10*.

*2) Evaluation Metrics:* We evaluated the proposed lazy counter and MBA encoding with the following metrics:

- **Number of read and update operations of FP-tree nodes**: The objective of the proposed FP-tree construction strategy is to reduce the unnecessary writes to PCM to repetitively increase the support counters in pattern recovery. To evaluate the efficacy of lazy counter (and rule out the effects of MBA scheme), the number of nodes being read and that being written are reported (See Figs. 5–6 and the accompanying explanations for details).
- **Maximum/average read/write counts of PCM cells**: The objective of the proposed counter encoding scheme is to equalize the wear of each bit in a counter. With the differential write feature [46], which means that a PCM cell is written only when its content is changed, we calculate the average write count of every PCM cell to examine the write traffic reduction to the PCM. In the meantime, the maximum write count among all PCM cells is reported to examine the lifetime and durability of the PCM for frequent pattern mining (Figs. 7–8).
- **The standard deviation of bit write counts of every node:** To observe the distribution of write activities over each FP-tree node, we calculate and plot the standard deviation of the bit write counts for each support counter, so as to see how fast the corresponding PCM cells will wear out (Fig. 9).
- **Total execution time, total energy consumption, and PCM lifetime**: To fairly evaluate the mining performance, we included all memory-access overheads to derive the time and energy consumption needed to process each dataset. In addition, assuming that PCM lifetime is $10^6$ writes, we repeated the datasets and estimated the number of transactions sustained before any PCM cell reaches its write lifetime (Figs. 10–12).

### B. Experimental Results

*1) Reduction of Read and Update Operations of FP-tree Nodes:* In this section, the effects of the reduction of read and update operations of *FP-tree nodes* generated during the FP-tree construction will be evaluated. Because the MBA scheme only affects the read and update operations of *memory cells/bits*, the results presented in this section reflect the effects of only the proposed lazy counter design. Figs. 5 and 6 show the total number of read and update operations of FP-tree nodes, respectively. The x-axis shows the cache size range from 4 KB to 32 KB for each dataset. Here, the read operations are due to the traversal of FP-tree structure and lookup of FP-tree nodes, while the update operations are due to the creation of FP-tree nodes and the updating of support counter of FP-tree nodes. From Figs. 5 and 6, it could be observed that EvFP-tree could considerably reduce the number of node update operations generated during the construction of FP-tree, because the lazy counter removes all unnecessary node updates by postponing the summarizing of support counters until the whole dataset is scanned. Moreover, the read operations are also reduced because a read operation must be carried out before an FP-tree node can be taken into the buffer for updates. While the write operations are reduced by the proposed lazy counter, so are the read ones. By varying the cache size, we observe that all datasets present a similar trend in reducing read and write operations. As expected, a smaller cache size lead to more read and write operations. Due to the good locality in dense datasets, it can get good performance with a small cache size. There is no big difference in big cache size compared with small cache size.

*2) Durability Enhancement of the NVM for Frequent-pattern Mining:* This section evaluates the durability enhancement effects of the PCM for frequent-pattern mining. To do this, we evaluated the average and maximum number of write operations to the PCM cells. While both the lazy counter and MBA encoding scheme could help reduce the number of write operations, only the MBA scheme equalizes the writes across different PCM cells. The experimental results presented in this section reflect the overall efficacy of durability enhancement of the whole EvFP-tree design, including the lazy counter and the MBA scheme.

Fig. 7 shows the maximum write count among all PCM cells after the mining process of each dataset is completed. To clearly show the effects of the proposed MBA scheme, we assumed that no wear-leveling strategy is used. (Please note that, although various wear-leveling techniques could be adopted to extend the PCM lifetime, our proposal of the MBA encoding scheme can accompany existing wear-leveling techniques to alleviate the performance overheads of data moving or space allocation.) In Fig. 7, it could be observed that EvFP-tree could reduce the maximum bit write counts by up to 98 times (occurred on the *accidents* dataset). Thus, if EvFP-tree is used, existing wear-leveling techniques can be less aggressive in data moving and less careful in space allocation, which further reduces the performance overheads. As compared to the results of average bit write count in Fig. 8, the relative reduction of average bit write count is significantly less than that of the maximum bit write count on certain datasets, such as *chess* and *accidents*, but not on other datasets, such as *connect* and *pumsb*$^*$. This is due to the different distribution of frequent itemsets in different datasets: For the datasets with some very frequent itemsets, the support counter of the corresponding FP-tree nodes of the frequent itemsets will be repetitively updated, thereby increasing the maximum bit write count. In contrast, if all itemsets are rare in the dataset, the effectiveness of EvFP-tree will be degraded, resulting in two phenomena:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

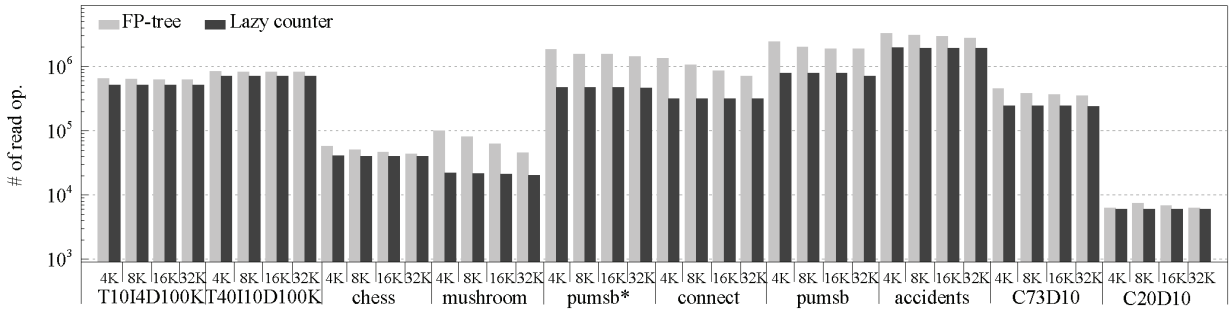



Fig. 5: Reduction of read operations of the counter.

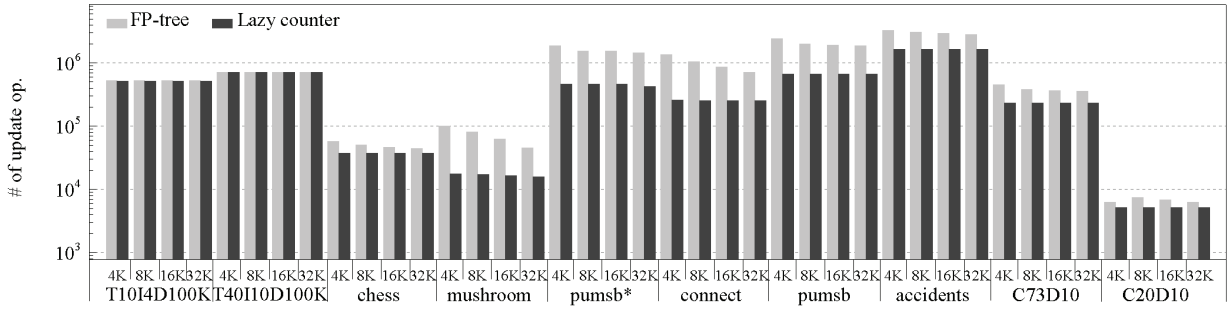



Fig. 6: Reduction of update operations of the counter.

- The relative reduction of the maximum bit write count of EvFP-tree against classical FP-tree scheme is smaller.
- The difference between the relative reduction of the average bit write count of EvFP-tree and that of the maximum bit write count is minor as well.

While the average and maximum bit write counts could provide an overall evaluation of the efficacy of EvFP-tree, it would be interesting to observe the standard deviation of the write count of the PCM cells/bits in each support counter. As shown in Fig. 9(a)–(j), the EvFP-tree (shown in the left part of each subfigure) has much lower standard deviation of bit write counts than the classical FP-tree design (shown in the right part of each subfigure) in most of the datasets. What worths noting is that the relative reduction of the standard deviation is most noticeable for FP-tree nodes with high standard deviation and low index. This is because the FP-tree is constructed from the root, and the low-indexed nodes that are generated early would be the shallow nodes close to the root of the FP-tree. When subsequent itemsets are being mined, the shallow, low-indexed FP-tree nodes would be more frequently updated, resulting in excessive skewed bit writes to the PCM cells of such nodes. In contrast, the standard deviations of write counts are much more even with the proposed EvFP-tree construction strategy. Because our proposed lazy counter construction strategy effectively eliminates unnecessary increments to the support counters, and the proposed MBA scheme could equalize the updates to different bits in the support counters of FP-tree nodes, the standard deviations of the bit-level write counters could be greatly reduced.

*3) Enhancement of the Overall Mining Performance:* To evaluate the overall mining performance of EvFP-tree, the total time and total energy consumption to process each dataset is calculated and shown in Fig. 10 and Fig. 11, respectively. From the figure, it can be observed that EvFP-tree can reduce the total processing time of each dataset by 40.28% on average when cache size is 4 KB. And the total energy consumption of each dataset can be reduced by 50.30% on average when cache size is 4 KB. Moreover, the effect of relative reduction of total process time is better for dense datasets (such as *Mushroom* and *Pumsb**), because both the lazy counter and MBA scheme are more effective for write reduction and bit-level wear-leveling on frequently occurring itemsets, respectively.

In addition, to evaluate the benefits of our strategies on lifetime, we repetitively process each dataset until any PCM cell is worn out, and observe the number of transactions sustained before a PCM cell reaches its lifetime threshold, i.e., $10^6$ writes. With the proposed EvFP-tree and classical FP-tree respectively, the PCM lifetime in terms of the number of sustained transactions before any PCM cell becomes worn out is shown in Fig. 12. Results show that our strategy can improve the number of sustained transactions by 87.20% on average when cache size is 4 KB.

## V. Related work

Apriori is one of the earliest methods to discover the frequently-occurring patterns in a dataset [4]. However, it often suffers from serious space overheads due to the generation of candidate patterns. To solve the problem of Apriori, many algorithms have been proposed. For instance, the proposal of frequent pattern tree (FP-tree) by Han et al. [16] solves the bottleneck of Apriori through compressing the crucial information of frequent patterns into a compact tree structure.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

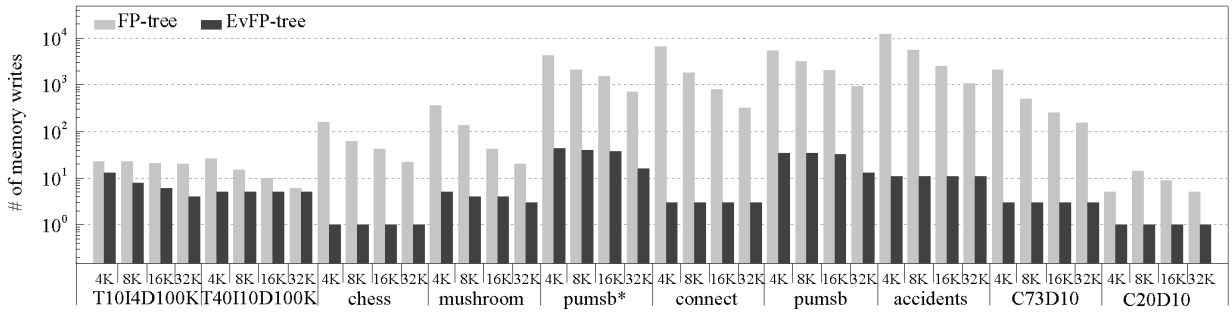



Fig. 7: The maximum bit write count among all PCM cells.

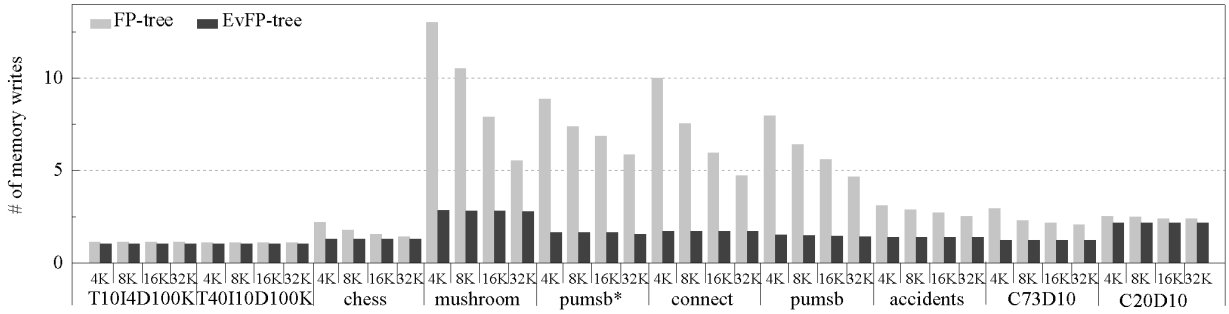



Fig. 8: The average bit write count of PCM cells.

In this way, as a database is processed for frequent-pattern mining, it needs to be scanned for only two rounds, which is much more efficient than Apriori. Another good news is that the generation of candidates of frequent patterns is no longer needed because the candidates can now be maintained in a space-efficient prefix-tree structure. The improvements on FP-tree can be divided into two major categories as follows.

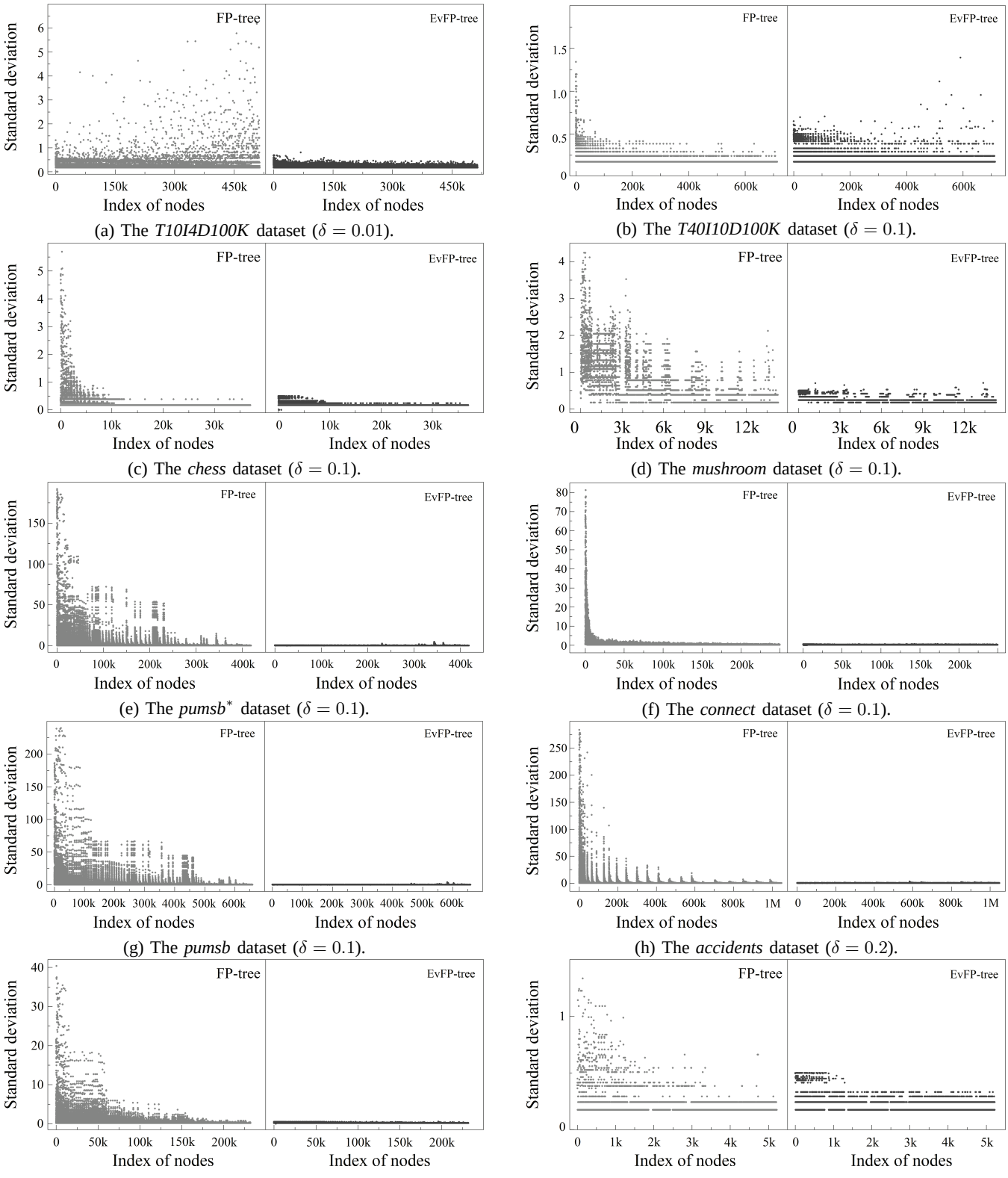
**Compact tree structure:** CFP-Tree [41] and nonordfp [36] concentrate on the construction of a more compact data structure than FP-tree so as to save space. CFP-Tree keeps the count of the node in a count array. All subtrees which share items of the root of the FP-Tree are combined together and move to the leftmost branch in CFP-Tree. So the number of nodes in a CFP-tree could be reduced by up to half as many as in an FP-tree. In nonordfp, the nodes are stored in an array, node pointers are indices to this array. Nodes that have the same item identifier are stored in a continuous part of this array without storing the item identifiers in the node. As the trie only traversed from the bottom to the top and never searched, each node contains a counter and a pointer to the parent and cut out the child maps in nonordfp. As a result, the nonordfp is a more compact data structure than FP-tree.

**Incremental mining:** There are also lots of algorithms that focus to improve the adaptiveness of the tree structure in dynamical databases. CATS tree [12] can insert and delete the transactions efficiently when the database dynamically changed. It can also mine the frequent patterns with different supports without rebuilding the tree structure. But every insertion or deletion of the transaction, CATS tree need to rearrange the order of the nodes in the tree. AFPIM [20], EFPIM [24], FUFP-tree [17], and CP-tree [42] algorithms perform incremental mining mainly by adjusting the tree structure. But these approaches require two database scans for both the original and incremented portions. CanTree [23] addresses the drawbacks and requires only one scan of the original database and incremented portions. CanTree keeps information of the whole transaction database according to some canonical order. Thus, it is easy to update the CanTree structure for incremental mining. BIT [31] algorithm merge two small consecutive duration FP-trees to form a complete FP-tree structure. But this approach is very complicated and lead to process overhead.

However, the design of these algorithms do not consider the characteristics of the emerging NVMs. Due to the attractive property of persistent, there are many studies aim at rethinking database algorithms design for NVM recent years. Chen et al. [11] propose to improve the performance for two common database algorithms, $B^+$-tree and hash joins, while they are applied in PCM. They employ unsorted leaf nodes to reduce memory writes caused by sorting nodes. To address the issues in [11], Chi et al. propose three different schemes, sub-balanced unsorted node scheme, overflow node scheme, and merging factor scheme to efficiently reduce the write accesses to PCM [13]. Choi et al. propose $PB^+$-tree [14] by separating a node to two area and making the size of one area smaller than the other to reduce sort operations. They also find a way to dynamically change the location of the counter to reduce the write operations. These techniques make the database algorithms on NVM more friendly.

However, with our best knowledge, there is still no work

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems




(a) The *T10I4D100K* dataset ($\delta = 0.01$).

(b) The *T40I10D100K* dataset ($\delta = 0.1$).

(c) The *chess* dataset ($\delta = 0.1$).

(d) The *mushroom* dataset ($\delta = 0.1$).

(e) The *pumsb*$^*$ dataset ($\delta = 0.1$).

(f) The *connect* dataset ($\delta = 0.1$).

(g) The *pumsb* dataset ($\delta = 0.1$).

(h) The *accidents* dataset ($\delta = 0.2$).

(i) The *C73D10* dataset ($\delta = 0.1$).

(j) The *C20D10* dataset ($\delta = 0$).

Fig. 9: The standard deviation of bit write counts of each node in the FP-tree with all datasets. The results for classical FP-tree (FP-construct) and EvFP-tree are shown in the left and right parts of each subfigure, respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2017.2681077, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

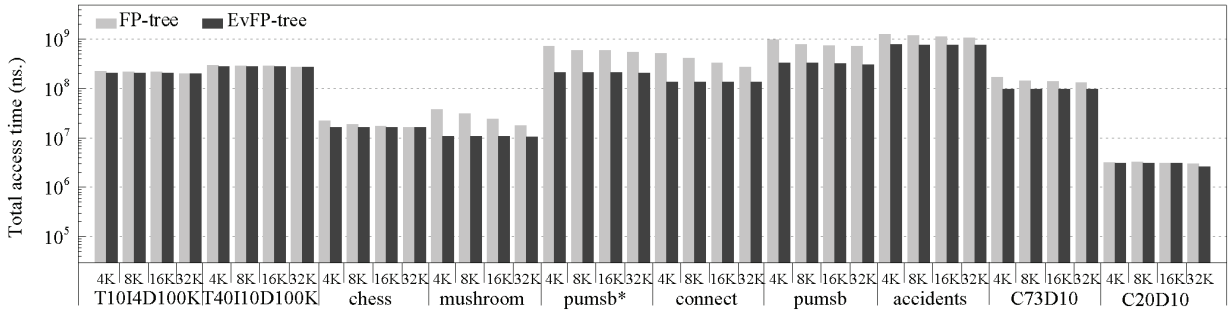



Fig. 10: Total mining time (in ns.) to mine each dataset.

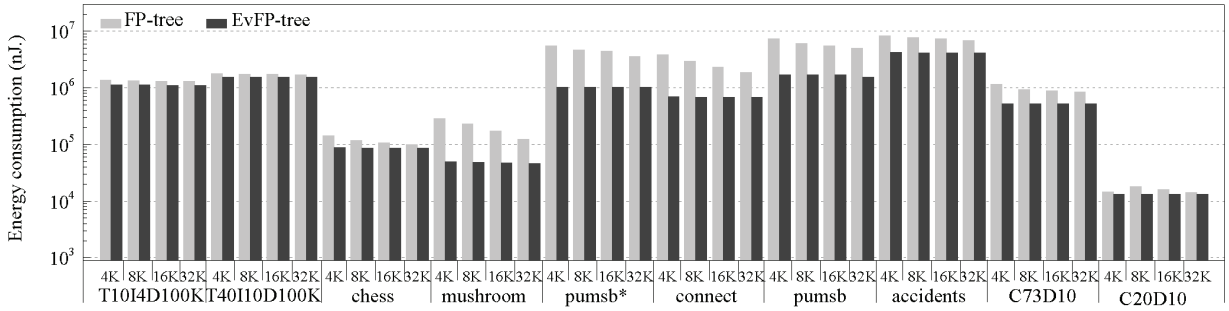



Fig. 11: PCM energy consumption (in nJ.) to mine each dataset.

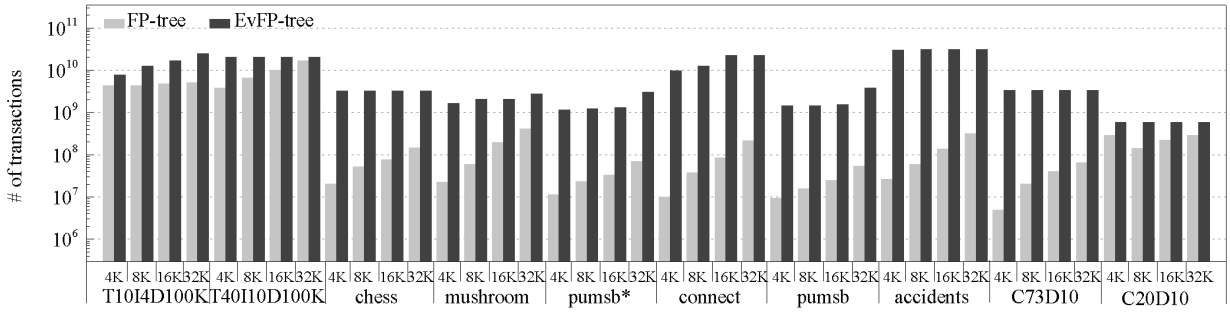



Fig. 12: PCM lifetime in terms of the number of transactions sustained before a PCM cell wears out.

that aims to reduce the memory writes when the FP-tree is constructed over NVMs. As a write operation could take much more time and energy than a read one on NVMs, it is necessary to augment the existing frequent-pattern mining approaches to reduce write operations, so as to reduce the overall time and energy of frequent-pattern mining. Based on these observations, we propose the lazy counter to reduce the memory writes of FP-tree, and the MBA encoding to make the writes on NVMs evenly. Compared with previous work, the strength of this work is that we integrate the frequent pattern mining and NVM together to utilize the attractive property of persistent of NVM. In addition, this work efficiently realizes the algorithmic optimization.

## VI. Conclusions

In this paper, we have proposed *evergreen FP-tree (EvFP-tree)*, a durable, efficient, and **nonvolatile memory (NVM) friendly technique of the well-known frequent-pattern tree (FP-tree)**. In EvFP-tree, we have proposed a *lazy counter* with a two-phase construction strategy to reduce NVM writes to enhance the performance and energy efficiency in the construction of an FP-tree. Such an idea to exploit the bulk information of multiple inputs of an offline algorithm could also be extended for the performance optimization of other online algorithms. Moreover, we have proposed a *minimal-bit-altered (MBA)* encoding scheme to equalize the wearing of different bits in the same support counter of FP-tree nodes to extend the NVM lifetime. The MBA scheme could effectively accompany existing wear-leveling techniques that equalize the accesses across different NVM words. As verified by the experimental results, EvFP-tree outperforms classical FP-growth algorithm by 40.28% on average in terms of the mining performance, and extends the NVM lifetime by 87.20% on average. And EvFP-tree reduces the energy consumption by 50.30% on average. In the future, we shall extend the proposed strategies to flash memory to enable high-performance data

analytics over NVMs.

## REFERENCES


[1] Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data/. Accessed: 2015-03-01.

[2] UCI machine learning repository. http://archive.ics.uci.edu/ml/. Accessed: 2015-03-01.

[3] Nicolas pasquier universit nice sophia-antipolis: Benchmark datasets. http://www.i3s.unice.fr/ pasquier/web/. Accessed: 2016-01-15.

[4] R. Agarwal and R. Srikant. Fast algorithms for mining association rules in large database. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, volume 1215, pages 487–499, 1994.

[5] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD'99)*, pages 254–260, 1999.

[6] B.-J. Chang, Y.-H. Chang, H.-S. Chang, T.-W. Kuo, and H.-P. Li. A PCM translation layer for integrated memory and storage management. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'14)*, pages 6:1–6:10, 2014.

[7] M. F. Chang, L. Y. Huang, W. Z. Lin, Y. N. Chiang, C. C. Kuo, C. H. Chuang, K. H. Yang, H. J. Tsai, T. F. Chen, and S. S. Sheu. A ReRAM-Based 4T2R Nonvolatile TCAM Using RC-Filtered Stress-Decoupled Scheme for Frequent-OFF Instant-ON Search Engines Used in IoT and Big-Data Processing. *IEEE Journal of Solid-State Circuits JSSC16*, 51:1–13, 2016.

[8] M. F. Chang, C. C. Lin, A. Lee, and C. C. Kuo. 17.5 A 3T1R nonvolatile TCAM using MLC ReRAM with sub-1ns search time. In *IEEE International Solid-State Circuits Conference ISSCC15*, pages 1–3, 2015.

[9] A. Chatzistergiou, M. Cintra, and S. D. Viglas. Rewind: Recovery write-ahead system for in-memory non-volatile data-structures. *Proceedings of the VLDB Endowment (VLDB'15)*, 8(5):497–508, 2015.

[10] C.-H. Chen, P.-C. Hsiu, T.-W. Kuo, C.-L. Yang, and C.-Y. Wang. Age-based PCM wear leveling with nearly zero search cost. In *Proceedings of the 49th ACM/IEEE Design Automation Conference (DAC'12)*, pages 453–458, June 2012.

[11] S. Chen, P. B. Gibbons, and S. Nath. Rethinking database algorithms for phase change memory. In *Proceedings of the Fifth Biennial Conference on Innovative Data Systems Research (CIDR'11)*, pages 21–31, 2011.

[12] W. Cheung and O. Zaiane. Incremental mining of frequent patterns without candidate generation or support constraint. In *Proceedings of the 7th IEEE International Database Engineering & Applications Symposium (IDEAS'03)*, pages 111–116, July 2003.

[13] P. Chi, W. C. Lee, and Y. Xie. Making B+-tree efficient in PCM-based main memory. In *International Symposium on Low Power Electronics and Design (ISLPED'14)*, pages 69–74, 2014.

[14] G. S. Choi, B. W. On, and I. Lee. PB+-tree: PCM-aware B+-tree. *IEEE Transactions on Knowledge & Data Engineering (TKDE'15)*, 27(9):1–1, 2015.

[15] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling of high-frequency accident locations by use association rules. *Transportation Research Record: Journal of the Transportation Research Board*, (1840):123–130, 2003.

[16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*, 29(2):1–12, May 2000.

[17] T. P. Hong, J. W. Lin, and Y. L. Wu. Maintenance of fast updated frequent pattern trees for record modification. In *Proceedings of the First IEEE International Conference on Innovative Computing, Information and Control (ICICIC'06)*, pages 570–573, 2006.

[18] J. Hu, Q. Zhuge, C. J. Xue, W.-C. Tseng, and E. H.-M. Sha. Software enabled wear-leveling for hybrid pcm main memory on embedded systems. In *Proceedings of the 2013 IEEE Conference on Design, Automation & Test in Europe (DATE'13)*, pages 599–602, 2013.

[19] W. S. Khwa, M. F. Chang, J. Y. Wu, and M. H. Lee. 7.3 A resistance-drift compensation scheme to reduce MLC PCM raw BER by over 100x for storage-class memory applications. In *IEEE International Solid-State Circuits Conference ISSCC16*, 2016.

[20] J. L. Koh and S. F. Shieh. An efficient approach for maintaining association rules based on adjusting FP-Tree structures. *Springer Berlin Heidelberg*, pages 417–424, 2004.

[21] C.-H. Lai, S.-C. Yu, C.-L. Yang, and H.-P. Li. Fine-grained write scheduling for pcm performance improvement under write power budget. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'15)*, pages 19–24, July 2015.

[22] M. Lee, D. H. Kang, J. Kim, and Y. I. Eom. M-CLOCK: Migration-optimized page replacement algorithm for hybrid dram and pcm memory architecture. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC'15)*, pages 2001–2006, 2015.

[23] C. K. S. Leung, Q. I. Khan, Z. Li, and T. Hoque. CanTree: a canonical-order tree for incremental frequent-pattern mining. *Knowledge and Information Systems*, 11(3):287–311, 2007.

[24] X. Li, Z. H. Deng, and S. Tang. A fast algorithm for maintenance of association rules in incremental databases. *Lecture Notes in Computer Science (LNCS'06)*, pages 56–63, 2006.

[25] D. Liu, T. Wang, Y. Wang, Z. Qin, and Z. Shao. PCM-FTL: A write-activity-aware NAND Flash memory management scheme for PCM-based embedded systems. In *Proceedings of 32nd IEEE Real-time Systems Symposium (RTSS'11)*, pages 357–366, 2011.

[26] D. Liu, T. Wang, Y. Wang, and Z. Shao. Application-specific wear leveling for extending lifetime of phase change memory in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD'14)*, 33(10):1450–1462, 2014.

[27] Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[28] C. Xu et al. Understanding the trade-offs in multi-level cell ReRAM memory design. In *Proceedings of the 50th Annual ACM/EDAC/IEEE Design Automation Conference (DAC'13)*, pages 1–6, May 2013.

[29] H. Zhang et al. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE'15)*, 27(7):1920–1948, Jul. 2015.

[30] M. Rafiquzzaman. *Fundamentals of digital logic and microcomputer design*. Wiley, 2005.

[31] S. G. Totad et al. Batch processing for incremental FP-tree construction. *International Journal of Computer Vision (IJCV'10)*, (5):28–32, 2010.

[32] S. Motaman, S. Ghosh, and N. Rathi. Impact of process-variations in sttram and adaptive boosting for robustness. In *Proceedings of the 2015 Design Automation and Test in Europe (DATE'15)*, pages 1431–1436, 2015.

[33] J. op den Brouw. An introduction to ring counters. *Technical Report*, pages 1–10, May 2015.

[34] J. Park, D. Shin, and H. G. Lee. Prefetch-based dynamic row buffer management for lpddr2-nvm devices. In *Proceedings of the 2015 IEEE Conference on Very Large Scale Integration (VLSI-SoC'15)*, pages 98–103. IEEE, 2015.

[35] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*, pages 14–23, 2009.

[36] B. Rcz. nonordfp: An FP-growth variation without rebuilding the FP-Tree. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, 2010.

[37] R. Saikkonen. Bulk updates and cache sensitivity in search trees. *PhD Dissertation at Helsinki University of Technology*, 2009.

[38] D. Schwalb, M. Dreseler, M. Uflacker, and H. Plattner. NVC-hashmap: A persistent and concurrent hashmap for non-volatile memories. In *Proceedings of the 3rd VLDB Workshop on In-Memory Data Mangement and Analytics (IMDM'15)*, page 4. ACM, 2015.

[39] S. Senni, R. M. Brum, L. Torres, and G. Sassatelli. Potential applications based on nvm emerging technologies. In *Proceedings of the 2015 IEEE Conference on Design, Automation & Test in Europe (DATE'15)*, pages 1012–1017, 2015.

[40] Z. Shao, Y. Liu, Y. Chen, and T. Li. Utilizing PCM for energy optimization in embedded systems. In *Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'12)*, pages 398–403, 2012.

[41] Y. G. Sucahyo and R. P. Gopalan. CT-PRO: A bottom-up non recursive frequent itemset mining algorithm using compressed fp-tree data structure. *FIMI*, 2004.

[42] S. K. Tanbeer, C. F. Ahmed, and B. S. J. Y. K. Lee. CP-Tree: A tree structure for single-pass frequent pattern mining. In *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining (PAKDD'08)*, pages 1022–1027, 2008.

[43] J. Wang, X. Dong, and Y. Xie. Building and optimizing MRAM-based commodity memories. *ACM Transactions on Architecture and Code Optimization (TACO'14)*, 11(4):36, 2014.

[44] T. Wang, D. Liu, Y. Wang, and Z. Shao. Towards write-activity-aware page table management for non-volatile main memories. *ACM Transactions on Embedded Computing Systems (TECS'15)*, 14(2):1–23, 2015.

[45] Y. Wang, H. Wang, and Z.-G. Gu. A survey of data mining softwares used for real projects. In *Proceedings of the 2011 IEEE International Workshop on Open-Source Software for Scientific Computation (OSSC'11)*, pages 94–97, Oct 2011.

[46] 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 *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07)*, pages 3014–3017, May 2007.

[47] J. Yue and Y. Zhu. Accelerating write by exploiting PCM asymmetries. In *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA'13)*, pages 282–293, 2013.