Exploiting Page Write Pattern for Power Management of Hybrid DRAM/PRAM Memory System

TIEFEI ZHANG¹, JIANGUO XING¹, JIXIANG ZHU¹ AND TIANZHOU CHEN²

¹School of Computer Science and Information Engineering
Zhejiang Gongshang University
Hangzhou, 310018 P.R. China
E-mail: {tfzhang; jgxing; zhujx}@mail.zjgsu.edu.cn

²College of Computer Science and Technology
Zhejiang University
Hangzhou, 310027 P.R. China
E-mail: tzchen@zju.edu.cn

The main memory has become a power bottleneck for computer systems. To reduce the energy dissipation of main memory, the non-volatile phase-change RAM (PRAM) has emerged as one of the promising memories due to its high density and low standby power. But PRAM has its intrinsic disadvantages of long write latency and high write energy. Hence, the hybrid DRAM/PRAM main memory is proposed to provide the advantage of low standby power of PRAM as well as high performance and low access power of DRAM. Data management, such as migrating frequently accessed data from PRAM to DRAM, is widely employed by past work to achieve the benefits of both DRAM and PRAM. But the past work is oblivious to the data migration energy overhead, which is considerable as confirmed by our study. In this work, we studied the page write access pattern of pages within and across applications and observed that two pattern widely exist, namely write-average and write-cluster. We then propose an adaptive page migration strategy (APMS) that migrates data of a page based on its write pattern to reduce the migration energy in DRAM/PRAM hybrid memory system. Our APMS exploits the write pattern of a page and determines the amount of data that should be migrated against the existing page-level data migration strategy. Experiments were conducted to verify the validity of the strategy. The results show that APMS can reduce the energy consumption of most applications with negligible performance impact and energy overhead.

Keywords: DRAM/PRAM hybrid memory, write energy, data migration, write pattern, data refresh

1. INTRODUCTION

The power consumption is a critical concern in current computer design. Main memory has become a power bottleneck for computer systems. Recent studies [1] have shown that CPU is no longer the primary energy consumer, main memory has become a major energy consumer, contributing to 30% [2] of total energy consumption of a motherboard (blade) on modern server systems. The problem is exacerbated by the continuing scaling of DRAM density, which in turn leads to significant amounts of standby power due to refresh [3, 4]. Different from DRAM, phase-change RAM (PRAM) [5] does not require refresh to retain its data because of its non-volatility. So it has much lower standby power. PRAM also has much higher density about 2-4 times [5] of that DRAM,
enabling good scalability along with the growing memory capacity demands of cores in CMP [6]. On the other hand, PRAM has its intrinsic limitations of long write latency and high write energy [7]. To take advantage of fast speed of DRAM and high density of PCM, there is a clear incentive for combining the two tech. into a single, hybrid memory system [8, 9].

A conventional hybrid memory system is composed of a large PRAM and a small DRAM. The PRAM functions as a background memory to exploit its low standby power. The DRAM works as a cache to maintain hot data which is frequently written to reduce write energy and write latency. Given the fact that the DRAM's capacity is limited, data management technique is developed to maximize the benefits of both DRAM and PRAM while minimize their negative effects. Two of the state-of-the-art techniques are page allocation and page migration. DRAM is preferable for the page which is frequently written [8]. Furthermore, read-intensive pages are prone to PRAM area when the write-intensive page are allocated to DRAM area [10, 11]. For the page migration, a critical step is to identify the frequently written pages in PRAM. Two schemes are proposed to track the hot pages in PRAM. The most straightforward way is to attach a counter to each page in PRAM to record its write times. Once the value of its write counter reaches a certain threshold, the page is assumed to be hot and then migrated to DRAM. Another method [9] is to monitor the memory access stream and then sort the pages according to access times and access frequencies. The obtained information provides support for page migration between PRAM and DRAM to improve performance and reduce energy.

The existing studies are oblivious to page migration energy. Each page migration involves a series of operations on a considerable amount of data, e.g., 8k for a typical physical page. Given the huge number of migrated pages, the page migration energy becomes quite significant. Furthermore, none of above researches has addressed DRAM refresh energy in the hybrid memory setting. As refresh energy consumes a large portion of DRAM energy [12], the refresh energy problem still deserves attention because DRAM’s capacity is still considerable though not comparable to PRAM.

In this paper, we propose an adaptive page migration strategy (APMS) based on page write patterns to reduce the migration energy in DRAM/PRAM hybrid memory system. The write access patterns of pages are studied, and two particular write patterns are widely observed within and across pages, namely write-average and write-cluster. The write updates are scattered within a page in the write-average pattern while the write updates are clustered in certain blocks of a page in write-cluster pattern. The goal of migrating data to DRAM is to utilize its lower write energy. So there is no need to migrate a full page if it belongs to write-cluster pattern since most of its data does not serve write requests but incurs energy overhead if being migrated. On the other hand, a full-page migration is necessary in the case of a write-average page (if it is frequently written). Our APMS exploits such information and determines the amounts of data that should be migrated against the always page-level data migration of existing strategies. As a result, our strategy not only helps alleviating the write energy in a hybrid memory system but also minimizes the migration energy overhead.

2. THE MOTIVATION

A data block in the last level cache will be written back to the main memory in the
event of cache replacement if it is dirty. The data in the main memory is organized in a page, whose capacity is much larger than that of a cache block. Thus, each page contains many cache blocks. For instance, there is 128 cache blocks in a typical 8KB-size physical page. The write pattern of a page is the distribution of write updates over cache blocks. As we see in Fig. 1, there are two typical write patterns. The first is write-average (WA) pattern, where groups of cache blocks bear similar number of write-back operations. The second is write-cluster (WC), for that most of the write updates occur over a few cache blocks. Both patterns exist widely within and across applications. Figs. 1 (a) and (b) show an example WA pattern and WC pattern from benchmark ammp and apsi, respectively. The x axis is the index of cache blocks in a single physical page, and the y axis denotes the number of write update operations. Fig. 1 (a) shows a typical write-cluster pattern, while Fig. 1 (b) demonstrates a typical write-average pattern. Figs. 1 (c) and (d) show that these two patterns also exist within a single application (galgel).

![Fig. 1. The write access patterns.](image)

The existing DRAM/PRAM data management mechanism migrates the frequently-written data from PRAM to DRAM by exploiting DRAM’s advantages such as low write energy and fast write speed. The side effect introduced by this approach is the migration cost, especially the energy consumed by a series of read and write operations along a page-size amount of data moving. A natural way to reduce the energy cost is to minimize the data migrated. As observed in Figs. 1 (a) and (d), for the page of write-cluster pattern,
there is no need to migrate the whole-page data because only a small portion of cache blocks are frequently written. By migrating the frequently-written data only, the migration energy consumption is reduced. Based on the above observation, we propose APMS towards data management between PRAM and DRAM. Instead of migrating all the data of a single page, the amount of data migrated by our strategy varies according to the write pattern of the targeted page. As a result, the amount of data migrated is reduced and energy is further saved.

3. THE OVERALL DESIGN OF APMS

The purpose of this section is to present the overall design of APMS. Fig. 2 shows the memory system implementing APMS. The hybrid memory is composed of a large PRAM and a small DRAM, sharing a single memory controller. The data from the disk is first assigned to the PRAM. A write tracking module is added to the memory controller, with the goal to track the write update information within a page. After a pre-assigned interval, another added module known as data migration module is triggered to do data migration. The amount of data migrated depends on the recorded write distribution information. After the migration, the location of the moved data should be updated at the memory controller for directing the following requests.

![Fig. 2. The hybrid memory system.](image)

The existing data management techniques [9] migrate data at page level by default. The critical attribute of APMS is to migrate the data at smaller granularity due to the write pattern of a targeted page. With a large granularity, the migration cost is high while a small granularity results in increased storage in write information tracking. To achieve a good trade-off between pros and cons, a page is split into mini pages, which is the basic unit for data migration. For the write-cluster pattern, only the mini page that possesses the frequently written blocks is migrated. And for the write-average pattern, whether all the mini pages being migrated or not is up to the overall number of write updates occurred so far. As a result, the migration cost is optimized by reducing the amount of data migrated. The next subsection will give the details of the write tracking module design.

3.1 The Write Tracking Table

As described in previous section, the amount of data migrated by APMS varies
from a single mini page to a whole page based on its write pattern. To record such information, each page is assigned with a write recorder entry, aiming to record the number of write backs occurred so far. We empirically determined that each mini page consists of 32 cache blocks. All the write record entries assigned to pages are organized as a write tracking table that resides in the memory controller. Each entry consists of page address and several write counters (each with 8 bits for one mini page). As for a 8-KB size page, it contains four mini pages. Thus, its write record entry has four write counters. As the memory footprint of an application is small within its executing period, not exceeding 512KB [13] usually, 128 entries is enough for a single application. As for multithreaded applications, more entries are required. The num of write record entries is empirically set to 128*64, and organized as a cache to manage the storage. As each entry contains the physical address of a page and four write counters, it requires (35+8*4)=67 bits to store one write entry (the physical page number is assumed 35 bits and each counter is with 8 bits). Then it requires overall 67kB storage overhead, as the main memory size is assumed to be 2.5Gb in this paper, the storage overhead is negligible. As the storage overhead is negligible. Besides, the information update of a entry is performed off the critical path, causing no performance impact.

The write tracking table supports a series of operations over record entries. When a data request to a particular page arrives, a new entry is created if the page's entry does not exist yet. The address domain is updated using the address of the coming request, and the corresponding write counter of the mini page is incremented by one. If the requested page’s entry already exists, then the write counter of the targeted mini page is updated. When a page is evicted to reclaim DRAM space, its corresponding record entry is deleted. When a page is evicted to reclaim DRAM space, its corresponding record entry is deleted.

3.2 The Migration Strategy Design

The data migration from PRAM to DRAM involves the following operations: Read the data from the PRAM data array to its row buffer, fetch the data out through the I/O circuits and send it to the row buffer of DRAM, then flush the data into the data array of DRAM. Each step consumes energy, termed as $E_{\text{read PCM}}$, $E_{\text{row PCM}}$, $E_{\text{row DRAM}}$ and $E_{\text{write DRAM}}$.

The energy cost of a data migration $E_{\text{migration}}$ is the sum of the above:

$$E_{\text{migration}} = E_{\text{read PCM}} + E_{\text{row PCM}} + E_{\text{row DRAM}} + E_{\text{write DRAM}}$$ (1)

$$E_{\text{read PCM}} = E_{\text{read PCM per bit}} \times \text{bits per row}$$ (2)

$$E_{\text{write DRAM}} = E_{\text{write DRAM per bit}} \times \text{bits per row}$$ (3)

The cache stores data back to main memory at block granularity, consuming different amounts of energy for DRAM($E_{\text{line DRAM}}$) and PRAM($E_{\text{line PRAM}}$). The $E_{\text{line PRAM}}$ is much larger than $E_{\text{line DRAM}}$ as the write energy of PRAM is higher. To take advantage of low write energy of DRAM, the energy benefits should outweigh the energy cost spent in data migration. So there is a minimum number of writes required in DRAM after the migration as follows:

$$W_{\text{min}} = E_{\text{migration}} / (E_{\text{line PRAM}} - E_{\text{line DRAM}})$$ (4)
which means that it is worth migrating the data if its left write times exceeds \( W_{\text{min}} \). Our strategy applies a history-based policy to estimate the write times. The rational of this strategy is based on the page-level locality [14], which means that the current write-intensive pages are likely to be frequently-written in the next period.

When a pre-set tracking period ends, the write counters of each entry are checked to identify those mini-pages with write update sums bigger than \( W_{\text{min}} \) as the migrated candidates. Fig. 3 shows the migration procedure. When data migration is complete, the memory controller assumes the responsibility of recording which portion of a page resides in DRAM or PRAM. This incurs limited storage overhead because each application's memory footprint is small and the data in DRAM would be evicted, allowing the recycle of the corresponding storage. Note that the data migration is transparent to software, requiring no extra modification to OS, reducing the implementation complexity.

![Fig. 3. The migration strategy.](image.png)

Once a read request arrives at the memory controller, it will be assigned to the waiting list instead of being serviced at once. During its waiting, the write tracking table is first searched to check the presence of the targeted data. If the data is found in DRAM, the data request will be forwarded and then serviced by the DRAM memory. Otherwise, the PRAM is responsible to handle the request. If the targeted data is not in main memory, then the data will be loaded from disk to PRAM. The above operations do not incur any performance overhead because they are off the critical path.

Compared to a read request, it becomes quite more complicated to handle a write request. Similarly, the first step is searching the write tracking table. If the requested data is found in DRAM, then the data will be written to DRAM. If the data is found in PRAM, then the data will be written to PRAM and meanwhile, the information of the targeted entry is updated. If the targeted entry is not found in the write tracking table, it will be created and initialized with the information of the incoming request. If the requested data is not in main memory yet, it should be loaded from disk to PRAM.

4. **DRAM SPACE RECYCLE**

With data continuously migrated to DRAM, some data in DRAM becomes rarely used as time passes by hence keeping them has no contribution to performance improvement and write energy saving but incurs significant refresh energy cost. Evicting rarely used data would not only help reclaim DRAM space but also reduce the refresh energy cost. Similar to the LRU eviction policy used in cache management, we propose a DRAM row eviction policy based on the access recency of the rows. The least recently used pages are assumed to be useless in the future, so they would be evicted. A major difference is that the LRU cache eviction policy operates over a set granularity, usually
only including 4-32 cache blocks, requiring limited resource to implement. While our policy performs over the DRAM space, which is hundreds of thousands of pages. Implementing the LRU eviction policy straightforwardly the way of that cache does is impractical. Therefore, we propose a ranking mechanism on basis of multi-queue (MQ) [15] to approximate the LRU policy at low implementation overhead but reasonable accuracy. To model a multi-queue structure, each row in a bank is associated with a $k$-bit counter. When the data is first migrated to DRAM, its $k$-bit counter is initialized to zero. If its data is not accessed within a tracking period, its counter is incremented by one. Otherwise, its counter is decremented by one. In this way, the rows with counter value larger than $2^{k-1}$ are assumed to be the eviction candidates, whose addresses can be organized as a queue to accelerate the selection of eviction victim.

![Fig. 4. The row eviction strategy.](image)

After the eviction candidates are determined, the next problem is to decide when and how to evict a page. The operations evolved in evicting a page is largely dependent on the current state of the page. The current state of the data in a page, whether it is valid or dirty, can be obtained from reading its state register. The most rare case is that all the data of a page is clean ever since its migration, that means no write requests ever occur. Then the operations are quite simple, deleting its page descriptor and then recycling its record entry. For the other cases, the data in DRAM has been updated and become dirty. To maintain memory coherence, the dirty data must be migrated back to PRAM. As mentioned in above section, it requires a series of read/write operations to data array and row buffers of memory banks to complete data migration. These accesses may be in contention with the other normal reads and writes, resulting in adverse impact on the performance. So that the migration request should be serviced off the critical path, during idle cycles at the memory bank. As previous work [16] shows that the average memory bandwidth utilization is low, leaving lots of opportunities for performing data migration. Scheduling the migration requests requires a similar methodology of that used in PreSET [12].

The data migration from DRAM back to PRAM requires updating the PRAM with the data transferred from DRAM at mini page granularity. Each mini page contains a lot of bits. For a 2KB mini page, there is overall 16384 bits. Straightforwardly writing such amounts of data causes significant energy cost because PRAM’s write energy is much larger than that of DRAM. As shown by previous researches [17-19], a great amount of memory write operations are redundant, that is, the write update to a bit data does not change its original value, or in other words, the newer value equals with the old version.
It is observed that the write redundancy phenomenon also exists at mini-page. So the same read-before-write tech. [17] is adopted in our design to alleviate the PRAM write energy.

5. EXPERIMENTS AND RESULTS

This section provides the performance and energy evaluation for APMS. Firstly, the energy models of DRAM and PRAM are discussed. Then the experimental setup is introduced and finally the experimental results.

5.1 Energy Model for DRAM and PRAM

The access energy of both DRAM and PRAM includes read/write energy of data arrays and row buffers. Table 1 shows the energy per bit for the above read or write operations [20]. The capacity of a cache block is 64 bytes (512 bits). In addition to the read/write energy, the energy consumed by accessing write tracking table should also be taken into account. The write tracking table is implemented as SRAM cache. Its access energy is about $8.2 \times 10^{-4} \text{nJ}$, being obtained by CACTI [21]. Each row is attached with a 6-bit counter to implement the ranking mechanism. So the storage overhead of each row is 14 bits (plus a state register), which is negligible compared to an 8KB row (about 0.2%). The energy consumption of operating these bits is also negligible.

<table>
<thead>
<tr>
<th>Energy Consumer</th>
<th>Read(pJ/bit)</th>
<th>Write(pJ/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM Array</td>
<td>1.17</td>
<td>0.39</td>
</tr>
<tr>
<td>PRAM Array</td>
<td>2.47</td>
<td>16.82</td>
</tr>
<tr>
<td>Row Buffer</td>
<td>0.93</td>
<td>1.02</td>
</tr>
</tbody>
</table>

5.2 Experiment Setup

The gem5 [22] full system simulator is adopted to collect the memory access stream. Then the stream is used as an input for an in-house hybrid PRAM/DRAM memory simulator to evaluate the energy and performance of APMS.

Table 2 shows the memory hierarchy configuration, where the hybrid memory configuration is the same as that of [20]. The test applications are memory-intensive ones.

<table>
<thead>
<tr>
<th>L1 Cache</th>
<th>32kB L1 instr./data cache, write back and write allocate policy, 2-way associative, 64 bytes block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Cache</td>
<td>512kB, write back and write allocate policy, 8-way associative, 64 bytes block size</td>
</tr>
<tr>
<td>Memory Arch.</td>
<td>DRAM:512MB,1rank(4banks), 8kB row buffer PCM: 2GB, 1 rank(8 banks), 4kB row buffer</td>
</tr>
<tr>
<td>Memory Latency</td>
<td>DRAM: hit time 40ns, miss 80ns PRAM: hit time 40ns, miss(clean/dirty) 128ns/368ns</td>
</tr>
</tbody>
</table>
selected from the SPEC2006. And multithreaded applications are from splash2. The test application each runs on the gem5 simulator. The first 1 billion instructions are executed as the warm up phase and after that the simulation begins on phase-oriented basis that each tracking phase contains 50M instructions.

At the end of each phase, the migration module in the memory controller is trigged to perform the data migration. During the simulation, each memory access is traced by recording its operation (read or write) type and the targeted address. When the simulation is complete, the collected memory access trace is forwarded to an in-house hybrid memory simulator for energy and performance evaluation of APMS. An existing page migration mechanism for DRAM/PRAM hybrid memory system known as RaPP [9] is adopted as a baseline. A critical difference between RaPP and APMS is that RaPP is oblivious to the write pattern of a page and thus always migrates data at page granularity.

5.3 Results

Fig. 5 shows the percentage of migration energy with APMS, varying from 52.5% (galgel) to 99% (facerec) compared to the baseline. The average energy reduction is about 10.5%. The effect of APMS for application facerec is not so obvious because most of its pages are frequently written and exhibit the write-average pattern. Data are most probably migrated at page level, leaving less room for APMS takes effect. The test application galgel achieves the most significant migrated energy saving. Because it has a lot of pages with write-cluster pattern.

The APMS helps to reduce the migration energy of all test application, but it has adverse impact on the access energy, e.g., the read energy. The APMS focuses on migrating frequently-written data while it is oblivious to the frequently-read data. Furthermore, APMS leaves those rarely written data in PRAM. Thus, compared to RaPP, a larger amount of data is read from and written to PRAM, resulting in increased access energy since the access energy of PRAM is higher than that of DRAM. The increased access energy diminishes the energy benefits of APMS. Fig. 6 shows that the dynamic
energy (including read, write and migration) saving of most applications is much less compared to the migration energy saving. For example, the case with galgel drops to 18.7% while that of ammp has no energy saving. In particular, the results of apsi is about −0.7%, which means APMS does not help but aggravate the overall dynamic energy. The ammp and apsi have higher dynamic energy consumption because a large amount of accesses to its page are reads, and also these reads are clustered in certain mini pages that are not migrated to DRAM. Since PRAM has higher read energy compared to the DRAM, the sum of read energy caused by those unmigrated mini-pages outweigh the energy reduction of APMS. However, APMS helps to reduce the dynamic energy for most of the applications.

The APMS affects the memory access latency in two ways. Firstly, migrating data to DRAM space can accelerate the following write accesses. Secondly, APMS is oblivious to read-intensive data and rare-written data so that these data is more likely to reside in PRAM than in the case of RaPP, leading to longer memory access latency. So further evaluation is required to achieve the actual memory access latency. Figure 7 shows the memory access latency with APMS compared to the baseline RaPP. For all the applications, the memory latency increases but in a negligible way. For the worst case with apsi, the memory access latency increases by 2.5%. On average, the memory access latency increases by only 0.7%.

6. RELATED WORKS

Our policy utilizes the redundant bits elimination [18, 19] as a way to minimize the number of bits that are written to PRAM in case of data migration back. We observe the number of write accesses among the data within a single row differs and then exploit such characteristic to migrate those frequently to DRAM at mini page granularity. Our policy is of much less complexity compared to the micro page strategy [13]. The micro page strategy clusters frequently accessed blocks into micro page and migrates those micro pages together. The micro page involves great changes to the existing hardware, and also requires the support from the operating system. Our policy is also different from
the row eviction policy [8]. A row is firstly allocated into DRAM, and then the DRAM row eviction periodically evicts a row out of DRAM to reserve room for the other frequently accessed rows. But our policy firstly allocates blocks to PRAM and then migrates those frequently accessed blocks to DRAM. By this way, our policy can fully utilize the space of DRAM to serve the write requests.

7. CONCLUSION

In this work, we studied the page write access pattern of different pages within and across applications and observed that two pattern widely exist, namely write-average and write-cluster. We then propose APMS that migrates data of a page according to its write pattern. Instead of migrating data at page granularity all the time, the migration unit of APMS is smaller-mini page. For pages of write-cluster pattern, only the mini pages with the frequently written blocks are migrated, leading to less data migrated and migration energy saving. Experimental results show that APMS can reduce the overall energy of most test applications compared to RaPP at negligible implementation cost and performance impact.

![Fig. 7. The memory latency impact.](image)

ACKNOWLEDGEMENT

A Project is supported by Zhejiang Provincial Natural Science Foundation of China LQ14F020001, LQ13F020004, LQ12F02017. A Project is supported by Scientific Research Fund of Zhejiang Provincial Education Department Y201432106. A Project is supported by NSFC 61379035.

REFERENCES


**Tiefei Zhang** was born in 1984. He received his B.S. degree in Control Science and Engineering from Zhejiang University, Hangzhou in 2007. He received his Ph.D. degree from College of Computer Science and Technology, Zhejiang University. He is currently a staff from Zhejiang Gongshang University. His research interests are energy-efficient design and power-aware design of computer architecture.

![Tiefei Zhang](image1)

**Jianguo Xing** was born in 1971. He received his Ph.D. degree in Control Science and Engineering from Zhejiang University, Hangzhou in 2002. He is an Associate Professor in the department of Computer Science at Zhejiang Gongshang University. His research interests include reconfigurable computing and Evolvable hardware.

![Jianguo Xing](image2)

**Jixiang Zhu**, born in 1984 and received his Ph.D. in Computer Science from WuHan University in 2010. He is currently a Lecturer at the Department of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His current research interests include image processing, 3-D reconstruction and intelligent algorithms. E-mail: zhujx@mail.zjgsu.edu.cn.

![Jixiang Zhu](image3)
**TianZhou Chen** was born in 1970. He obtained his B.S. degree in Computer Software in 1994. He studied for M.S. degree in Computer Architecture, and received his Ph.D. degree in Computer Application from Zhejiang University in 1998. He is a Professor of Computer Science at Zhejiang University. And he is the Vice Director of computer systems engineering research institute of Zhejiang University. He is a member of IEEE and ACM. His current research interests include computer architecture, multicore system, on-chip interconnection, embedded system and power-aware computing.