A Frequent-Value Based PRAM Memory Architecture

Guangyu Sun, Dimin Niu, Jin Ouyang, Yuan Xie
CSE,Pennsylvania State University
{gsun, dun118, jouyang, yuanxie}@cse.psu.edu

***Abstract*—Phase Change Random Access Memory (PRAM) has great potential as the replacement of DRAM as main memory, due to its advantages of high density, non-volatility, fast read speed, and excellent scalability. However, poor endurance and high write energy appear to be the challenges to be tackled before PRAM can be adopted as main memory. In order to mitigate these limitations, prior research focuses on reducing write intensity at the bit level. In this work, we study the data pattern of memory write operations, and explore the frequent-value locality in data written back to main memory. Based on the fact that many data are written to memory repeatedly, an architecture of frequent-value storage is proposed for PRAM memory. It can significantly reduce the write intensity to PRAM memory so that the lifetime is improved and the write energy is reduced. The trade-off between endurance and capacity of PRAM memory is explored for different configurations. After using the frequent-value storage, the endurance of PRAM is improved to about $1.6X$ on average, and the write energy is reduced by 20%.**

I. Introduction

The emerging phase change random access memory (PRAM) has recently become a promising candidate for future replacement of DRAM-based main memory , due to its distinctive advantages of high density, fast read speed, low leakage, non-volatility, and excellent scalability [1]. Recently, there has been extensive research of using PRAM as the replacement of traditional memories at various level of memory hierarchies, such as last level caches [2], [3] and main memory [1], [4]–[10].

Compared to DRAM-based main memory, PRAM has a higher density and a comparable read speed. Some well-known limitations of PRAM, however, become the obstacles of using PRAM-based main memory (PRAM memory). First, the write cycle of PRAM (or lifetime endurance) is in the range of 10^8 with different process technologies, and it is not sufficient under frequent accesses to main memory [3], [6], [7]. Second, the write energy to a PRAM cell is much larger than that to a DRAM cell, due to PRAM's nature of current-driven storage [10].

Extensive work has been done at the architectural level to mitigate the limitations of PRAM memory. The research of improving endurance can be classified into two categories: reducing write intensity and wear-leveling. In order to reduce the write intensity, Zhou *et al* proposed the method of "data-comparison write (DCW)" to avoid writing redundant bits, values of which are not changed in the write operation [6]; the "Flip-N-Write" method was proposed to increase the number of such redundant bits to improve the efficiency of DCW method [5]. In addition, different wear-leveling policies are presented to even out the write intensity among all PRAM cells so that the endurance of PRAM memory is increased. [3], [4], [6], [7], [9]. Xu *et al* also pointed out that the data-comparison write can reduce the write energy of PRAM [10].

The prior work is mainly based on the "bit-level" temporal locality. The value of each bit is compared between old data and new data to find the redundant bits. The benefits come from the speculation that a bit value may be kept unchanged during successive write operations. However, in some cases, these methods may not work efficiently because it ignores the pattern of data. For example, assume the data length is 8-bit. Two data , "11110000" and "11111111", are written into the same memory space and evict each other repeatedly. During each write operation, four bits have to be changed, even when an extra status bit is induced to apply "Flip-N-Write" policy [5]. Although there are only two different data involved, the bit-level temporal locality cannot be found in the last four bits. The "frequent-value locality" of data [11], however, exists because these two data are repeatedly written. If the data level locality is identified and utilized, the write intensity to memory can be reduced.

In this paper, the frequent-value locality is explored in the data that are written to PRAM memory. A frequent-value based data storage architecture is proposed for PRAM memory. Static and dynamic profiling methods are presented to identify frequent values for different applications. With this architecture, the write intensity to PRAM memory is studied at the "data-level" instead of at the bit-level. Through exploring the frequent-value locality, the data, which are frequently written back to PRAM memory, are stored with a compressed (encoded) form. Consequently, the write intensity to PRAM memory is significantly reduced. In addition, since such approach is achieved at the data-level, the frequent-value based storage can be used in parallel with those bit-level methods seamlessly to further improve the endurance of PRAM memory.

This work was supported in part by NSF grants (0702617, 0903432, 0905365, 0916887) and SRC grant.

II. Background

In this section, a brief introduction of the PRAM technology and its limitations is presented. In addition, the definition of frequent-value locality and related work are also introduced.

A. PRAM Technology

Different from the conventional RAM technologies (such as SRAM/DRAM), the information carrier of PRAM is chalcogenide-based materials, such as $Ge_2Sb_2Te_5$ and $Ge_2Sb_2Te_4$ [12]. The crystalline and amorphous states of chalcogenide materials have a wide range of resistivity, about three orders of magnitude, and this forms the basis of data storage. The amorphous, high resistance state is used to represent a bit '0', and the crystalline, low resistance state represents a bit '1'.

Nearly all prototype devices make use of a chalcogenide alloy of germanium, antimony and tellurium (GeSbTe) called GST. When GST is heated to a high temperature (normally over $600\,^{\circ}\mathrm{C}$), it will get melted and its chalcogenide crystallinity is lost. Once cooled, it is frozen into an amorphous and its electrical resistance becomes high. This process is called RESET. One way to achieve the crystalline state is by applying a lower constant-amplitude current pulse for a time longer than the so-called RESET pulse. This is called SET process [13]. The time of phase transition is temperature-dependent. Normally, it takes tens of nanoseconds for the RESET and more than 100ns for the SET of each PRAM cell [13], [14].

Compared to DRAM memory cells, a PRAM cell has a comparable read latency and higher density. The write latency of a PRAM cell,

978-1-4244-7516-2/11/$26.00 ©2011 IEEE

however, is much longer than that of the DRAM cell. In addition, the write cycles (the number of writes in lifetime, or endurance) of a PRAM cell are much lower than that of a DRAM cell (and therefore a shorter lifetime).

B. Related Work of Frequent-value locality

The concept of frequent-value locality was first proposed by Zhou *et al* [11]. This work pointed out that, at any given point in the program execution, several distinct values occupy a vast fraction of the values in memory accesses. These distinct values are named as frequent values. For some applications, the top three frequently accessed values can occupy more than 50% of total accesses. Based on this locality, a frequent-value cache (FVC) is proposed. Since only frequent values are kept in FVC, the data in FVC are stored in a compact form. Consequently, FVC can help reduce the cache miss rates efficiently with low overhead.

As frequent-value locality provides the opportunity of storing data in a compact form, it is normally used for data compression [15], [16]. Yang *et al* also leveraged the frequent-value locality to reduce the access energy of caches [17]. In their work, frequent values were read and written in an encoded form, the length of which was less than that the original data. Therefore, the dynamic energy consumption is reduced because less bits are activated in the read/write operations.

III. Frequent-value Based PRAM Memory

In this section, the frequent-value locality is proved to exist in the data written to memory. Based on this observation, the traditional memory architecture is modified to supply the frequent-value based storage in PRAM memory.

A. Frequent-value locality in data writes

The following terms are defined for the discussion:

- **FV Length:** the bit length of the frequent value.
- **Encoded data:** the data that are identified as frequent values and are stored as encoded form in PRAM memory.
- **Original Data:** the data that are not identified as frequent values.
- **FV Number:** the total number of frequent values.
- **FV Ratio:** $\frac{total\ number\ of\ frequent\ values}{total\ number\ of\ data\ written\ to\ PRAM\ memory}$

In order to identify the frequent-value locality in data written to PRAM memory, diverse applications from SPEC and PARSEC benchmark suites are well studied by using the simulation tool SIMICS. For all benchmarks, the data written back from last level caches are tracked so that the frequently written values can be identified. The results show that more than half of benchmarks exhibit high levels of frequent value locality, indicating that the frequent-value locality exists in the data written to PRAM memory. On average, about 30% of data written to PRAM memory involve only eight distinct values, and for some applications, 40% of writes consist of only eight distinct values.

Although frequent-value locality exits in data written to memory, there are several issues that should be resolved before applying the frequent-value based storage to PRAM memory: (1) The frequent values need to be stored in a compressed pattern in order to reduce the write intensity to PRAM memory. (2) Although the frequent values are only compressed during write operations, the encoded values need to be identified in read operations. (3) Data compression/decompression may decrease the performance of write/read operation. The architecture need to be adapted to mitigate the impacts. (4) The frequent values and other parameters, such as FV length and FV number, should be chosen carefully because they may have impacts on the lifetime and performance of PRAM memory. (5) The storage of compressed data induces space overhead in PRAM memory. The trade-off between capacity and lifetime of PRAM memory need to be explored. All of these issues will be discussed in the following subsections and techniques will be proposed to mitigate the overhead.

B. Architecture for Frequent-value Storage

The traditional memory structure need to be modified in order to apply the frequent-value based storage efficiently. Figure 1 illustrates a row of memory array in the modified architecture. The memory row is divided into several data blocks based on the FV length. As shown in the lower part of the figure, a data block is composed of data bits and an extra bit called frequent-value-bit (FV-Bit). The data bits in a data block can be used to store either the original data or the encoded data. The FV-bit identifies whether the original data or the encoded data are stored in a data block. When the original data are stored in the data block, the FV-bit is set to bit '0', and all bits are required to represent a valid data, as in the traditional memory. On the contrary, if a frequent value is identified and stored in the data block, only $log_2(FV\ number)$ bits are used to store the encoded data, and the FV-bit is set to bit '1'. It means that only $log_2(FV\ number)$ bits need to be updated during the storage of a frequent value, and therefore the write intensity is greatly reduced.

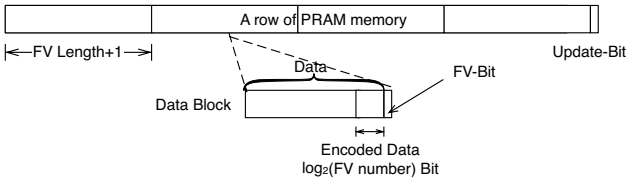

Fig. 1. The structure of a row of PRAM memory.

In addition to FV-Bit inside each data block, an extra status bit, named *Update-Bit*, is also added into each row of PRAM array. The Update-Bit is set to bit '0' when the data is first load from the secondary storage, such as HDD; it is set to bit '1' when data are written back from the last level cache to the row of PRAM array. With the help this Update-bit, the read-only PRAM memory rows are differentiated from those storing updated data. Consequently, read-only PRAM memory can be accessed directly without being involved in the process of dealing with frequent values. The detailed usage of this bit will be shown later in this section.

The whole structure view and corresponding data flow of the PRAM memory are shown in Figure 2. The structure includes a memory array, which is composed of PRAM memory rows, and the peripheral circuitry used for read and write operations. In order to simplify the illustration, we hide some conventional components, such as address decoders, sense amplifiers, etc., but emphasize the components managing the frequent values.

The circuitry below the PRAM memory array is responsible for read operations. In read operation, the data block is the basic unit of the process. Each data blocks is read out and managed individually from a PRAM memory row. Since either the original data or the compressed data can be stored in a data block, the valid data should be identified before being loaded into the memory buffer. As shown in Figure 2, a decoder (DEC) and a frequent-value-table (FV-Table) are used to translate encoded data to the original data.

The FV-Table records all frequent values, which can be identified in the PRAM memory. The input of the decoder is composed of the $log_2(FV\ number)$ bits used for encoded data. The output of the decoder is used to select the valid data stored in the FV-Table. In

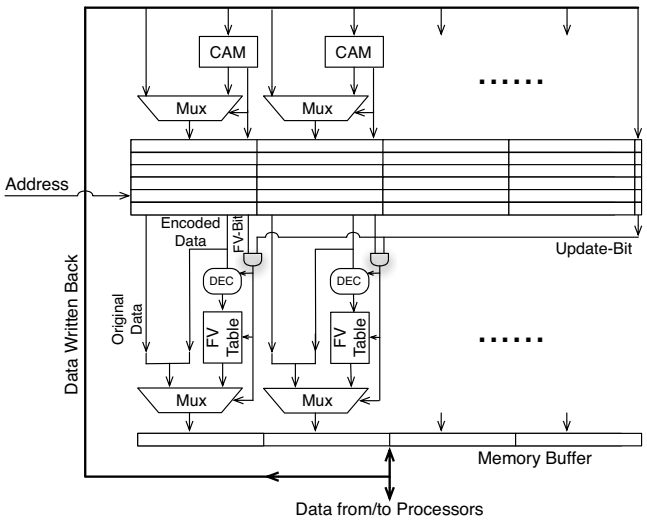

Fig. 2. The structure of the PRAM memory including control and data flow.

order to minimize the timing overhead of read operations, all bits of the data block are read out as original data at the same time when encoded bits are processed. The original data and the output of FV table are fed into a multiplexer (MUX), which is controlled by FV-Bit and Update-bit shown in Figure 1. Only if both FV-Bit and Update-Bit equal to '1', the decoded frequent value is output from the multiplexer and written into the memory buffer. Otherwise, the original data of the data block are written into the memory buffer.

In order to reduce the energy overhead, FV-Bit and Update-Bit are combined as a signal to make both the decoder and FV table work selectively. Selective working controlled by a signal has already been used in prior research [18]. The value of this signal is just the same as the one used to control the multiplexer, which is shown as the output of the *AND* gate in Figure 2. If a row of PRAM memory is never updated or there are no frequent values stored in the data block, this signal can disable the decoder and FV table.

The write operation is shown as the flow above the memory array in Figure 2. When the data are evicted from the memory buffer and written back to a row of PRAM memory, the data are searched to identify whether they consist of any frequent values. As shown in the figure, the content addressable memory (CAM) is employed to record all frequent values and the corresponding encoded bits. If the data equals to any frequent value in CAM, only $log_2(FVnumber)$ encoded bits are written back to the data block, and both the FV-Bit of the data block and the Update-Bit of the memory row are set to '1'. Otherwise, the original data is written to the data block and the FV-Bit is set to '0'. Note that the CAM and FV-table can be combined together because they record the same frequent values. They are separated in the figure in order to clearly illustrate the read and write operations.

C. Structure Parameters

The most important parameter is the length of a frequent value because it determines the FV ratios and storage overhead in PRAM memory. As the length of a frequent value decreases, the number of data blocks increases. The storage overhead increases because an FV-Bit is induced in each data block. On the other hand, the FV-Ratio is increased when we use smaller size of data block. It is because the frequent value with shorter FV length has a higher probability to be written repeatedly. The impact of data block size, however, varies for different applications, which have different patterns of data.

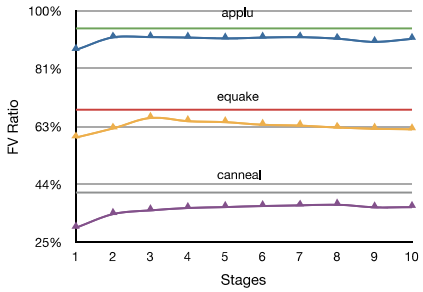

Fig. 3. FV ratios using FVs of 10 stages.

The write intensity, however, may also be increased if the FV length is too small. Although FV-ratio increases with a smaller FV length, more encoded bits may be written to memory because the number of data blocks in a memory row is increased. For example, when FV length is reduced by half, a frequent value is divided into two parts. It is possible that both of these two parts are identified as new frequent values. Consequently, we need $2 \times log_2(FVnumber)$ to store the same data in PRAM memory. When FV length is too small, the increase of encoded bits may offset the benefits from increasing FV-ratio, and total write intensity to PRAM memory may be increased. In addition, the hardware overhead is increased as the FV length is reduced. As shown in Figure 2, the access to each data block is in parallel so that the impact on performance is minimized. Consequently, the number of duplicated peripheral circuitry equals to the number of data block in a PRAM memory line, which is decided by FV length.

FV number also has an impact on performance of the memory architecture. Apparently, a higher FV-ratio can be achieved with a larger FV table because more frequent values can be identified. However, the capacities of FV table and CAM are increased with the FV number. Consequently, the hardware overhead is increased, and it takes more time to decode those encoded $log_2(FV\ number)$ bits and to search the CAM and FV table. It should be mentioned that the benefits of increasing FV number is related to the FV length. For some applications, where the frequent-value locality is very low for a large FV length, the FV-ratio may not be increased much with a large FV number.

IV. Profiling and Management of Frequent Values

In this section, two methods of generating frequent values for different cases are introduced. Also, the methods to manage these frequent values for different configurations are discussed.

A. Static Profiling

The static profiling means that the fixed frequent values are profiled on individual application and are generated before running the application. This method is similar to the approach of "*Find Once for a Given Program*" in prior work [19]. This is a software based approach. Since the profiling is finished in advance, it will not cause run-time profiling overhead. In addition, the frequent values are found based on the profiling of the whole program so that the FV ratio can be kept in a high level on average. This method, however, also has its own limitations. It requires the support from compilers, and the profiling of the whole program is time consuming. These frequent values need to be included in the executable code of each application. More importantly, the frequent values can be different with various input data of applications. Therefore, the static profiling is suitable for cases that applications are executed repeatedly with similar input.

B. Dynamic Profiling

Different from the static profiling method, the dynamic profiling method generates frequent values during the run-time execution. It can be achieved in hardware without requiring support from compilers. The run-time profiling overhead, however, may be induced. Some run-time profiling methods are introduced in prior work [19] for on-chip caches or data transaction. In these methods, old frequent values are continuously evicted from the FV table and new values are inserted to the table during the execution. Whenever a frequent value is evicted, a refreshing operation should be called to ensure the consistency in the memory. A refreshing operation is to flush the corresponding frequent value stored in the memory and to restored the original date. However, these methods is not suitable for the PRAM memory because the continuous refreshing operation will cause significant degradation of performance. More importantly, the refreshing operation will induce extra write intensity and therefore reduce the lifetime of the PRAM memory. In order to solve these problems, a novel dynamic profiling technology is proposed for PRAM memory, which generates the frequent value incrementally without causing any refreshing.

The basic idea of this method is to profile the frequent values from the beginning of each program and fill the FV table incrementally during the execution. It is inspired by the observation that *the top frequent values of written data do not vary much during the execution of the application.* This characteristic of frequent-value locality can be proved from results in Figure 3. In the experiments, each application is divided into 10 different stages based on the execution time. The top 32 frequent values of each stage are profiled statically. Then, each application is executed for 10 rounds. In each round, the top 32 frequent values profiled in a different stage is stored in FV table to work as the frequent values for the whole application. In other words, we examine the efficiency of using frequent values of different stages. The FV ratio of each round is shown in the Figure 3. Note that the straight line is the result of FV ratio of using static profiling. The results show that most of frequent values are common for different stages. Another interesting observation is that *zero is always one of top frequent values for all applications.*

Based on these observations, we propose the method of *global incremental profiling*, which is described as follows:

- The FV table is initialized with only one entry of frequent value, which equals to zero.
- A write access counting table (WAC table) is used to profile the run-time frequent values.
- Each time a data block is written to PRAM memory, the WAC table is updated using the swapping method [19]: If the value in data block is already present in entry i, then the counter at entry i is incremented by one; When the counter saturates, the entry i is swapped with entry $i-1$ values; When a new value is encountered, and there is no free entry, an entry is freed from the bottom of the table.
- After a profiling period P_r, the WAC table is compared with the FV table. The value of the first entry in the WAC table that is not present in current FV table is inserted into FV table as a new frequent value. The CAM is updated at the same time.
- When the FV table is full, the values in FV table are fixed and used for frequent storage in the rest of execution. The dynamic profiling is disabled by clock gating.

It has been proved that such a swapping method can approximately profile the top frequent values in WAC table without sorting the total access numbers of these values [19]. The WAC table has no impact on the performance because it is not in the critical path. The dynamic profiling quality of WAC is related to the profiling period P_r. This is further discussed in Section VI. Compared to the static profiling, the global incremental profiling is suitable for the case of running applications with different input data.

C. Wear leveling of Frequent Values

Wear leveling is still required even when we apply the frequent storage techniques to reduce the write intensity. As shown in Figure 1, $log_2(FV\ number)$ bits are used to store the encoded frequent values. These bits wear faster than the rest in a data block, especially for applications with high frequent locality. Thus, we shift the positions of $log_2(FV\ number)$ bits, which store the encoded frequent values, inside the data block. Note that this shift operation is different from that in prior work [6], because only $log_2(FV\ number)$ bits are shifted.

The FV-bit and Update-bit in the data block, however, are not shifted. After the data is loaded into the PRAM memory line, the Update-bit is, at most, changed once before data is evicted. The FV-bit changes when the value stored in a data block is changed between original data form and the encoded form. The experimental results show that this bit wear our much slower than data cells in the data block. Therefore, the position of these bits are fixed because they will not wear out faster than the data cells.

V. Complementing Existing Techniques

As we have addressed, our technique of frequent-value storage is to reduce the write intensity to PRAM memory by exploring the locality at data-level. The approaches proposed in prior research focus on the bit-level. It means that the frequent-value storage can be easily integrated with these bit-level techniques to further improve the lifetime of PRAM memory. For example, the DCW technique can be applied with frequent value storage by comparing each encoded bit before writing. For "Flip-N-Write" technique, the only difference is that the hamming distance is just calculated for $log_2(FV\ number)$ bits if a frequent value is written. In addition, the frequent value storage can help reduce the overhead of using these bit-level techniques. For the read operation before write, when an encoded frequent value is identified, only $log_2(FV\ number)$ bits rather than the whole data block need to be read out for comparison.

Similarly, the wear leveling techniques in prior work can also be applied with frequent value storage. Note that, if we apply the shift of frequent values introduced in the previous section, we don't need the byte-level shift operation in prior work [6]. Since a lot of written data can be identified as frequent values, an approximately uniform write intensity can be achieved with the shift of encoded bits for frequent values.

VI. Evaluations

In this section, comprehensive experimental results are provided and the structure parameters are discussed.

A. Baseline Configuration

The parameters of baseline configuration are listed in Table I. We use an eight-core in-order CMP as the processor. The second level cache has a capacity of 32MBytes in order to reduce the write intensity to PRAM memory. The size of write buffer before PRAM memory is set to 128 entries. The write buffer is large enough to mitigate the impact of long write latency of PRAM memory. For all the benchmarks, the write buffer is never full so that the read operation will not be blocked for a long time if there is a burst of write operations. The memory controller, however, does not adopt

the technique of "write cancellation" [8] for several reason: (1) The written latency, which varies with the written data, makes it difficult to cancel a write operation; (2) More importantly, the write cancellation aggravate the endurance problem since some write operations are canceled and re-written repeatedly to PRAM memory.

TABLE I
BASELINE CONFIGURATIONS.

Processor	8-core in-order CMP, 1GHz
Caches	D/I L1 caches: 32+32KB, L2 caches: 32MB
PRAM memory	4GB, 32KB memory line, 128-entry write buffer read latency: 60 cycles, write latency: 160 cycles per 16Byte

B. The Evaluation Metrics

Applying frequent-value storage to PRAM memory requires extra bits to record the status of a data block or a row PRAM memory. The effective capacity of PRAM is reduced after using the architecture of frequent-value storage. As we discussed, the effective capacity and lifetime of PRAM vary with different configurations of frequent-value storage. It shows the trade-off between the lifetime and capacity of PRAM memory.

In order to explore the trade-off and fairly evaluate the endurance of PRAM memory, we propose a new metric called "capacity-lifetime product" ($Capacity \times Lifetime$). Note that $Capacity$ represents the effective capacity after using techniques to improve the lifetime of PRAM memory.

C. Evaluation of Frequent-value Locality

Figure 4 lists the results of FV ratios for different FV numbers and FV lengths. The FV number varies from 8 to 128. The circuit level simulations show that the access latency to CAM or FV table of 128 entries can be finished within one cycle. Consequently, we choose 128 as the largest number of frequent values in this work to minimize the overhead on performance. These frequent values are profiled with static profiling method. The results show that the FV ratio increases when there are more frequent values identified in the PRAM memory. The FV ratios of some applications, however, show low sensitivity to FV numbers. For these applications, there is a very high frequent-value locality in the data written to PRAM memory. FV-ratios are very high even when there are only eight frequent values. On the contrary, the FV-ratios of some applications are sensitive to the number of frequent values, such as *equake*, *canneal*, *ferret*, etc. Besides the top eight frequent values, other frequent values are also written to PRAM memory intensively.

As we have discussed, the impact of the FV number also depends on the FV length. For the last two applications (*canneal* & *ferret*) in Figure 4(a)(b)(c), FV-ratios only increase a little, when the FV number is increased to 128. In Figure 4(d), when the length of frequent value is reduced to $64Bit$, the FV ratios of these two applications become very sensitive to FV numbers. It is because the length of a register in the processor exactly equals to $64Bit$. Since the data output of the processor is written in such a granularity, a high frequent-value locality is achieved with a FV length of $64Bit$.

The FV ratios for different FV lengths are shown in Figure 5, with a fixed FV number of 128. For all applications, the FV ratios are increased as FV length decreases. For most applications, increase of the FV ratio is not significant when FV length is reduced from $64Bit$ to $32Bit$. We also find exceptions in the last several applications. We study the written data in these applications and find that, the upper half of the $64Bit$ is kept the same for lots of data from processors. Consequently, the FV ratios are increased much when the FV length is reduced to $32Bit$.

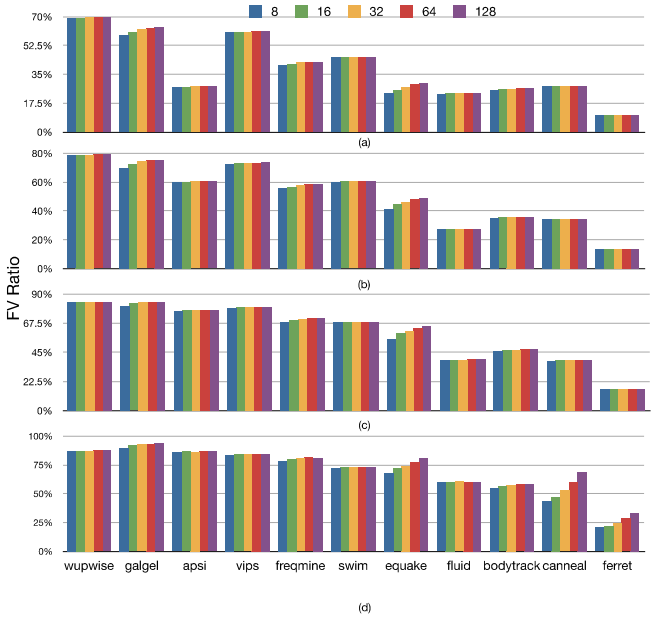

Fig. 4. FV ratios for different FV numbers and FV lengths. FV length= (a) 512Bit; (b) 256Bit; (c) 128Bit; (d) 64Bit.

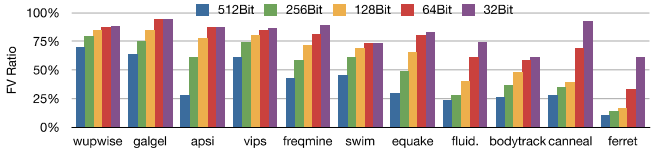

Fig. 5. FV ratios for different FV lengths (FV number=128).

Figure 6 compares the FV ratios with different profiling methods. The label *Static* represents the static profiling method; *Global* represents the proposed global incremental profiling method. The incremental profiling period is set to 2^{13} of write operations. For applications with high frequent-value locality, the FV ratios do not vary much for different profiling methods. For applications with low frequent value locality, the frequent values vary a lot in different stages of the program execution. We can achieve larger FV ratios with the static profiling method. Since the static profiling is not suitable for applications with diverse input, in the rest of this work, only the dynamic profiling is employed.

The results in Figure 6 show that, for most applications, FV ratios of global incremental profiling method are very closed to those of using static profiling. The process of global incremental profiling is finished in the early stage of the program execution. The profiled global frequent values are related to the profiling period, P_r. In Figure 7, we list the results of FV ratios using the global incremental method with different P_rs. For all applications, the FV ratios vary little with P_r. The global frequent values profiled in different stages are similar. Consequently, we can generate similar global frequent values in FV tables with different profiling periods. The FV-ratios, however, decrease a little when the profiling period is too long. The reason is that it takes longer time to fill the FV table with a longer profiling period. Based on these results, we use 2^{13} of write operations as the profiling period.

D. Evaluation of Endurance

The improvement of lifetime is evaluated by iteratively running these applications till a failure happens in one PRAM memory cell. Figure 8 shows the improvement of lifetime when the global

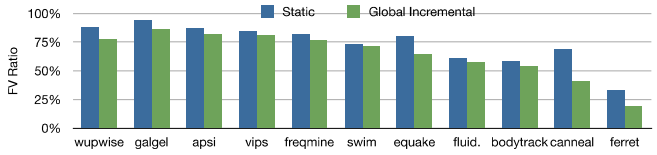

Fig. 6. FV ratios of different profiling methods. (FV number=128, FV length=$64Bit$).

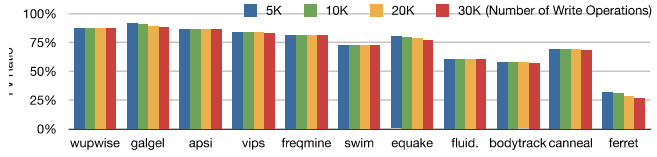

Fig. 7. FV ratios of incremental method with different profiling periods. (FV number=128, FV length=$64Bit$).

incremental profiling is used. For applications with high frequent-value locality, the lifetime is improved by more than 10 times when the length of frequent value is larger than $64Bit$. For these applications, the improvement of lifetime is reduced when FV length decreases. It is because FV ratios are not increased much with shorter frequent values. However, the number of data blocks in a PRAM memory line is increased and more bits are used to store the encoded bit, as we discuss in subsection III-C. On the contrary, for applications with low frequent-value locality, the highest improvement of lifetime happens when we use $64Bit$ or $32Bit$ as the FV length. For these applications, the FV ratios are greatly increased with short frequent values and we have more benefits of storing more frequent values as encoded bits.

In Figure 9, we evaluate the benefits of combining data-level and bit-level techniques together. We compare the results of $Capacity \times Lifetime$ between two cases. In the first case (DCW), only the bit-level technique in prior work is used. In the second case ($DCW + FV$), the frequent value storage is used in together with DCW. The results show that the $Capacity \times Lifetime$ can be improved to about 3X in the best case. On average, the $Capacity \times Lifetime$ of using both techniques is improved to $1.6X$ of that only using only bit-level technique.

E. Evaluation of Write Energy

Figure 10 compare the results of write energy for different applications. The first case is using the baseline PRAM memory without using frequent value storage. The second case is using frequent-value storage with global incremental profiling. The P_r is set to 2^{13}. The FV length and FV number are set to $128Bit$ and 64, respectively. The results show that, on average, the write energy is reduced to 80% of the baseline after using the frequent-value storage.

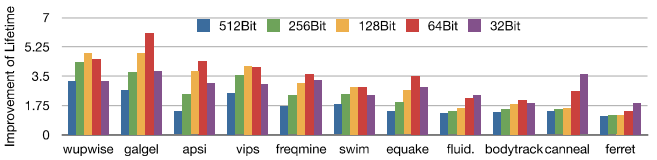

Fig. 8. Lifetime improvement with the global incremental profiling.

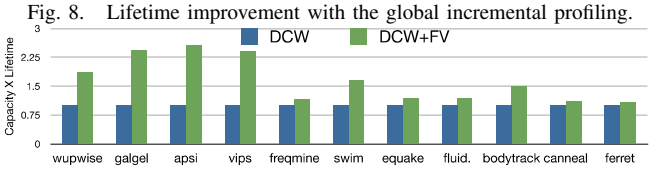

Fig. 9. Lifetime improvement of the combined technology.

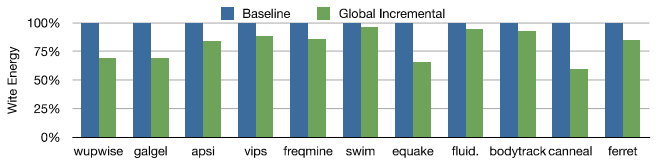

Fig. 10. Reduction of write energy.

VII. Conclusion

The emerging Phase-Change Random Access Memory (PRAM) is a promising candidate as the replacement for DRAM-based main memory, with benefits of high density, fast access, zero standby leakage, and non-volatility. However, PRAM also has limitations such as poor endurance, latency and energy overhead associated with write operations. In this paper, we study the data pattern of memory write operations, and explore the frequent-value locality of PRAM memory. An architecture of frequent-value storage is proposed for PRAM memory based on the data locality. This approach can significantly reduce the write intensity to PRAM memory so that both lifetime and performance can be improved. Also a novel dynamic profiling technology is presented for this architecture. The trade-off between endurance and capacity of PRAM memory is explored for different configurations. After using the frequent-value storage architecture, the endurance of PRAM is improved to about $4X$ on average, and the write energy is reduced by 27%.

References

[1] B. C. Lee and *et al.* Architecting phase change memory as a scalable dram alternative. In *Proceedings of ISCA 2009*, pages 2–13.
[2] X. Wu and *et al.* Hybrid cache architecture with disparate memory technologies. In *Proceedings of ISCA 2009*, pages 34–45.
[3] Y. Joo and *et al.* Energy- and endurance-aware design of phase change memory caches. In *Proceedings of DATE 2010*.
[4] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of ISCA 2009*, pages 24–33.
[5] S. Cho and H. Lee. Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance. In *Proceedings of MICRO 2009*, pages 347–357.
[6] P. Zhou and *et al.* A durable and energy efficient main memory using phase change memory technology. In *Proceedings of ISCA 2009*, pages 14–23.
[7] M. K. Qureshi and *et al.* Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Proceedings of MICRO 2009*, pages 14–23.
[8] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of HPCA 2010*, pages 123–132.
[9] N. H. Seong, D. H. Woo, and H. S. Lee. Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In *Proceedings of ISCA 2010*, pages 134–145.
[10] W. Xu and *et al.* Data manipulation techniques to reduce phase change memory write energy. In *Proceedings of ISLPED 2009:*, pages 237–242.
[11] Y. Zhang, J. Yang, and R. Gupta. Frequent value locality and value-centric data cache design. *SIGPLAN Not.*, 35(11):150–159, 2000.
[12] D.H. Kang and *et al.* One-dimensional heat conduction model for an electrical phase change random access memory device with an $8f^2$ memory cell (f=0.15 μm). *Journal of Applied Physics*, 2003.
[13] S.J. Hudgens. OUM nonvolatile semiconductor memory technology overview. *In Proceedings of Materials Research Society Symposium*, 2006.
[14] Y. Zhang and *et al.* An integrated phase change memory cell with ge nanowire diode for cross-point memory. *In Proceedings of VLSIT 2007*, pages 98–99.
[15] J. Yang, Y. Zhang, and R. Gupta. Frequent value compression in data caches. In *Proceedings of MICRO 2000*, pages 258–265.
[16] P. Zhou and *et al.* Frequent value compression in packet-based noc architectures. In *Proceedings of ASP-DAC 2009*, pages 13–18.
[17] J. Yang and R. Gupta. Energy efficient frequent value data cache design. In *Proceedings of MICRO 2002*, pages 197–207.
[18] M. Mehrara and T. Austin. Exploiting selective placement for low-cost memory protection. *ACM Trans. Archit. Code Optim.*, 5(3):1–24, 2008.
[19] J. Yang and R. Gupta. Frequent value locality and its applications. *ACM Trans. Embed. Comput. Syst.*, 1(1):79–105, 2002.