Improving Energy Efficiency and Lifetime of Emerging Memory Systems

BY

BAHAREH POURSHIRAZI
B.S., Shahid Beheshti University, 2010
M.S., Shahid Beheshti University, 2013

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2019

Chicago, Illinois

Defense Committee:

Zhichun Zhu, Chair and Advisor
Ajay Kshemkalyani, Computer Science
Igor Paprotny
Wenjing Rao
Zhao Zhang

To my parents!

ACKNOWLEDGEMENT

First off, I would like to express my sincere gratitude to my dissertation advisor, Dr. Zhichun Zhu, for her invaluable guidance, encouragement and support during my PhD research.

I would like to thank my dissertation committee members Dr. Zhao Zhang, Dr. Igor Paprotny, Dr. Wenjing Rao and Dr. Ajay Kshemkalyani for their insightful comments on my dissertation. I would like to also thank Dr. Gokhan Memik of Northwestern University with whom I have been fortunate to collaborate on several research projects. I would like to also thank Dr. Elaheh Bozorgzadeh from University of California at Irvine for presenting our paper in DATE2018 conference. Special thanks to the amazing professors and staffs of ECE department of University of Illinois at Chicago for their enormous support.

I am forever indebted to my parents for their unconditional love, constant encouragement and support throughout my entire life. They have always been and will be my greatest teachers. I am also thankful to Majed. This PhD would not have been possible without his love, encouragement and the unbelievable faith he has in me.

BP

CONTRIBUTION OF AUTHORS

The contents of Chapter 3 have been published in IEEE International Parallel and Distributed Processing Symposium (IPDPS2016) [66]. Prof. Zhichun Zhu, my advisor, was the leader of this project. I was responsible for coming up with the ideas, performing the experiments and writing the paper.

The contents of Chapter 4 have been published in The International Symposium on Memory Systems (MEMSYS2017) [65]. Prof. Zhichun Zhu, my advisor, was the leader of this project. I was responsible for coming up with the ideas, performing the experiments and writing the paper.

The contents of Chapter 5 have been published in Design, Automation and Test in Europe Conference (DATE2018) [63] and in ACM Transactions on Design Automation of Electronic Systems (TODAES2019) [64]. This research study was conducted in collaboration with two individuals from the Northwestern University. Prof. Zhichun Zhu, my advisor, was the leader of this project. I was responsible for coming up with the ideas, performing major parts of the experiments and writing the paper. Majed Valad Beigi of Northwestern University helped me with performing the experiments and writing the paper. Prof. Gokhan Memik of Northwestern University helped me with writing the paper.

The study presented in Chapter 6 was conducted in collaboration with two individuals from the Northwestern University. Prof. Gokhan Memik of Northwester University and Prof. Zhichun Zhu lead this project. I was responsible for coming up with the ideas, performing the experiments and writing parts of the work, which are included in this thesis.

TABLE OF CONTENT

TABLE OF CONTENT (Continued)

TABLE OF CONTENT (Continued)

CHAPTER **PAGE**

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

DRAM	Dynamic Random-Access Memory
PCM	Phase Change Memory
ReRAM, RRAM	Resistive Random-Access Memory
STT-RAM	Spin-Transfer Torque Random-Access Memory
NVM	Non-Volatile Memory
LLC	Last Level Cache
OS	Operating System
RAS	Row Address Strobe
SDRAM	Synchronous Dynamic Random-Access Memory
DDR	Double Data Rate
LPDDR	Low Power Double Data Rate
SRAM	Static Random-Access Memory
TLB	Translation Lookaside Buffer
MEA	Majority Element Algorithm
eDRAM	Enhanced Dynamic Random-Access Memory
CMP	Chip Multi-Processor
SECDED	Single Error Correction/Double Error Detection

LIST OF ABBREVIATIONS (Continued)

ECC	Error Correction Code
LRU	Least Recently Used
MRU	Most Recently Used
NAS	NASA Advanced Supercomputing
PARSEC	Princeton Application Repository for Shared-Memory Computers
SPEC	Standard Performance Evaluation Corporation
ACT	Activation command
PRE	Precharge command
SR	Self-Refresh
DPD	Deep Power Down
CPU	Central Processing Unit
IPC	Instructions Per Cycle
MQ	Multi Queue
MPKI	Misses Per Kilo Instructions
WPKI	Writes Per Kilo Instructions
RPKI	Reads Per Kilo Instructions
GBps	Giga Bytes per second
MC	Memory Controller

SUMMARY

Main memory is a storage component in a computer system, where data of currently running applications and programs are stored. In modern computing systems, the number of concurrently running applications and each application's working set size are increasing as a result of continued advancements in technology. These have resulted in a growing aggregate amount of data that the main memory must support. For several decades, Dynamic Random-Access Memory (DRAM) has been the dominant technology for building main memories. However, DRAM can no longer satisfy the memory capacity demands of the modern-day applications due to its scalability limit; it is very expensive and difficult to scale DRAM cells down to feature sizes smaller than 20nm [27, 50]. On the other hand, due to the dynamic, leaky nature of its capacitive cells, DRAM requires periodic refresh operations to maintain its data integrity. In addition to wasting energy, refresh operations degrade system performance by interfering with regular accesses to the main memory. Ever worse, the adverse effects of DRAM refresh are expected to aggravate with each generation of technology. It is predicted that refresh would account for 50% of throughput loss and 50% of the total energy consumption in a future 64GB DRAM system [5, 46]. In current server systems, DRAM memory consumes 20% to 40% of the total system energy [45, 78].

Due to the limited scalability and high refresh power of DRAM, other new technologies such as Phase Change Memory (PCM), Resistive Random-Access Memory (ReRAM) and Spin-Transfer Torque RAM (STT-RAM) have recently emerged as potential alternatives to DRAM. These Non-Volatile Memory (NVM) technologies are much more scalable than DRAM and within the same area budget used by a DRAM, can provide a much higher capacity for the main memory due to

their higher densities [38, 39, 70]. Moreover, NVM cells are resistive and can preserve their data without being refreshed. However, NVMs also have a number of shortcomings. First, they have longer access latencies compared to DRAM. Second, NVMs consume much higher dynamic energy compared to DRAM (especially for write operations). Finally, NVM cells have limited write endurance. Hence, simply replacing DRAM with an NVM, without any modifications, could adversely impact memory system performance, energy efficiency, and lifetime.

In this thesis, we present novel architectural techniques that enable incorporating emerging non-volatile technologies into the memory system design while fulfilling system requirements on performance and energy efficiency. We start by studying "hybrid main memories". Hybrid main memories, which incorporate both DRAM and NVM, enable systems to benefit from the large capacity of an NVM and lower access latency and energy of a DRAM. We first present *Refree*, a scheme that eliminates DRAM refresh operations in a hybrid DRAM/PCM main memory [66]. Then, we present *NEMO*, a scheme that improves the energy efficiency of a mobile device with hybrid DRAM/PCM main memory by placing cold memory pages in PCM [65]. Our third work called *WALL*, focuses on PCM-based main memories. More specifically, the energy efficiency and lifetime of a PCM-based main memory is improved by reducing the number of writebacks from the Last Level Cache (LLC) to PCM. Finally, our last work called *DynaSwap* is a page swap management scheme that improves performance and energy efficiency of a flat address space hybrid DRAM/NVM main memory.

Refree eliminates DRAM refresh operations in a hybrid DRAM/NVM main memory, where DRAM serves as a hardware-managed cache for the PCM. The basic idea behind *Refree* is to evict a row from DRAM if at any point the row has to be refreshed. In fact, such rows mostly hold nonvaluable (i.e., useless in near future) data. Thus, there is no need to refresh and keep those

rows in the DRAM. In addition, a recently accessed row has already been "refreshed" by the access and does not need to be refreshed either. To keep the data integrity, the dirty columns of the row that is being evicted from the DRAM cache must be written back to PCM. Since the PCM has long write latency, we propose a scheme that distributes writebacks of a dirty DRAM row over an epoch time (i.e., 128ms) instead of performing them all at once, to prevent long-time blockage of other requests, which are actually DRAM read misses, to the PCM storage. *Refree* can effectively reduce the memory power consumption with only a small performance impact. The effectiveness of *Refree* would further improve for future systems with larger DRAM sizes.

NEMO improves the energy efficiency of a hybrid DRAM/PCM main memory in a mobile device. To do so, *NEMO* powers off as many power-hungry DRAM components as possible, as long as it does not impact the performance. Specifically, when the mobile device is in idle state, which is the case most of time, only a selective set of data that is critical to performance is kept in a single DRAM rank, while the rest of data is stored in PCM, so that the remaining DRAM ranks can be powered off. To do so, *NEMO* classifies memory pages based on their usage frequency and recency into hot and cold. Then, it places the hot pages that are more likely to be reused in future in the DRAM, which has lower access latency compared to PCM, and stores the cold memory pages in PCM, which has near-zero idle power. In addition, *NEMO* predicts the number of DRAM ranks that need to be powered on when the mobile device becomes active for further energy saving. *NEMO* can effectively reduce the memory power consumption without negative performance impact.

WALL improves performance, energy efficiency, and lifetime of a PCM-based main memory system by reducing the number of writebacks from LLC to PCM. In general, *WALL* consists of a writeback-aware set balancing mechanism and a writeback-aware replacement policy. Writebacks

of the last level cache are not uniformly distributed among its sets; some sets have far more writebacks than others while some sets rarely see a writeback. The proposed set balancing mechanism reduces the number of writebacks by employing the underutilized sets with infrequent writebacks as storage units (inside LLC) for storing the evicted dirty lines of sets with many writebacks. Moreover, the proposed writeback-aware replacement policy tries to keep the dirty blocks that are frequently accessed after eviction in the LLC. To do so, it allows the dirty eviction victims (i.e., dirty LRU block) to stay in the cache and be re-accessed; if the block becomes LRU block again without being accessed, it will be evicted from LLC then. The WALL design is very simple with small overheads.

In a hardware-managed flat address space hybrid DRAM/NVM main memory, a swap group is defined as a group of pages in DRAM and NVM that can be swapped with each other. *DynaSwap* dynamically associate swap groups with each other in such a way that a swap group with many high frequently accessed pages can benefit from the DRAM space of a swap group with low frequently accessed pages. In other words, unlike previous studies that create swap groups solely based on the physical address of memory pages, we try to create swap groups based on their access patterns. Assigning enough DRAM segments to a swap group based on its demand (i.e., its access patterns in the current interval) can improve performance and energy efficiency of the memory system by reducing the number of unnecessary swaps.

CHAPTER 1

INTRODUCTION

In modern computer systems, "memory wall" problems including main memory's high access latency, high energy consumption and lack of scalability are among the major bottlenecks of system performance and energy efficiency. In today's computer systems with multi-core and many-core processors, the increase in the number of concurrently running applications and each application's working set size have resulted in a growing aggregate amount of data that the main memory must support. For several decades, DRAM has been the dominant technology for building main memories. However, DRAM can no longer satisfy the memory capacity demands of the modern-day applications due to its scalability limit; it is very expensive and difficult to scale DRAM cells down to feature sizes smaller than 20nm [27, 50]. On the other hand, due to the dynamic, leaky nature of its capacitive cells, DRAM requires periodic refresh operations to maintain its data integrity. In addition to wasting energy, refresh operations degrade system performance by interfering with regular accesses to the main memory. Ever worse, the adverse effects of DRAM refresh are expected to aggravate with each generation of technology. It is predicted that refresh would account for 50% of throughput loss and 50% of the total energy consumption in a future 64GB DRAM system [5, 46]. In current server systems, DRAM memory consumes 20% to 40% of the total system energy [45, 78].

Due to the aforementioned problems of DRAM, NVM technologies such as PCM [40, 70, 90] and ReRAM [11] have emerged as DRAM alternatives. These NVM technologies are much more

scalable and denser than DRAM [38, 39]. Moreover, NVM cells are resistive and can preserve their data without being refreshed. Therefore, the static energy overhead of NVMs is much lower than that of DRAM. However, NVMs generally suffer from longer access latencies and higher dynamic energy consumptions (especially for the write operations) compared to DRAM. Moreover, NVM cells have limited write endurance. Hence, simply replacing DRAM with an NVM as main memory, without any modifications, can adversely impact memory system performance, energy efficiency and lifetime.

To benefit from the large capacity of an NVM while keeping access latency and energy close to that of a DRAM, hybrid main memories, which incorporate both DRAM and NVM, have been proposed. In a hybrid DRAM/NVM main memory, DRAM can serve as a hardware-managed cache for the NVM [41, 51, 70, 84]. Existing studies have shown benefits from a relatively small DRAM cache. However, in future, satisfying system demands on performance and energy-efficiency with such fair-sized DRAM caches becomes impossible. Generally, a larger DRAM cache can alleviate overheads of hybrid main memory on system performance and energy consumption by accommodating more data and reducing the number of accesses serviced in NVM. However, the larger the DRAM cache, the higher the refresh costs. We address this problem in Chapter 3 by presenting *Refree*, a scheme that eliminates DRAM refresh operations in a hybrid DRAM/PCM main memory [66]. Specifically, when it is time to refresh a row, *Refree* evicts the row from the DRAM cache instead. This can be done since a recently accessed row has already been "refreshed" by the access; while a row that hasn't been accessed for a long time is very likely to hold obsolete data and does not need to be refreshed and kept in the DRAM. If an evicted row is dirty, it will be written back to the PCM. To alleviate the potential performance loss due to the long PCM write latency, *Refree* distributes writebacks of a dirty DRAM row over an epoch time (i.e., 128ms) to prevent long-time blockage of other requests to the PCM devices. Our simulation results reveal

that *Refree* can achieve an average of 11.7% reduction in memory power consumption and 4.2% performance improvement on a quad-core system running NAS [60] and PARSEC [6] applications with 4GB DRAM and 32GB PCM, compared to the standard auto-refresh scheme. Compared with a previously proposed refresh-reduction scheme [5], *Refree* can save main memory power by 3.1% on average, with a negligible 0.2% performance loss.

To reduce DRAM refresh and background power in a hybrid DRAM/PCM main memory (i.e., DRAM cache plus PCM) in a mobile device, we present a scheme called *NEMO* in Chapter 4. Mobile devices run on small batteries with limited capacities. Hence, energy consumption is an important factor that determines the usability duration of a mobile device. Main memory consumes a significant portion of the total system energy especially when the mobile device is idle, which is the case most of the times [13]. Moreover, in current and future systems, it is very likely that users tend to run various mobile applications simultaneously, which further increases the aggregate memory demands of mobile devices. Therefore, optimizations on memory energy consumption have become even more critical in those devices. Our proposed scheme, *NEMO*, powers off as many power-hungry DRAM components as possible, as long as it does not impact the performance. Specifically, when the mobile device is in idle state, only a selective set of data that is critical to performance is kept in a single DRAM rank, while the rest of data is stored in PCM, so that the remaining DRAM ranks can be powered off. To do so, *NEMO* classifies memory pages based on their usage frequency and recency into hot and cold. Then, it places the hot memory pages that are more likely to be re-used in future in the DRAM, which has lower access latency compared to PCM, and stores cold memory pages in PCM, which has near-zero idle power. In addition, *NEMO* predicts the number of DRAM ranks that need to be powered on when the mobile device becomes active for further energy saving. Our simulation results reveal that *NEMO* achieves on average, 10.2% reduction in memory power consumption and 1.7% performance

improvement on a system running various combinations of *Moby* benchmark applications [25] with 128MB DRAM and 1GB PCM, compared with the approach that simply puts DRAM into self-refresh low-power mode during idle state.

The main shortcomings of non-volatile memories are mostly related to the write operations; the latency and energy consumption of writing to an NVM is much higher than those of reads. To deal with the high overheads of write operations in NVM-based main memories, there are two common types of solutions. First category is to minimize the impact of writes on performance by optimizing the NVM architecture. Second category is reducing the total number of writes sent to the NVM main memory by modifying the Last Level Cache (LLC) management policies [80]. To alleviate the write-related overheads of a PCM-based main memory, in Chapter 5, we present a scheme called *WALL* [63, 64], which falls into the second category. *WALL* improves performance, energy efficiency, and lifetime of a PCM-based main memory system by reducing the number of LLC writebacks. In Chapter 5, we first investigate the writeback behaviour of LLC sets and show that writebacks are not uniformly distributed among sets; some sets observe much higher writeback rates than others. Then, we propose a writeback-aware set-balancing mechanism that employs the underutilized LLC sets with few writebacks as an auxiliary storage for the evicted dirty lines from sets with frequent writebacks. We also propose a simple and effective writeback-aware replacement policy to avoid the eviction of the dirty blocks that are highly re-used after being evicted from the cache. Our experimental results show that *WALL* achieves an average of 30.9% reduction in the total number of LLC writebacks, compared to the baseline scheme, which uses the LRU replacement policy. As a result, *WALL* can reduce the memory energy consumption by 23.1% and enhance PCM lifetime by 1.29×, on average, on an 8-core system with a 4GB PCM main memory, running memory-intensive applications.

To benefit from the total capacity of both DRAM and NVM and their aggregate bandwidth in a hybrid DRAM/NVM main memory, DRAM can also be used as part of the OS-visible main memory [14, 36, 37, 67, 74, 75]. Such organization is called a “flat address space hybrid main memory”. In a flat address space hybrid memory, DRAM has a limited capacity. Moreover, memory access behaviour of programs changes during execution. Hence, to take advantage of performance benefits of DRAM, data may need to be migrated (i.e., swapped) between DRAM and NVM. Since hardware requires meta-data storage to keep track of the migrated data, migrations are typically performed at a coarse granularity (e.g., memory pages). Existing schemes on flat hybrid memories typically partition the memory space into “swap groups” and allow only DRAM (i.e., fast memory) and NVM (i.e., slow memory) pages that belong to the same swap group to be swapped with each other. However, within a given interval, a swap group may contain more “high frequently accessed” pages than the number of DRAM segments (i.e., each segment contains a memory page) assigned to it (i.e., “swap group associativity”). This can cause frequent back and forth migrations of those pages between DRAM and NVM. Meanwhile, pages in another swap group may all be “low frequently accessed”. Hence, most of the page migrations, which are very costly in terms of performance and energy, can be avoided by dynamically adjusting the structure of the swap groups based on programs behaviour. To address the limitation of statically structured swap groups, we propose *DynaSwap* in Chapter 6. Specifically, *DynaSwap* dynamically associate swap groups with each other in such a way that a swap group with many high frequently accessed pages can benefit from the DRAM space of a swap group with low frequently accessed pages. In other words, unlike previous studies that create swap groups solely based on the physical address of memory pages, we try to create swap groups based on their access patterns. Assigning enough DRAM segments to a swap group based on its demand (i.e., its access patterns in the current interval) can improve performance and energy efficiency of the memory system by reducing the number of unnecessary swaps. Our experimental results show that *DynaSwap* can efficiently

utilize DRAM capacity and improve the overall performance and main memory energy efficiency by 30.1% and 13.5% on average, respectively, compared to a state-of-art baseline design.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Main Memory Organization

Conventional memory systems are organized hierarchically (Figure 1). The highest level of the hierarchy is "channel"; each channel can operate independently from other channels. A channel contains one or more "ranks". Ranks in a channel share the channel bandwidth but can operate in parallel (i.e., rank-level parallelism). Moreover, each rank contains one or more "banks". Banks can also operate in parallel (i.e., bank-level parallelism). However, bank-level parallelism is restricted by both the channel bandwidth and the resources shared between banks in a memory device (e.g., device power). Finally, a bank is a two-dimensional array of memory cells; consisting of many "rows" and "columns".

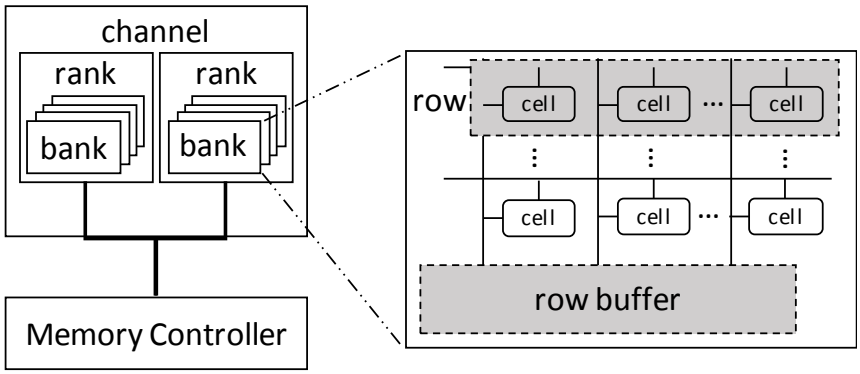


Figure 1. The organization of memory system.

On an access to a bank, the row holding the requested data is first copied into a “row buffer” (row is “activated”). In this way, the subsequent requests to the row will be served promptly from the row buffer (i.e., row buffer hit).

2.2 DRAM Refresh Operations

Dynamic Random-Access Memory (DRAM) has been the dominant technology for building main memories for several decades. The storage element of a DRAM cell is a capacitor that stores data in the form of a small electrical charge. The DRAM capacitive cells are naturally leaky and lose their data over time. To prevent data loss from happening, DRAM devices require periodic refresh operations. The process of refreshing restores the charge leaked from each individual cell of a row by “activating” or “opening” the row. Typically, the refresh operation for each DRAM row is performed once every 64ms (32ms at high temperatures) as specified by JEDEC standards [28, 29]. This time duration is called “retention time” or t_{RET}.

In commodity DRAMs, two modes of refreshing are supported: 1- “Burst Refresh”: In this mode, with every refresh command from the main memory, all DRAM rows undergo refresh in succession. This scheme was used in early generations of DRAM and has one major drawback: refreshing in a bursty fashion degrades system performance by delaying the regular accesses to the memory module for a long time. 2- “Distributed Refresh”: This is employed in current DDR devices to avoid the long latency of burst refreshing. In this mode, rows of a bank are divided into 8K groups (called “refresh groups”) and each group is refreshed within a 7.8μs (3.9μs at high temperatures) time interval (called “refresh interval” or t_{REFI}). Modern DRAM controllers send an auto-refresh (AR) command to the DRAM device once every t_{REFI}, and then the device decides which rows to be refreshed using an internal refresh counter. The time that the device takes to

complete the refresh operation is called “refresh cycle time” *or* t_{RFC} and is proportional to the number of rows per refresh group.

Refresh operations add significant power and performance overheads to the system. The power overhead comes from the energy consumed to activate the rows. The performance cost mainly results from the inaccessibility of a bank to regular requests while performing refreshes. Alas, these negative effects are expected to exacerbate as DRAM density increases. As the number of rows per bank increases, so does the number of rows that are refreshed with each auto-refresh command. In fact, while other timing parameters of DRAM remain almost constant from one generation to another, t_{RFC} is growing exponentially [5, 59].

To improve the performance and energy efficiency of DDR devices, some prior studies have attempted to reduce the number of refresh operations by considering either the access recency [1, 20, 26] or the retention time [5, 46] of rows. The proposed techniques in [20, 46] perform refreshing at a row-level granularity. In old asynchronous DRAM devices, the memory controller was able to perform row-level refreshing by issuing RAS-only commands [57]. However, this method requires additional power for sending row addresses on the bus and thus is deprecated by JEDEC standards. In current devices, issuing an ACTIVATE command followed by a PRECHARGE command is the only way to implement row level refreshing.

The access-aware refresh reduction schemes are based on the fact that a row needs to be opened and thereby is automatically refreshed on regular accesses. In other words, for up to t_{RET} after accessing a row, the integrity of its data is guaranteed [20]. Moreover, a refresh operation to a row that doesn’t hold valid data is utterly unnecessary and can be skipped [26]. In the meantime, the leakage current differs among the cells of a DRAM device. In general, a cell can be either “leaky” or “normal”. Of all the cells, very few are leaky and lose their data faster, whereas

the rest are normal and retain their stored charge for a longer period of time [23, 33]. The time period that a cell can safely preserve its stored data is referred to as its "retention time". The standard retention time (t_{RET}) is in fact the retention time of the leakiest cell on the device. However, a row that doesn't have any leaky cell is "strong" and can be refreshed at a slower than nominal rate. Hence, previous retention-aware refresh reduction schemes schedule the refresh operations for every single row [46] or a refresh group [5] based on the cell's retention time information.

2.3 Mobile DRAM Power Management

In power management, a rank is the smallest unit that can be controlled to operate in several different power states. In general, the power consumption of a rank can be classified into two main categories, "active" and "background" [49]. The power required to serve memory reads and writes is the active power, while the background power is consumed all the time even without any memory accesses. Background power contributes significantly to the total DRAM power [49]. In fact, in addition to the periodic refresh operations, the other components of a rank, such as row and column decoders and sense amplifiers, are also power hungry [42]. Hence, different low-power operating states are provided by SDRAM architectures to disable some of these sources of energy consumption and reduce the background power.

The power consumption of a rank varies among different power states. To reduce the background power, a rank can be put into a low power state when it is idle. However, to power up the rank and restore it to the active state, its disabled hardware components need to be reactivated. Transition among different power states incurs latency and energy penalties. Basically, a power state is described by its power consumption and resynchronization time, the time that it takes to exit the power state and go back to the active mode. A power state that

TABLE I. Power states for a 1GB LPDDR2.

Power States	Power (mW)	Comments
ACT	20.5	DRAM power for ACT/PRECHARGE commands
ACT_PDN	9.4	Background power used during active power-down
PRE_PDN	3.4	Background power used during precharge power-down
SREF	1.4	Self-Refresh (SR) standby power
DPD	0.2	Deep Power-Down (DPD) standby power

consumes less energy typically has higher resynchronization time. The major power states provided by Low-Power DDR2 (LPDDR2) are summarized in TABLE I. The power consumption values are calculated with DRAM System Power Calculator [56]. The current and voltage parameters are obtained from a LPDDR2 datasheet [55].

2.4 Phase Change Memory

Phase Change Memory (PCM) is a type of non-volatile memory technology that has been explored as an alternative to DRAM due to its better scalability, lower leakage energy and non-volatility [66, 72]. One of the key advantages of PCM over DRAM is its scalability. In fact, PCM can scale down to feature sizes as small as 8nm [27]. Another benefit of the PCM technology is its zero-leakage power. The resistive characteristic of PCM cells allows them to retain their data for a long time with no need for periodic refreshes. Also, PCM exhibits much better static energy parameters compared to DRAM [88]. Despite of its many advantages, PCM also has a number of drawbacks. Compared to DRAM, each read/write operation in PCM has longer latency and consumes more energy [84]. Further, a PCM cell has much lower write endurance. TABLE II summarizes PCM and DRAM attributes.

TABLE II. Comparison between DRAM and PCM [89].

Memory Technology	NVM	Idle Power	Read Latency	Write Latency	Endurance
DRAM	No	~0.1 W/GB	50 ns	~20-50 ns	∞
PCM	Yes	≪ 0.1 W	50-100 ns	~1 us	10^8

2.4.1 Impact of Write Operations on PCM

Phase change memory has higher write access energy and latency than DRAM (2 to 8 times [39]) and limited write endurance. A previous study [2] has shown that the long-latency write operations can increase the effective latency of read requests by 1.2 to 1.8 times. In addition, the number of write operations in PCM affects its lifetime. The PCM-based memory system lifetime can be calculated as [70]:

$$\text{System Lifetime} = \text{Y (years)} \approx \frac{\text{w}_{\text{max}} \cdot \text{S}}{\text{B}} \cdot 2^{-25}$$

In this formula, B is the write traffic (or write rate, GBps), Y is the maximum number of years that a PCM with size S and cell endurance of w_{max} can last. The effect of reducing the write traffic on the expected PCM lifetime is shown in Figure 2. It should be noted that the results of the figure are independent of S and w_{max}. It is also assumed that the writes are distributed uniformly across the entire PCM memory.

2.5 Hybrid Main Memory

Non-volatile memory (NVM) technologies such as PCM, ReRAM and STT-RAM have been explored as potential replacements to DRAM due to their higher density, better scalability, and

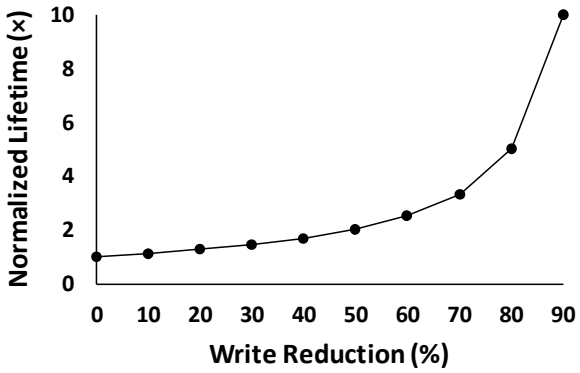


Figure 2. Impact of reducing write traffic on PCM lifetime.

lower leakage power [81, 83]. However, NVMs have two major limitations and cannot replace DRAM without any modifications. First, NVMs have higher access latency and energy compared to DRAM. Second, NVM cells have limited write endurance, which can adversely impact their lifetime [15, 77, 85].

Hybrid main memories, which incorporate both DRAM and NVMs, have been proposed as a potential solution that can benefit from both technologies. In a hybrid main memory, DRAM, which has lower read/write latency and energy, can serve as a cache for the NVM [41, 51, 70, 84]. In this case, the entire working set of the application initially resides in NVM. Then, upon every DRAM cache miss, the requested data is copied from NVM into the DRAM cache. DRAM can also be used as part of a flat address space hybrid main memory [14, 36, 37, 67, 74, 75] to enhance the overall OS-visible capacity of memory. The flat address configuration is beneficial for data-intensive applications (i.e., with large working sets) when DRAM space constitutes a large fraction of the overall memory capacity [14, 36, 37, 67, 74, 75].

2.5.1 DRAM as a Cache

The benefits of using a small DRAM as a conventional cache in a DRAM/PCM hybrid main memory was first investigated by Qureshi et al. [70]. In that system, the DRAM component of the main memory is organized as a 16-way set associative cache with cache blocks as large as a memory page. In general, the major challenges of designing a DRAM cache structure are minimizing hit latency, tag storage overhead and complexity of page placement and migration. Prior studies have proposed various DRAM cache structures based on their design goals.

To design a power-efficient and high-performance hybrid DRAM/NVM main memory, three questions must be answered: First, what granularity of data movement between DRAM and NVM will satisfy system's demands best? Prior studies, which consider DRAM as a cache for either a DRAM-based main memory or NVM component of a hybrid main memory, migrate data either at the granularity of a memory page [43, 70, 71, 84, 87] or a processor's cache block [18, 22, 44, 48, 51, 69]. In [70], the migration of data from PCM to DRAM is performed in pages, but, upon evicting a row from DRAM, only the dirty columns of the row are written back to PCM to reduce the number of PCM writes. The DRAM row buffer locality can be effectively exploited in such "row-based" DRAM caches. Nevertheless, migrating data at a finer-granularity reduces the migration latency and prevents the subsequent memory requests to be blocked for a long time but with a higher management cost [22, 48]. Moreover, cache structures with blocks of the same size as a processor cache block manage the cache space more efficiently by minimizing overfetching.

Second, how to effectively manage caching overheads? In order to have fast accesses, previous study [70] uses a separate 1MB SRAM to store the caching information (i.e., tags, dirty/valid bits and LRU information) for a 16-way set-associative 1GB DRAM cache with 4KB blocks. This cache structure is simple and incurs low management costs. However, for the same size (1GB) finer-

grained DRAM cache (e.g., with 64B blocks) or a larger DRAM cache, the area overhead of a separate SRAM storage makes it impractical. To overcome this challenge, some prior studies [48, 51, 69] have proposed to store the caching information in the DRAM cache itself. For example, *LH-cache* [48] places an entire cache set in a DRAM row and reserves three out of 32 columns of the row for tags; and *Alloy cache* [69] integrates both tag and data into a single entity and transfers an entity in five bus cycles. However, the tag storage overhead of such tag-in DRAM caches is still large and occupies 12.5% (i.e., 128MB per 1GB) of the total DRAM cache space [69]. To address the aforementioned challenge, a recent study [44] has proposed a page-based, fully associative, *tagless DRAM cache* that completely removes the cache management data structures. To eliminate cache tags, the conventional TLB is replaced with cache-map TLB (cTLB) that stores virtual-to-cache, instead of virtual-to-physical address mappings. The cTLB uses the same hardware resources as the original page table. In addition, the cache replacement mechanism is implemented by exploiting a Global Inverted Page Table (GIPT), which stores the cache-to-physical mappings for the cached pages, and a Free Queue that maintains a list of cache blocks to be evicted. This cache structure has zero tag storage overhead, low average access time (i.e., high hit rate due to being fully associative) and high energy efficiency [44].

Finally, the last question: what data is worthy of occupying DRAM cache space? This question is answered by previous studies in two different ways. The first one is based on the access pattern of rows [43, 71, 87]. Those schemes predict that a row is more likely to be reused in near future if it has been frequently accessed in the past (i.e., hot page). Hence, they migrate hot pages from NVM to DRAM. The second one is based on the number of row buffer misses of NVM rows [84]. This scheme counts the number of row buffer misses for the recently accessed rows in PCM and migrates to DRAM the rows that have been highly reused but mostly missed in row buffers.

2.5.2 DRAM as Part of Memory

In a flat address hybrid memory, fast and slow memory modules (e.g., DRAM and NVM) are organized in parallel as two separate storage elements, each holding a portion of the working set [43, 71, 87]. A flat address space hybrid main memory can be managed by software to create a heterogeneous main memory. However, in such memory systems, the difficult task of page placement and migration must be handled by OS, which also needs hardware support to gather per-page access statistics [44]. Hence, previous studies have mostly focused on hardware-managed techniques [14, 36, 37, 67, 74, 75]. To improve performance and energy efficiency of a hardware-managed flat hybrid memory, pages can be swapped between the slow (e.g., an NVM) and fast (e.g., DRAM) memory modules to ensure that frequently accessed pages are placed in the fast memory.

Sim et al. [75] proposed a hardware-managed flat hybrid memory design, which we will refer to as *PoM* (i.e., Part of Memory). PoM performs migrations at the granularity of pages (i.e., 2KB) and uses a "remapping table" to keep track of the pages it swaps between the fast and slow memory. To keep the bookkeeping costs practical, migrations are only allowed within sets of pages, which we will refer to as "swap groups". In other words, multiple pages of slow memory and a page of fast memory form a swap group, and only one page of the group can reside in the fast memory at a time. PoM monitors the memory accesses with a counter per swap group and initiates a page migration when the counter reaches a threshold.

In [14], a cache-like flat memory management scheme called *CAMEO* is proposed to close the gap between cache and flat memory architectures. CAMEO works similarly to PoM but does so at the granularity of cache lines (i.e., 64B). It places the remapping meta-data next to data within the same row and initiates a migration upon every access to the slow memory. The major

limitation of both PoM and CAMEO is that multiple frequently accessed pages/lines may exist within the same swap group. In such case, the direct-mapped structure of both designs (i.e., multiple rows/blocks in the slow memory are mapped to a row/block in the fast memory) forces pages/lines in a swap group to compete for a single row/block of the fast memory. This can cause frequent back and forth movement of those pages/lines between the fast and slow memory modules.

Using smaller swap granularities enables CAMEO to keep the swap bandwidth low. However, it requires higher meta-data storage and eliminates the opportunity to benefit from programs spatial locality. To optimize the swap granularity, *SILC-FM* [74] supports sub-block interleaving between two pages in the fast and slow memories (i.e., migration granularity ranges from 64B to 2KB). It also improves the migration flexibility by making the direct-mapped structure of swap groups into set-associative. However, while PoM and CAMEO allow a page to reside anywhere within a swap group (i.e., fast swaps), SILC-FM requires the original mapping in a swap group to be restored before each swap (i.e., slow swaps).

The *MemPod* scheme proposed in [67] further improves migration flexibility by making it fully-associative. More specifically, it partitions the memory space into large clusters and allows an any-to-any page swap between the fast and slow memories within a cluster. To predict the future hot pages for migration to the fast memory, MemPod employs the *Majority Element Algorithm* (*MEA*) [31] and performs migrations at the granularity of pages after pre-defined time epochs. The major limitation of MemPod is that its fully-associative structure comes at the cost of a significant increase in the meta-data area overhead.

A recent study, *PageSeer* [37], adapts correlation prefetching (i.e., the pages that are accessed in some order at some point are likely to be accessed in the same/similar order in the future) to identify the pages that are to be accessed soon and exploits page walks to migrate those pages

into the fast memory ahead of time. It associates an access counter with each page and triggers a swap for the page when the counter reaches a threshold. The swap restriction of PoM is also relaxed by making the swap groups set-associative. However, this design suffers from slow swaps. Moreover, PageSeer also assigns a limited number of statically-specified fast memory rows to all the swap groups while swap groups can have different access behaviours.

2.6 Other Related Works

2.6.1 Reducing Overheads of DRAM Refresh

Some recent studies have tried to alleviate the power and performance overheads of refreshing in DRAM-based main memories or eDRAM-based on-chip caches. Some existing schemes such as *Flikker* [47], *RAIDR* [46] and *Flexible Auto-Refresh* [5] reduce unnecessary refresh operations by taking DRAM rows retention time into account. More specifically, Flikker allows the programmer to divide the application data into critical and non-critical portions. It then refreshes the non-critical part of the memory at a lower than nominal rate and the critical part at the regular rate. The RAIDR scheme groups rows into bins based on their retention time and refreshes bins at different rates. Since most of the DRAM cells and thus their corresponding rows are strong and need to be refreshed at lower rates, RAIDR is able to remove a large percentage of refresh operations. Flexible Auto-Refresh skips the unneeded refreshes while it performs the remaining refresh operations using the default auto-refresh mechanism. To do so, the architecture of the memory controller is modified to enable reading, writing and incrementing the refresh counter in a DRAM device.

Some studies reduce the number of refresh operations by considering the access pattern of DRAM rows. Among those are *Smart Refresh* [20] for off-chip DRAM main memories and *Refrint*

[1] for on-chip eDRAM caches. The smart refresh scheme eliminates refreshes to the accessed rows. This scheme employs a time out counter for each row that is reset to its maximum upon an access or refresh to the row. The Refrint scheme refreshes only the data that is likely to be used in near future and has not been accessed recently. Generally, DRAM can be used as either an off-chip cache in a hybrid main memory [70] or an on-chip cache in 3D CMPs [69]. Moshnyaga et al. [58] proposed a software-based scheme to reduce the off-chip DRAM cache refresh energy in a DRAM/Flash memory system. This technique recognizes active and non-refreshed banks based on the access pattern of their data and disables refresh to the banks that contain only non-modified data in a given time period. For on-chip eDRAM-based caches, memory access behaviour is exploited in [1, 10]. They postpone refresh to the rows that are accessed intensively and bypass refresh to the dead cache lines.

2.6.2 Data Placement and Migration in Hybrid Main Memories

One of the challenging issues related to the hybrid main memories is to decide which data to place in which memory component. For hybrid memory systems that consist of multiple different technologies, managing data placement and movement between the two technologies is a major challenge. Some of the previous studies including [71, 87] take page accesses into consideration and migrate hot pages from PCM to DRAM. The memory controller uses a modified multi-queue algorithm for determining hot and cold pages. In addition, since row buffer misses are much more costly in terms of energy and latency in PCM, Yoon et al. [84] migrate data that frequently misses in the row buffer from PCM to DRAM. Based on their cost-benefit analysis, they determine a dynamic threshold to decide whether a row has low row buffer locality or not.

2.6.3 Reducing Energy Consumption of Main Memory in Mobile Devices

One of the prime consumers of energy in mobile systems is the main memory. To reduce main memory's power consumption, a prior study [17] has investigated the effectiveness of some energy management mechanisms on smartphones. In that work, Power-Aware Virtual Memory (*PAVM*) [24] and Immediate Power Down (*IPD*), Immediate Self Refresh (*ISR*) [21] schemes are discussed. The PAVM scheme is an OS-level approach that maps all pages of an application into a few ranks and turns on only those ranks when the application is running. However, one major limitation of PAVM is that when multiple applications are simultaneously running on a device, the memory pages of each application may reside in a different rank. In fact, the worst-case scenario happens when the applications that are frequently launched together, each gets assigned to a distinct memory rank. In that case, all the memory ranks need to be turned on most of the times. The IPD/ISR mechanism immediately puts an active rank into a lower power state after serving a memory request. Though this scheme could work well for workloads with lower memory intensities, it can have negative impacts on system energy efficiency and performance for the memory-intensive applications.

A recent study [13] pointed out that devices such as smartphones and tablets are idle most of the times. To save refresh power during the long idle periods, Chou et al. [13] proposed Morphable ECC (*MECC*). It reduces refresh rate in idle mode by using strong error correction, while in active mode, it prevents performance degradation by using weak error correction. There are two main downsides of MECC. First, current mobile devices are not supported with ECC. Thus, MECC that requires the same area overhead as traditional SECDED (i.e., 12.5%) is not the best solution for compact mobile devices. Second, MECC does not power off the DRAM main memory in idle mode. Thus, only up to half of the DRAM idle power would be saved. The limited memory size of portable devices is another challenging issue that has a direct impact on user experience. The LRU-based

task killing policy of Android platforms results in restarting a large number of applications when the available memory capacity is not sufficient to accommodate all the applications' datasets. To deal with this problem, some prior studies have benefited from non-volatile memory technologies such as PCM either as a swap area or in a hybrid main memory [7, 17, 34, 88].

2.6.4 Reducing Overheads of Write Operations in NVMs

Many recent studies have focused on mitigating the overheads of write operations in PCM-based main memories. Lee et al. [15] proposed a scheme called *eager writeback*, which writes the LRU dirty cache lines back into the main memory before their eviction to improve system performance. A variation of eager writeback is proposed in [25] to improve PCM performance by early and eagerly writing back the long latency SET operations. In [24], the concept of write cancellation is introduced to prioritize reads over writes to immediately service the incoming reads; this scheme cancels the conflict writes. The performance overhead of write operations is alleviated by parallelizing read or write accesses with an ongoing write in [1]. Moreover, Zhou et al. [38] developed a non-blocking PCM bank design which aims to service subsequent reads or writes in parallel with an on-going write. Xia et al. [32] explored the possibility of removing the unmodified data from a single write and then, merging modified data of multiple writes to be sent within one write request to improve PCM write bandwidth. In addition, Zhang et al. [37] has shown that only a small portion of the main memory is frequently accessed in a given time period. Based on that observation, their scheme records and predicts the memory regions' write frequencies in order to select a proper write latency (i.e., the number of SET iterations) for every incoming memory write operation to improve system performance and memory lifetime.

There are also many studies on reducing the write overheads of other types of non-volatile memory technologies such as STT-RAM and ReRAM. Zhang et al. [36] proposed a scheme called

Mellow Writes that extends memory controllers to selectively perform slow writes to reduce the impact of writes on endurance and performance of the ReRAM-based memories. Kultursay et al. [13], investigated the possibility of replacing DRAM with STT-RAM for main memories. In that work, STT-RAM write overheads are reduced by bypassing the row buffer writes and tracking dirty blocks to perform partial writes within a row. Typically, STT-RAM is considered as an alternative to SRAM caches or used in SRAM-NVM hybrid caches. To alleviate the write overheads of STT-RAM in a hybrid cache, Wang et al. [29] presented an adaptive placement and migration policy based on the access pattern of different classes of write operations in LLC. Wu et al. [31] partitioned hybrid cache into read and write regions and migrated cache blocks within the cache to mitigate the write overheads of STT-RAM.

Some studies have proposed techniques for reducing the number of LLC writebacks to the non-volatile component of a hybrid main memory consisting of an NVM and DRAM [6, 35]. In reference [6], a miss penalty-aware LRU-based cache replacement policy, called *MALRU* is proposed to consider the asymmetry of cache miss penalty on DRAM and NVM. The MALRU scheme keeps the high-latency NVM blocks as well as the low-latency DRAM blocks with good temporal locality in a reserved area to protect them from being evicted from the LLC. Similarly, Zhang et al. [35] proposed a writeback-aware LLC management scheme for hybrid main memory systems to reduce the number of writebacks to NVM by improving the hit ratio of the NVM memory blocks in the cache. It should be noted that these techniques are only applicable for hybrid main memories. To balance the pressure on cache sets, the set balancing cache (i.e., *SBC*) proposed in [27] tries to associate sets with maximum “saturation counters” (i.e., sets with large datasets) with sets with small saturation counters. Though SBC reduces cache miss rate, it is not always able to reduce the number of writebacks. There are some sets with saturation counters smaller than the maximum value, which write back more frequently than those with maximum

saturation counter values. However, the SBC scheme shares resources between two sets only when a set's saturation counter reaches its maximum.

CHAPTER 3

A REFRESH-FREE HYBRID DRAM/PCM MAIN MEMORY SYSTEM

3.1 Introduction

Technological advances have scaled up the size of applications working sets drastically. This growing trend is expected to continue even faster in future. Moreover, the number of cores that share a single memory system is increasing on chip multiprocessors. Hence, the aggregate amount of data that the main memory must be able to support is becoming larger over time. This makes main memory an even more critical component in modern computing systems. Meanwhile, DRAM is facing two main challenges. First, its scalability is limited. Second, it requires periodic refresh operations, which consume considerable amount of time and energy. Recently, scalable non-volatile memory technologies such as PCM have emerged as DRAM alternatives. However, PCM, as other NVMs, has a number of shortcomings. First, PCM has higher access latency and energy compared to DRAM. Second, PCM cells have limited write endurance.

To overcome the shortcomings of both technologies, hybrid DRAM/PCM main memories have been proposed. Such designs typically consist of a modest sized off-chip DRAM cache for a much larger PCM storage. However, in future, satisfying system demands on performance with such a small DRAM cache, no matter how well managed, might be impossible. On the other hand, using a larger DRAM cache can also incur significant performance and energy overheads due to DRAM refresh operations. In this study, we present *Refree* [66], a scheme that eliminates DRAM refresh operations in a hybrid DRAM/PCM main memory system.

The basic idea behind *Refree* is to evict a row from DRAM if at any point the row has to be refreshed. In fact, as will be shown later in this chapter, such rows mostly hold nonvaluable (i.e., useless in near future) data. Hence, there is no need to refresh and keep those rows in the DRAM cache. In addition, a recently accessed row has already been "refreshed" by the access and does not need to be refreshed either. To keep the data integrity, the dirty columns of the row that is being evicted from the DRAM cache must be written back to the PCM. Since PCM has long write latency, we propose a scheme that distributes writebacks of a dirty DRAM row over an epoch time (i.e., 128ms) instead of performing them all at once, to prevent long-time blockage of other requests (i.e., DRAM read misses) to the PCM storage. Generally, to remove DRAM refresh operations completely, *Refree* considers two refresh-reducing factors that have been proposed in previous studies [20, 46]: the access pattern and retention time of DRAM rows. More specifically, each valid row is monitored for time periods equal to half of its retention time. Upon any access to the row within each of such time periods, the row will be marked as accessed. The access actually gives immunity to the row from being refreshed or invalidated at the end of the time period. On the other hand, a non-accessed row will be invalidated from the DRAM cache or if it is dirty, be written back to PCM.

Our experimental results show that *Refree* reduces the hybrid main memory's power consumption with negligible performance impact compared to existing refresh-reduction techniques. For a quad-core system with a hybrid main memory system of a 4GB DRAM cache and 32GB PCM, *Refree* reduces the memory power consumption of NAS [60] and PARSEC [6] applications by 11.7% and 3.1% on average, compared to baseline auto-refresh and one of the most effective refresh-reduction schemes, respectively. *Refree* improves performance by 4.2%, on average, compared to baseline auto-refresh while incurs only a negligible performance overhead, by 0.2% on average, compared to a recent refresh-reduction scheme.

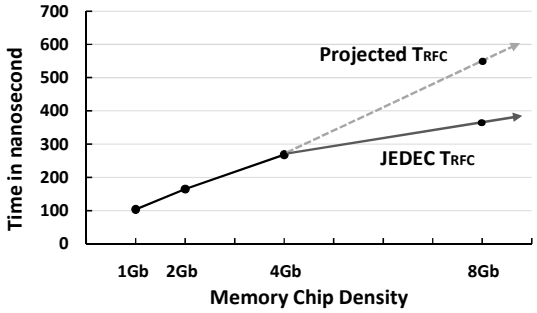


Figure 3. Variation and projection of t_{RFC} for different memory chip densities [59].

3.2 Motivation

The adverse effects of DRAM refresh are expected to aggravate with each generation of technology. It is predicted that refresh accounts for 50% of throughput loss and 50% of the total energy consumption in a future 64GB DRAM system [5, 46]. Figure 3 shows the variation and projection of t_{RFC} (i.e., latency of each auto-refresh operation) for different memory chip densities for Micron DDR3-800 [59]. The significant growth of t_{RFC} with the increase in memory size is depicted in the figure. For instance, the value of t_{RFC} for a 32GB DDR4 is 1.34 times more than that of a 16GB memory [5]. The auto-refresh operation energy is also proportional to t_{RFC} [54]. Therefore, using larger DRAM caches, although necessary, can incur significant performance and energy penalties due to DRAM refresh operations.

In addition, storing memory pages that are rarely accessed in the DRAM cache is only a waste of energy and the precious cache space. Hence, our **main goal** in this work is to address these two issues simultaneously by eliminating DRAM cache refresh operations and putting those rarely accessed pages away in PCM. In this work, a hybrid main memory, which includes a DRAM cache and a larger PCM storage, is considered. *Refree* aims to eliminate DRAM refresh operations

completely. Considering that the refresh cost is more prominent in larger DRAMs, *Refree* is expected to become even more effective for future DRAM/PCM hybrid memory systems. It is also worth mentioning that, in this work we have considered PCM as NVM, but our scheme is also applicable for other types of NVMs.

3.3 *Refree* Overview

To eliminate DRAM refresh operations, *Refree* considers both the access pattern of rows and their retention times at the same time. In general, *Refree* performs a periodic row-level monitoring and classifies DRAM rows into two categories, "accessed" or "non-accessed". An accessed row does not need to be refreshed because it has recently been activated by a request and thus refreshed; and a non-accessed row does not need to be refreshed either because it has not been accessed for a long time and thus is very likely to hold inactive data and can be evicted from the DRAM cache. In this way, all refresh operations on the DRAM cache will be eliminated.

In our work, DRAM rows are monitored for time intervals equal to half of their retention times for any accesses. We believe that this time interval (i.e., 128ms) is long enough so that if a row is not accessed within it, we can conclude that the row is "inactive" and thus there is no need to keep it in the DRAM cache. Our experimental results also support such conclusion. An evicted row must also be written back into PCM if it has been updated in the DRAM cache. Initially, all rows are marked as non-accessed. The status of a row changes to accessed at its insertion to the DRAM cache or receiving a regular memory request from the CPU.

The skipping of refresh operation for an accessed row or the eviction of a non-accessed row is performed at an "evaluation point" of the row, which is when a DRAM row's current monitoring process ends, and next monitoring process starts. The monitoring process restarts when the row's

current epoch time, which is set to half of the row's retention time in our experiment, expires. We will discuss the monitoring process in detail next.

3.4 Monitoring Process

A row must be monitored after being inserted in the DRAM cache. If the row receives no requests from one evaluation point to the next, it may need to be evicted from the cache. The main challenge here is to determine the time interval between two consecutive evaluation points of each DRAM row. This interval must be long enough so that if a row remains non-accessed during the interval, it could be concluded that the row is not needed in the DRAM cache and keeping it in PCM is sufficient. Otherwise, highly active rows might be evicted from the cache by mistake, causing significant power and performance degradations. On the other hand, the interval cannot be too long for keeping data integrity. A row, even if remains non-accessed during the interval, must be able to safely retain its data. To effectively address this issue, we have brought DRAM rows' retention time into the picture. In fact, majority of rows in a DRAM device are strong and have a very long retention time of 256ms. A previous study [46] has shown that in a 32GB DRAM system only up to 28/978 rows have a 64ms/128ms retention time while the rest of the rows have a 256ms retention time.

Ideally, once a row is accessed, its data remains undamaged for a time interval equal to its retention time. However, keeping track of the exact elapsed time is impractical. Two of the previously proposed access-aware, refresh-reduction schemes have tackled this problem using two different techniques. The first one considers a counting-down timeout counter for each row, which is reset to its maximum upon any access [20]. The second one divides every retention time into a number of steps and within one retention time, refreshes rows at the same step as that they were accessed at in the previous retention time [1]. In our work, to keep the hardware overheads

at minimum and guarantee the data integrity, we set the evaluation point at every row's half retention time. This means that for majority of the rows, we are considering a time interval as long as 128ms to find out whether the row must be kept in the cache or not. More specifically, for each row, its access status from one of its evaluation points to the next is represented with a single access bit. The access bit of a valid row will be set to '1' upon every access and reset to '0' at every evaluation point.

Each row must be evaluated at its evaluation points: the row's access bit is checked. If the access bit is '1', the monitoring process will restart for the row; otherwise, the row will be evicted from the DRAM cache and written back to the PCM if dirty. Similar to a previous study [46], our scheme classifies the rows of each DRAM bank, based on their retention time, into three categories: rows with retention time of 64ms (weak), 128ms (mediocre) or 256ms (strong). Thus, weak rows will be evaluated every 32ms, mediocre rows every 64ms and strong rows every 128ms.

The group of rows that are evaluated in each refresh interval is called "evaluation group". To form the evaluation groups, we divide the total number of rows in each DRAM bank into 8K groups. The evaluation groups are classified into three categories: (1) "strong": evaluation groups that include strong rows only, (2) "weak": evaluation groups that include at least one weak row and (3) "mediocre": evaluation groups that consist of at least one mediocre row but no weak rows. The proposed monitoring process is depicted in Figure 4. To determine the evaluation points on the time line, we consider a two-bit "period counter" that is incremented every 32ms. The first refresh intervals after incrementing the period counter (i.e., every 32ms) are assigned to the weak evaluation groups. We call these refresh intervals "weak evaluation intervals". In each weak evaluation interval, all the valid rows of a weak evaluation group are evaluated. A weak evaluation group may also contain mediocre and strong rows. Ideally, those rows can be evaluated less frequently. However, to minimize the hardware and management overheads, *Refree* considers all

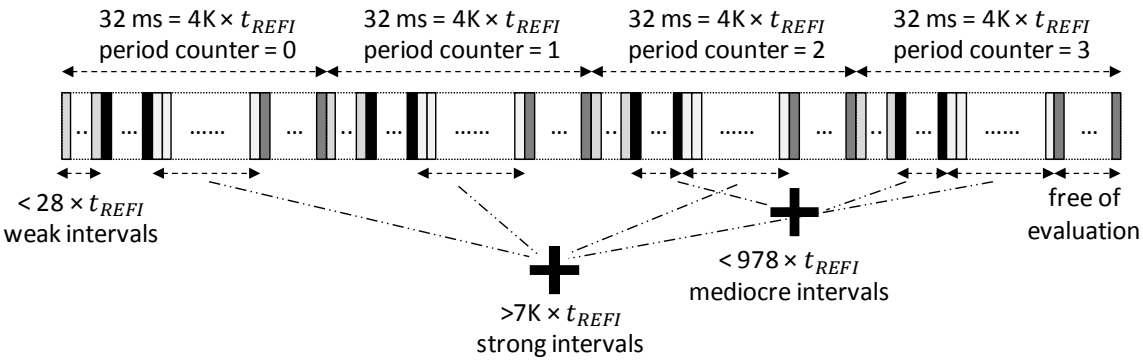


Figure 4. Different types of evaluation intervals in a 128ms time epoch.

rows in a weak evaluation group as weak. Note that most groups are strong groups. The immediate refresh intervals after the weak evaluation intervals are called "mediocre evaluation intervals", which are assigned to the mediocre evaluation groups. Generally, all the mediocre evaluation groups can be evaluated every 64ms when the period counter is even. However, to unify the assignment of evaluation intervals, *Refree* evaluates half of the mediocre evaluation groups every 32ms. To do so, a one-bit flag is used; the flag is initialized to the LSB of the period counter every 32ms (i.e., '0' when the period counter is even and '1' when it is odd). When a mediocre evaluation group is found, *Refree* evaluates it if the flag is '0' and then sets the flag to '1'; otherwise *Refree* skips that mediocre group and sets the flag to '0'. In each mediocre evaluation interval, all the valid rows of a mediocre evaluation group are evaluated. A mediocre evaluation group may also have some strong rows (but not weak rows). Ideally, those strong rows can be evaluated less frequently. However, similar to the scenario explained for the weak evaluation groups, *Refree* considers all rows in a mediocre evaluation group as mediocre to minimize hardware and management overheads. Finally, all the remaining vacant refresh intervals can be used for evaluating the strong evaluation groups ("strong evaluation intervals"). To unify the assignment

of evaluation intervals, *Refree* evaluates one-fourth of the strong evaluation groups every 32ms. To do so, a two-bit flag is used, which is initialized to the value of the period counter every 32ms. When a strong evaluation group is found, *Refree* evaluates it if the flag is '00' and then increments the flag; otherwise *Refree* skips that strong group and only increments the flag.

To determine the evaluation intervals types, a 2-bit counter called "category counter" is used. *Refree* considers an evaluation interval weak, mediocre or strong based on the category counter's value; '0' representing weak, '1' representing mediocre, '2' representing strong and '3' representing "free of evaluation". The category counter is reset to '0' every 32ms and is incremented after evaluating the groups that belong to the category represented by the counter's current value. To traverse the evaluation groups, a 13-bit counter called "general group counter" is used.

3.5 DRAM Cache Structure

Data can be stored in the DRAM cache at different granularities. The size of a DRAM cache line could be as small as a processor cache line or as large as a memory page. Generally, finer grained caching manages the cache space more efficiently but with higher management overhead. To the best of our knowledge, among the previously proposed DRAM cache structures, Alloy cache structure [69] might be the best candidate for an off-chip Giga-scale DRAM cache due to its minimum cache management overheads. The Alloy cache is a direct-mapped, tag-in DRAM cache that has a cache line size of a processor's cache line and transfers both data and tag as one entity.

In our work, writing back the dirty non-accessed rows to the PCM storage is needed to eliminate DRAM refreshes. Hence, it seems that it would be more beneficial to cache all or at least some of the columns of a PCM row in a single DRAM row (e.g., considering each DRAM row either

as a set or a single DRAM cache line). The reason is: in this way when a row needs to be written back, all the write requests sent to the PCM device will hit in the row buffer. However, our experiments indicate that the average number of dirty entities in each valid row of the DRAM cache with Alloy structure for the benchmarks shown in TABLE V is relatively small (i.e., 7 out of 28 entities). Furthermore, the experimental results show that most of the rows in an Alloy cache are active most of time, which reduces the number of evictions and thus writebacks. Hence, we use Alloy cache as our DRAM cache structure. We have also considered a small buffer, called *WBB*, for handling the eager writebacks sent to the PCM by the DRAM controller. The requests buffered in *WBB* will be sent out to the PCM when the PCM is idle or *WBB* is full. It is worth noting that *Refree* exploits Line Level WritesBacks (LLWB) proposed in [70] to alleviate the overheads of writing a DRAM row back to the PCM. This scheme writes back the dirty columns (i.e., dirty entities) of a row only.

For each row, a valid status bit is considered to determine the validity of a row in the DRAM cache. As stated before, *Refree* evaluates only the valid rows of the DRAM cache because it is obvious that invalid rows do not need to be refreshed at all. Similarly, one dirty bit is used to indicate whether a row is dirty or not. This bit is set for the row with at least one dirty entity. The total hardware overhead incurred by these status bits is only 16KB (2×64K bits) per bank.

3.6 Storage Overhead

TABLE III summarizes the storage overhead of *Refree* per DRAM rank. Note that our proposed approach does not need the information about each individual row's retention time. Hence, *Refree* uses the bloom filters of RAIDR [46] for storing the category of evaluation groups instead of row's retention time bins. As depicted in the table, the total storage overhead is almost 194 KB per rank or 0.02% of total DRAM capacity.

TABLE III. *Refree*'s total storage overhead.

Type	Storage
Row status (valid/dirty) information	16 KB
Row access information	8 KB
General group counter size	13 bits
Row counter size	16 bits
Mediocre groups flag	1 bit
Strong groups flag	2 bits
Category counter size	2 bits
Storage Overhead per Bank	~ 24.005 KB
Evaluation groups category information	1.25 KB
Period counter	2 bits
Total Storage Overhead	**~ 194 KB**

3.7 Results

3.7.1 Methodology

In this work, we model a quad-core processor using Gem5 simulator [19] integrated with NVMAIN [62], a cycle accurate main memory simulator designed to simulate emerging non-volatile memories at the architectural level. The system configuration of our experiments is shown in TABLE IV. We collect runtime statistics from the full-system simulations. The DRAM configurations (i.e., timing and current parameters) are obtained from [52]. The PCM configurations are generated by NVSIM [16] and CACTI [8]. Note that the cell parameters used in NVSIM are based on the projections by [12]. The benchmarks used in this study are chosen from NAS [60] and PARSEC [6] as depicted in TABLE V. Six NAS and three PARSEC workloads covering the range of memory footprints of the whole NAS and PARSEC suites are selected. For all the workloads, we use either sampled reference or native input sets to represent a real-world execution scenario and run the applications for two Billion instructions.

TABLE IV. System configuration.

Processor	4-core, 4.0 GHz, out-of-order	
L1 Cache	Private, 64KB per core, 8-way, LRU, 64B cache line, write-back, write allocate	
L2 Cache	Shared, 1MB, 8-way, LRU, 64B cache line, write-back, write allocate	
Memory Controller	Open page, FR-FCFS, 64-entry queues (per-rank), address mapping: page interleaving	
Main Memory	DRAM	4GB, DDR3, 1333Mbps 2 Channels, 2 Ranks per channel
		DRAM Device: MT41J512M8
	PCM	32GB, 4 Channels, each with 8GB DIMM, 1 Rank per channel
		t_{SET}= 150ns, t_{RESET}= 100ns, t_{RCD}= 120ns

TABLE V. Workloads.

Workload	Applications
NAS	bt, cg, ft, ua, mg, sp
PARSEC	caneal, dedup, freqmine

We compare *Refree* with hybrid main memory systems that each employs a different DRAM refresh scheme. The implementations are all based on the configuration parameters depicted in TABLE IV. Also, it is assumed that in a DRAM memory rank, 28 rows are weak, 978 are mediocre and the remaining ~511K rows are strong.

1) *All-bank Auto Refresh (AR)*: In this scheme, the memory controller divides DRAM rows into 8K groups and refreshes each group within a refresh interval or t_{REFI}.

2) *Flexible AR (REFLEX)*: This scheme, which is based on the policy employed in [5], refreshes weak and strong rows once every 64ms and 256ms, respectively, through auto-refresh command. To the best of our knowledge, this recently proposed scheme has the lowest power consumption and execution time among the previously proposed refresh reducing schemes.

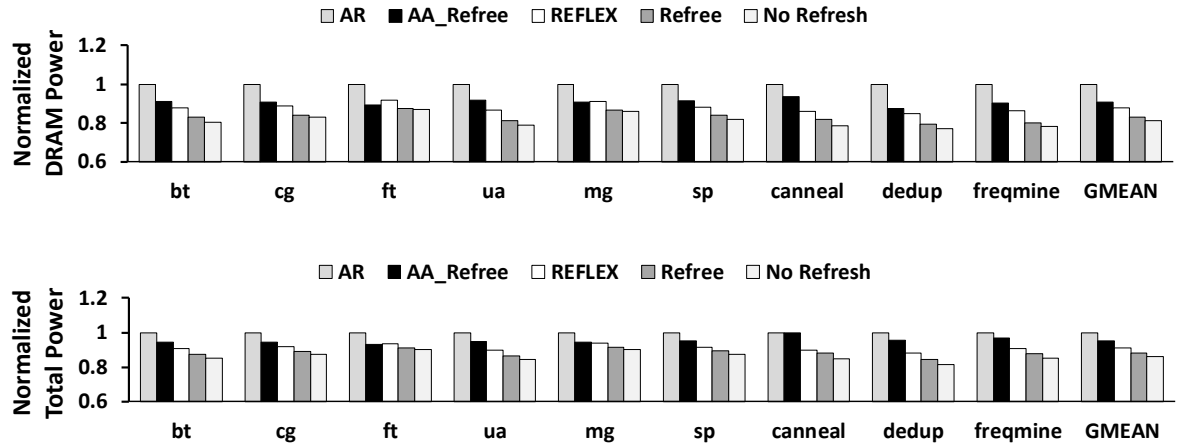


Figure 5. Normalized DRAM cache power consumption (top) and normalized total power consumption of the hybrid main memory system (bottom).

3) *Access-Aware Refree (AA_Refree)*: This is *Refree*; but considers all DRAM rows as weak. In other words, it assumes a 64ms retention time for all the DRAM rows.

4) *No Refresh*: This is the *ideal scheme* that assumes there is no need to refresh DRAM rows at all.

3.7.2 Power Evaluations

Figure 5 shows the power consumption of the 4GB DRAM cache (top) and the total power consumption of the hybrid main memory (bottom) for different refresh schemes. The results are normalized to baseline auto-refresh. *Refree* achieves up to 20.7% and 6.9% reduction in DRAM cache power consumption (16.9% and 5.6% on average), compared to AR and REFLEX, respectively. This reduction is mainly because of the following reasons: First, *Refree* is both access- and retention-aware, which decreases the number of candidate rows for either refresh or eviction/writeback. Second, to evict a non-accessed clean row from the DRAM cache, *Refree*

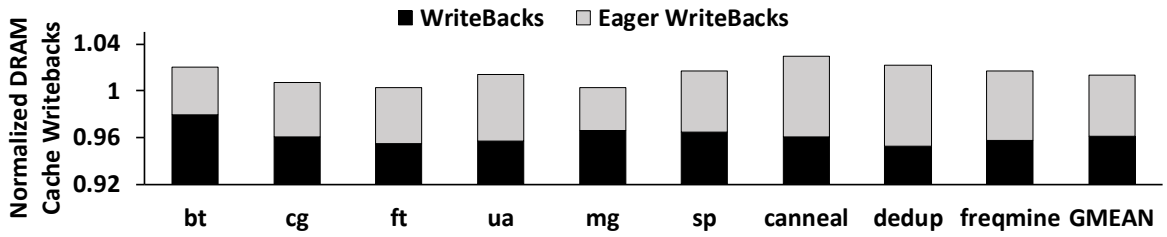


Figure 6. Normalized *Refree*'s total number of writebacks.

resets the valid status bit of the row to '0'. Thus, there will be no need to access the row itself in the DRAM. On the other hand, REFLEX or any other refresh scheme refreshes all the DRAM rows at some point any way. Third, the non-accessed rows that are unlikely to be reused in near future can be stored in PCM for a long time with no additional cost. However, to retain such rows in DRAM, they need to be periodically refreshed. *Refree* reduces the total power consumption of the hybrid main memory by 11.7% and 3.1% on average, compared to AR and REFLEX, respectively. Results also reveal that *Refree* reaches within 2.2% of total power consumption as compared to the ideal no-refresh scheme. Generally, the power consumption of *Refree* depends on its increase on the number of writebacks to the PCM device and DRAM cache miss rate.

Figure 6 shows the total number of DRAM cache entities written back to the PCM storage by *Refree* for the selected benchmarks. The results are normalized to baseline auto-refresh. The increase in the total number of writebacks is only 1.4% on average. The eager writebacks of *Refree* are 5.7% of the total number of writebacks on average. However, the number of normal writebacks under *Refree* is dropped compared with auto-refresh. This indicates that a large percentage of *Refree*'s eager writebacks actually overlaps with the writebacks that would naturally happen throughout the execution of the baseline. In other words, the data that *Refree* recognizes as

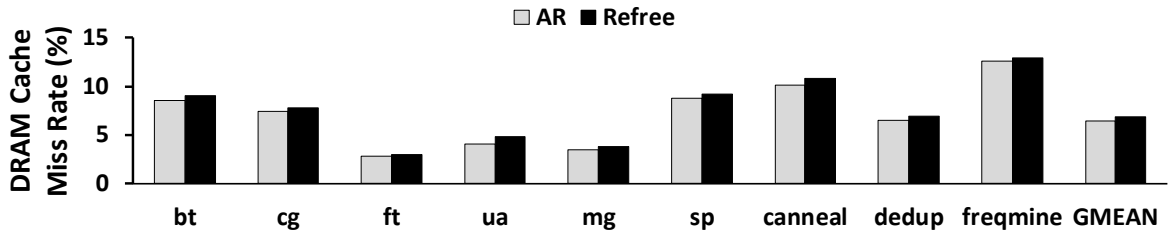


Figure 7. DRAM cache miss rate (%).

inactive and proactively writes back, will most probably be written back and replaced later, when the baseline scheme is running.

Figure 7 depicts the values of DRAM cache miss rate for baseline auto-refresh and *Refree*. Simulation results show that *Refree* only increases the average DRAM cache miss rate slightly from 6.4% by AR to 6.9% by *Refree*. Note that, all of the three schemes, AR, REFLEX and the ideal no-refresh, use the same DRAM cache management policy. Hence, the miss rate values and the number of writebacks are also the same in the three schemes. *Refree* consumes 7.4% less power than AA_Refree. This is because *Refree* is retention-aware. The reduction is directly related to having a longer monitoring interval for stronger rows and thereby a more accurate access-based row classification.

3.7.3 Performance Evaluations

Figure 8 compares the overall IPC values for different refresh schemes. The results are normalized to baseline auto-refresh.

Refree improves performance by up to 9.2% (4.2% on average), compared to AR. The reason for this improvement is: First, *Refree* eliminates refresh operations entirely and it does not increase the DRAM cache miss rate considerably. Second, the obstruction of regular requests to

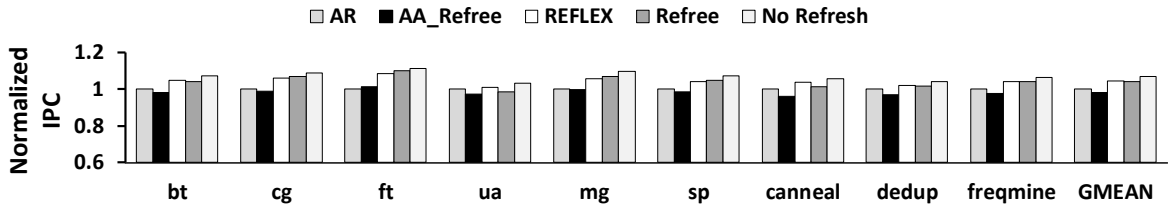


Figure 8. Normalized Overall IPC.

the PCM by *Refree*'s proactive writebacks is largely prevented by distributing writebacks over a large time epoch and the use of WBB. Third, *Refree* allows the memory controller to employ bank-level parallelism while row evaluation process is in progress. Fourth, requests to a DRAM bank can be served simultaneously with the eviction of non-active clean rows from it. Last, new data can replace a proactively evicted data without waiting for its eviction or writeback.

Compared to REFLEX, *Refree* has a negligible performance degradation, by 0.2% on average (up to 0.6%). The small increase in the DRAM cache miss rate is the main reason for such degradation. *Refree* also achieves, on average, 5.8% performance improvement compared to AA_Refree. The monitoring time interval considered in AA_Refree is two or four times shorter than most of the intervals of *Refree*. This causes some active rows to be evicted from the DRAM cache by mistakes, thereby increasing the DRAM cache miss rate and the number of writebacks to PCM.

3.7.4 Scalability

Figure 9 shows the hybrid main memory's total power consumption and the overall IPC values for different DRAM cache sizes, respectively. All the results are normalized to baseline auto refresh with 1GB DRAM cache.

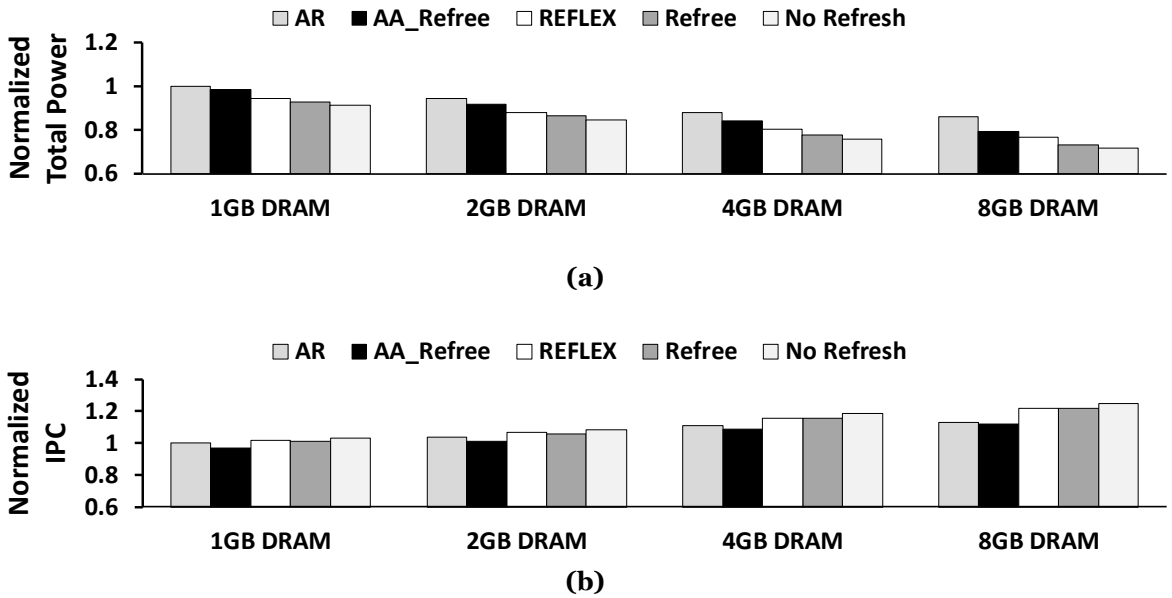


Figure 9. Scalability results: (a) normalized total power consumption of the hybrid main memory for different DRAM sizes; (b) normalized system IPC for different DRAM sizes.

Results show that *Refree* can achieve, on average, 7.3%, 8.6%, 11.7% and 15.0% reduction in the total power consumption of a hybrid main memory with a 1GB, 2GB, 4GB and 8GB DRAM cache, respectively, compared to baseline auto-refresh. In addition, our scheme improves performance of a hybrid main memory with a 1GB, 2GB, 4GB and 8GB DRAM cache, on average by, 0.9%, 1.9%, 4.2% and 7.4% respectively, compared to the baseline auto refresh.

Compared with REFLEX, *Refree* reduces the total power consumption of a hybrid main memory with a 1GB, 2GB, 4GB and 8GB DRAM cache by 1.6%, 2.0%, 3.1% and 4.3% on average, respectively. Also, while *Refree* degrades performance of a hybrid main memory with a 1GB, 2GB and 4GB DRAM cache by 0.8%, 0.7% and 0.2%, on average, it improves the performance of a hybrid main memory with an 8GB DRAM cache by 0.2% on average, compared to REFLEX. The *Refree* scheme is more effective for larger DRAM caches because: First, the cost of refresh

operations is higher in larger DRAMs. Second, when DRAM capacity and thus the number of requests served in DRAM increases, a larger portion of the hybrid main memory's energy and execution time will be reduced by improving the DRAM cache. Third, REFLEX always reduces a fixed number and up to a specific percentage of refresh operations while decisions made by *Refree* are access-based and dynamic.

3.8 Conclusion

In this chapter, we proposed a scheme called *Refree* to eliminate refresh operations of the DRAM cache in a hybrid DRAM/PCM main memory to improve system performance and energy efficiency. Our proposed scheme, *Refree*, takes all the refresh-reducing factors including rows' access pattern and retention time into consideration.

In general, a row that is accessed at least once within its retention time does not need to be refreshed. On the other hand, most of the rows in a DRAM device are strong and have very long retention times. Hence, a row that is not accessed within such long retention time can be recognized as not frequently accessed or dead and does not need to be refreshed and kept in the DRAM cache either. *Refree* then evicts an inactive row from the DRAM cache instead of refreshing it and writes it back to PCM if the row is dirty. The experimental results revealed that *Refree* can effectively reduce the memory power consumption with small performance impact. The effectiveness of *Refree* would further improve for future systems with larger DRAM sizes.

CHAPTER 4

AN ENERGY-EFFICIENT HYBRID DRAM/PCM MAIN MEMORY FOR MOBILE DEVICES

4.1 Introduction

Today, mobile platforms such as smartphones and tablets are the most commonly used computing devices in our daily lives. The capacity of main memory in mobile devices is limited due to scalability limit of DRAM (e.g., 1GB for iPhone 6). This can degrade user experience while multiple applications with large memory footprints are running on the device [34]. In fact, a recent study [76] has reported that in Android systems, more than 15% of applications must be relaunched when the available memory capacity is insufficient. To deal with this problem, hybrid main memory systems have been studied for mobile devices [7, 17, 34, 88]. In this study, we consider a hybrid DRAM/PCM main memory.

Mobile devices run on small batteries and need power management to extend battery life and ensure device usability [9]. Meanwhile, DRAM consumes large amounts of background energy even when the mobile device is idle, and its processor is powered off. The usage pattern of mobile devices is bursty; they are idle 90% to 95% of time and experience only sudden short bursts of activity between their long idle periods [13, 30]. The energy consumption of a DRAM-based main memory during idle periods contributes to 30% of the total system energy, including both active and idle periods [9]. To reduce the memory energy consumption during idle periods, DRAM

memory can be put into low power modes [13, 17], which only perform self-refresh operations to maintain data integrity.

In this study, we propose *NEMO* [65], a scheme that improves e**ne**rgy efficiency of **mo**bile devices with a hybrid DRAM/PCM main memory. The basic idea behind *NEMO* is to prevent unnecessary consumption of energy by the main memory without having a negative impact on performance. To do so, *NEMO* powers off as many power-hungry DRAM components as possible by dynamically managing data placement and movement between the two memory modules. More specifically, when the mobile device is in idle state, to minimize DRAM's power consumption, only a selective set of data that is critical to performance and fits in a single DRAM rank is kept in DRAM, while the rest of data is stored in PCM, which has near zero idle power. Moreover, to reduce memory power consumption during active states, *NEMO* predicts the number of DRAM ranks that need to be powered on based on user's behaviour in the past.

For each application, *NEMO* classifies its memory pages into two categories, "hot" and "cold", based on pages access frequency and recency in the past. It then places the hot pages, which are much more likely to be reused when user re-runs the application, in DRAM that has lower access latency compared to PCM. This actually prevents performance loss. On the other hand, keeping cold memory pages, which are less likely to be reused in future, in DRAM would cost large amounts of refresh energy especially during the mobile device's long idle periods. Hence, cold memory pages are stored into PCM, which has near-zero idle power. It should be noted that, page migrations between DRAM and PCM are performed during the idle periods and are off the critical path of memory accesses to minimize performance penalty.

Our experimental results indicate that *NEMO* can effectively improve energy efficiency and performance of a hybrid DRAM/PCM main memory used in a mobile device. For a mobile system with a hybrid main memory of 128MB DRAM cache and 1GB PCM running Moby benchmarks

[25], *NEMO* reduces the memory system power consumption by 10.2% and improves performance by 1.7%, on average, compared with simply putting DRAM into self-refresh mode during idle period.

4.2 Motivation

Reducing the energy consumption of mobile devices has become a major concern in recent decades. On such devices, the number of power-hungry hardware components is growing as a result of technological advancement. In the meantime, the battery size is limited; thereby supply cannot meet demand if energy dissipation is not managed wisely.

The usage pattern of portable devices, such as smartphones and tablets, is bursty with long idle periods. For such devices, the active periods account for only 5%-10% of time [13, 30], and are very short (a few minutes). Thus, the number of frequently accessed memory pages for the running applications can be small in each active period. Furthermore, the number of running applications and their memory intensity vary among active periods. In some active periods, the memory system is very busy serving requests from multiple memory-intensive applications, while in others, the memory system is not that overloaded. Meanwhile, during the long idle periods (90%-95% of time), a great portion of the total system energy is consumed by the main memory. Therefore, it can be concluded that: 1) in active mode, DRAM memory will consume a great deal of unnecessary background power if all of its components remain turned on all the time; and 2) in idle mode, although there are no accesses to the memory system, DRAM continues to consume background energy to retain data.

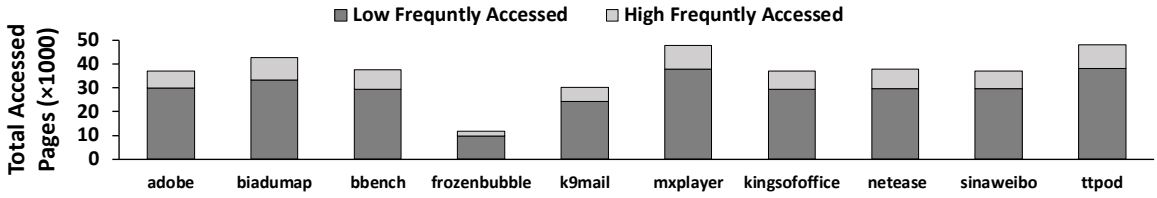


Figure 10. Total accessed memory pages.

To improve main memory efficiency, a prior study [17] has also benefited from a hybrid DRAM/PCM main memory system for mobile devices, in which the entire working set of the most frequently invoked applications is placed in DRAM and the rest of the applications' working set is stored in PCM. However, there are some major shortcomings with that approach. First, the working set of a highly invoked application is not entirely hot (i.e., highly accessed). Our results show that, on average, only 20.6% of all the accessed memory pages are highly accessed (i.e., accessed more than 2^8 times) for a selected set of mobile applications as depicted in Figure 10. The experimental parameters and methodology are explained in Section 4.10.1. Second, the highly accessed memory pages of the less frequently invoked applications are always accessed in PCM. This can significantly degrade system's performance and energy efficiency when those cold applications are running. Finally, when the number of highly invoked applications is large, fitting the entire working set of all those applications in DRAM becomes impossible; thereby the working set of some hot applications would be inevitably moved to PCM. This can also degrade system performance and increase power consumption.

In order to reduce DRAM background power during idle times, a recent study [13] has proposed *Morphable ECC* (*MECC*), which reduces refresh rate by using strong error correction codes. However, MECC only reduces the energy spent by refresh operations, which counts for less

than half of DRAM background power consumed during idle mode. In this study, our **main goal** is to reduce main memory's background power during both active and idle modes without impacting the performance. We consider a hybrid main memory system that includes a DRAM cache and a larger PCM for mobile devices. It should be noted that our scheme is also applicable for other types of NVMs. Our scheme reduces main memory's background power by dynamically predicting the number of DRAM ranks that need to be turned on during the active period and only powering on one DRAM rank for critical data during the idle period.

4.3 *NEMO* Overview

Our proposed scheme, *NEMO*, benefits from the large capacity provided by PCM to store the less frequently accessed (i.e., cold) memory pages of all applications during the long idle periods, while the most frequently accessed (i.e., hot) memory pages are stored in DRAM for better performance. In idle mode, the hottest pages of running applications are collected in a single DRAM rank, referred to as the "hot rank". The hot rank is then put into self-refresh state to maintain its data. The contents of the rest of DRAM ranks are stored into PCM and those ranks are put into deep power-down state, the lowest possible power state. In active mode, the number of DRAM ranks that need to be turned on in addition to the hot rank is predicted based on the applications' launching pattern and memory intensity in past. Figure 11 depicts the workflow of *NEMO*. The rank and page management process is illustrated in Figure 12 and is explained in detail next.

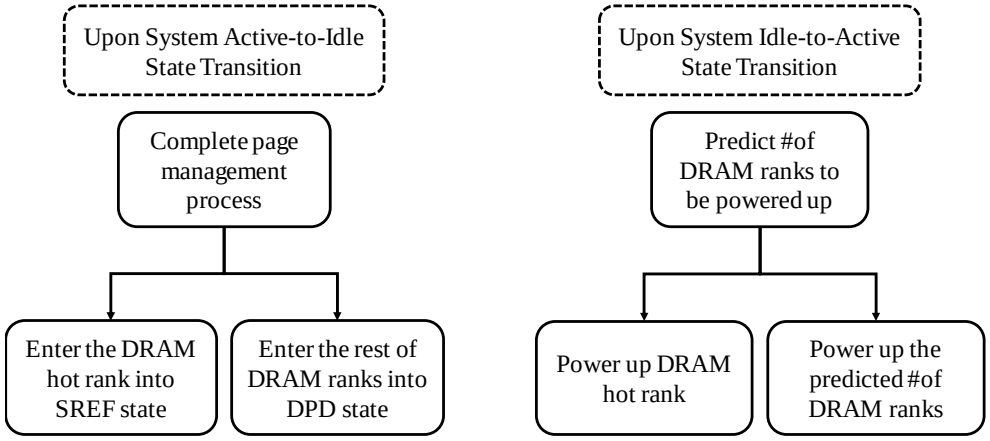


Figure 11. Workflow of *NEMO*.

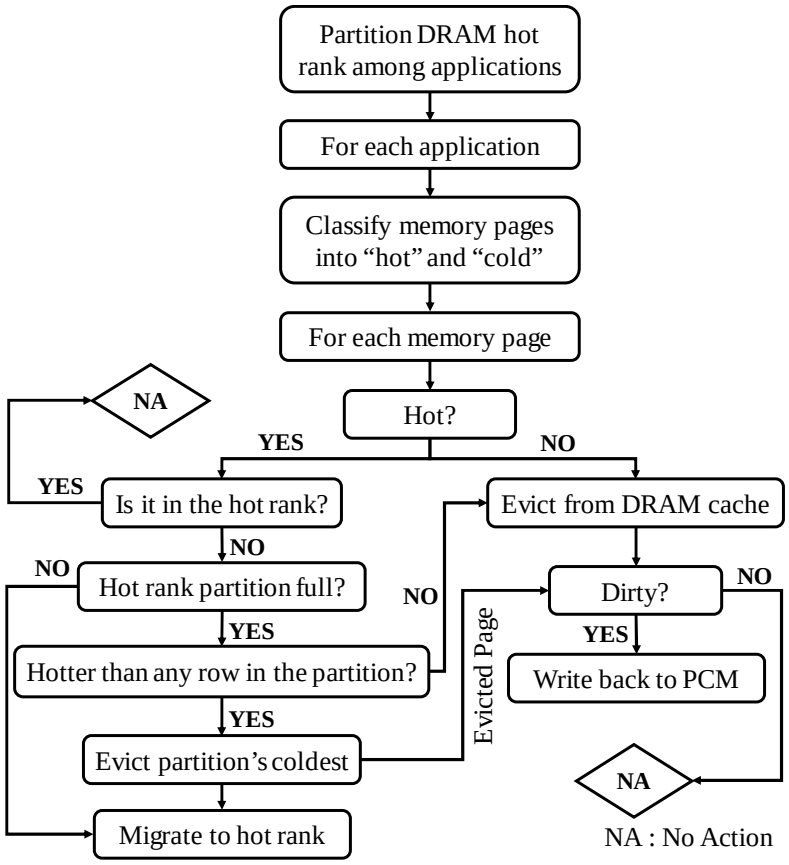


Figure 12. Page management process.

4.4 Memory Page Classification

NEMO classifies memory pages of an application into two categories, hot and cold. It then allocates a small portion of the DRAM cache to the applications' hottest pages so that they are always accessed in DRAM, which has lower access latency and dynamic energy compared to PCM. In general, this approach is beneficial because the hottest memory pages (of all the applications), which are more probable to be reused in future, are always kept in DRAM and cannot be evicted or replaced by the cold pages. On the other hand, during the long idle periods, cold memory pages are stored into PCM to prevent consuming energy for retaining nonvaluable data in the DRAM.

To classify an application's pages into two categories based on their hotness, *NEMO* adopts an algorithm similar to *Multi Queue* (*MQ*) proposed in a previous study [92]. The Multi Queue algorithm uses multiple LRU queues numbered from 0 to n-1 (i.e., $Q_0 \ldots Q_{n-1}$) plus a history buffer (i.e., Q_{out}) that keeps the access frequencies of the recently evicted blocks. When a cache block is accessed for the first time, its descriptor, which includes the block identifier and its frequency counter, is inserted into the head entry of Q_0. The frequency counter of the block is incremented on every cache hit to the block. Once the frequency counter of the block reaches 2^{i+1}, its descriptor will be promoted from Q_i to Q_{i+1}. Moreover, an expiration time is associated with each cache block. If the block stays in Q_i for a time period equal to its expiration time without any accesses, it will be demoted to Q_{i-1}. The frequency counter of a demoted block is halved by shifting right one bit. The block identifier and access frequency of a recently evicted block is kept in Q_{out} for some period of time; Q_{out} is a FIFO queue of limited size. It is worth mentioning that the updates to the MQ structure are performed by the memory controller and are off the critical path of memory accesses.

In this study, each DRAM cache block has the same size as a memory page. Similar to the previous studies [49, 87, 92], our scheme assumes n = 16 and that as few as eight queues (i.e., Q_8

to Q_{15}) are sufficient to separate hot pages from the others. *NEMO* considers an MQ structure for each application that has been executed so far. The MQ structure of an application is not updated when the application is not running, and it is cleared and updated again when the application is re-launched. To turn off as many power-hungry DRAM components as possible during the long idle periods (90-95% of time), *NEMO* stores the hottest pages of all the applications in the DRAM hot rank and migrates the dirty cold pages of all the applications to PCM, which has near-zero idle power. The hot rank is shared among all the applications, but not uniformly. The hot rank partitioning scheme is discussed next.

4.5 Hot Rank Partitioning

In a mobile device, applications are not used in the same manner. User may favour some applications over the others and use them more frequently. In other words, the likelihood of being invoked in an active period varies among the applications. At the same time, mobile applications differ in their working set sizes. Thus, when partitioning the hot rank, *NEMO* considers both the usage probability of applications and their memory capacity demands. On entering an idle mode, the DRAM hot rank is partitioned among the applications as following:

$$C_K = \frac{W_K \times CD_K}{\sum_{i=1}^{APPS} W_i \times CD_i} \times \frac{C_{DRAM\_RANK}}{C_{DRAM\_ROW}}$$

In this formula, $\lfloor C_K \rfloor$ is the number of rows (pages) allocated to application K in the hot rank; CD_K is the capacity demand of application K; W_k is the weight associated to application K based on its usage frequency and recency; C_{DRAM_RANK} and C_{DRAM_ROW} are DRAM rank size and row size, respectively; and *APPS* is the total number of applications invoked at least once so far. More specifically, the capacity demand or CD_K represents the amount of memory space that application

K requires to store its hottest pages. To quantify the capacity demand, the number of memory pages that belong to the eight highest-ranked LRU queues (i.e., Q_8 to Q_{15}) of the application is counted.

Furthermore, to compute applications' weights, we use a modified MQ structure, referred to as *AMQ*, with m LRU queues (i.e., q_0 to q_{m-1}). Once an application is used for the first time, its descriptor (i.e., identifier and frequency counter) is put at the head entry of q_0. The weights are positive integers between 1 to *APPS* and are assigned in descending order to the applications selected in order from q_{m-1} to q_0, and in each queue, from the most to the least recently used position. In other words, the higher the usage frequency and recency of an application in past, the larger its weight. The reason is that such application is also more probable to be reused in the near future. In this study, we assume $m = 8$ and an expiration time of two active periods, which means if an application is not used for two consecutive active periods, its descriptor will be demoted from q_i to q_{i-1}. On a side note, since we are using the floor value of C_K, at most *APPS* rows may not be allocated to any application at the end. In such case, we assign those rows to the application with the highest weight.

After determining the share of an application from the hot rank, *NEMO* migrates the hottest pages of the application (those that are not already in the hot rank) to its assigned portion. The migrations start from the LRU queues with the highest rank until the assigned portion becomes full. In other words, hot pages are selected in order from Q_{15} to Q_0, and in each queue, from the most to the least recently used position (i.e., tail to head) from the application's MQ structure. Then, the dirty rows in the rest of the DRAM ranks will be written back to PCM, which consumes a negligible amount of idle power. Note that all these migrations take place during the idle period and thereby do not incur any performance penalty. The hot rank is then put into SR (Self-Refresh)

state to maintain its data till the following active period. The rest of the DRAM ranks are put into DPD (Deep Power Down) state, which consumes close to zero background energy.

4.6 DRAM Cache Management

In this work, a DRAM/PCM hybrid main memory is considered for mobile devices. The application's working set initially resides in PCM, and DRAM is employed as a hardware-managed cache for the PCM. Generally, data can be stored in DRAM caches at different granularities. The size of a DRAM cache block could be as small as a processor cache block or as large as a memory page. For our proposed scheme, *NEMO*, using a page-based DRAM cache structure that stores all the columns of a PCM row in a single DRAM row is more beneficial for two reasons: First, row buffer locality is expected to be high for mobile applications. In fact, our experimental results reveal that for the selected set of mobile workloads, the row buffer hit rate ranges from 81.4% for *FrozzenBubble* to 89.9% for *TTPod* when the workloads run individually, and row interleaving address mapping is used. Second, in this way, when a row needs to be written back, all the write requests sent to the PCM device will hit in the row buffer. This minimizes the energy and latency overheads of the writeback operations. On the other hand, mobile devices are small, thereby it is very important to keep the area overheads at minimum. Thus, in this study, we use the page-based *tagless DRAM cache* structure proposed in [44]. The latency penalty of the tagless cache is zero in case of a DRAM cache hit (for cachable pages); and is a cache fill plus the GIPT (global inverted page table) update latency in case of a DRAM cache miss.

The cache replacement policy is modified in our work; *NEMO* chooses the eviction victims from the coldest memory pages of the applications, starting from the coldest application. In other words, it selects victims from the lowest-ranked, non-empty LRU queue of the applications MQ structures, while the applications are sorted in ascending order of their weights. It is worth

mentioning that this approach is different from the original MQ replacement policy, which uses only one set of LRU queues for the cache. In fact, simply employing the non-modified MQ regardless of the applications usage pattern is not helpful for mobile devices. This is because the cold memory pages of an application that is running in the current active period could evict the hot memory pages of the applications executed in the previous active periods (because those pages are demoted to the lower-ranked queues) even though those applications might be reused in the near future again.

4.7 Active Mode Management

Memory capacity demand is not the same among the active periods. For an active period, the required DRAM cache space could be smaller than the actual DRAM size (e.g., when only one application with low memory intensity is running). Therefore, by predicting the number of the DRAM ranks that need to be turned on (in addition to the hot rank), *NEMO* also minimizes DRAM background power during the active periods. The reason that we choose prediction over adjustment is that the active periods are relatively short and busy; thus, gradually powering up the DRAM ranks when the mobile device is in the active state can be detrimental to system performance. The prediction hides the powering up penalty by turning on the DRAM ranks simultaneously with the mobile device idle-to-active state transition. Moreover, to prevent the misprediction penalty during the initial active periods where there is not enough knowledge about the system's past, *NEMO* turns on all the DRAM ranks for a specific number of active periods (two in this study) at the beginning.

In general, to predict the number of DRAM ranks to be powered up, *NEMO* predicts how many applications and which ones are likely to be invoked in the next active period. Then, the number of DRAM ranks required to store the predicted applications' data is calculated. Specifically, the

prediction model works in three steps as following. First, the number of applications that are likely to be invoked in the next active period is predicted. To do so, a small "prediction buffer" is used, which is indexed from 1 to p. The i'th element of the prediction buffer is incremented whenever the number of running applications in an active period is i. In other words, the i'th element of the prediction buffer returns the number of active periods in past during which i applications were running (p'th element for $i \geq p$). The index of the maximum value of all the elements stored in the prediction buffer is used as the predicted number of applications for the next active period. In case of having multiple indices with the same maximum value, the largest index is used as the predicted number. Second, after predicting the number of applications, the applications themselves are chosen from the AMQ structure in descending order of their weights. The number of memory pages accessed by the predicted applications (referred to as *NoP*) is then counted as the number of memory pages that are not migrated to the hot rank and are in Q_0 to Q_{15} of the applications MQ structures. Finally, the minimum number of DRAM ranks required to accommodate those pages is determined as $(\frac{\text{NoP}}{\text{C}_{\text{DRAM\_Rank}}})$.

4.8 Page Migrations

There are two types of page migrations performed by *NEMO* at the beginning of an idle period: 1) migrating hot memory pages to the DRAM hot rank; 2) writing back the dirty cold pages (i.e., the pages that are not migrated to the hot rank) to PCM.

To minimize the runtime overhead of page migrations, *NEMO* takes advantage of bank-level parallelism in two ways: First, consecutive row migrations are sent to different banks of the hot rank in a round robin way (i.e., the first row to the first bank, the second row to the second bank and so on). To indicate whether a memory page is in the hot rank or not, we add one additional

bit, referred to as *HR* (Hot Row) to cTLB. After migrating a memory page to the hot rank, we update the page's cache address and set its *HR* bit to one in cTLB. Second, writebacks to PCM and migrations to the hot rank are not initiated from the same bank in a DRAM rank. Moreover, in order to reduce the number of DRAM cache writebacks during the idle periods, *NEMO* eagerly writes back the dirty cold pages of the applications (sorted in ascending order of their weights) to PCM whenever the bus is idle and clears the pages' dirty bits. The eager writebacks start from the lowest-ranked, non-empty LRU queue, and in each queue, from the least recently used position. It should be noted that page migrations to PCM and to the hot rank can also be performed in parallel by equipping each DRAM rank with one extra row-buffer similar to [49]. However, we do not use this method because it requires some fundamental changes to the structure of the DRAM ranks.

4.9 Storage Overhead

The *NEMO* design includes three components: MQ module, AMQ module, and the prediction buffer, which are integrated into the memory controller. Song et al. [76] have reported that based on their collected usage logs from different users, a user had used on average 52 applications over a period of two weeks but only 10 applications among them had been heavily used. Hence, in this study we assume that the MQ module can keep the MQ structures of up to sixteen applications and if the total number of invoked applications exceeds sixteen, the new application MQ structure replaces the coldest application's MQ structure starting from its lowest ranked LRU queues.

1) MQ module: The size of each block descriptor in our design is 117 bits, among which 16 bits are used for the corresponding cache address, 32 bits for the block's last access time, 4 bits for the queue number in MQ, 15 bits for the frequency counter, 4 bits for the application number, and 46 bits for the pointers to other descriptors. Upon eviction of a block, the physical page number

(PPN) of the block is recovered from GIPT, then the evicted block is added to the history buffer of MQ. The history buffer Q_{out} is a small buffer with 30K entries with 61 bits each (42 bits for PPN, 15 bits for the frequency counter and 4 bits for the application number). For a 128MB DRAM cache, the total storage overhead of the MQ module is 1.13MB. The MQ structures are stored in the DRAM hot rank. However, to avoid performance degradation, similar to [49], we add a small on-chip entry cache (16KB with 1K entries) to the memory controller for storing the most recently used MQ structures. To find a DRAM cache block's MQ entry, the memory controller uses hashing with the corresponding cache address. Misses in the entry cache generate requests to DRAM.

2) AMQ module: Assuming 8 LRU queues, the space overhead of AMQ is 100 bits per application descriptor: 10 bits for the application number (for all the applications), 3 bits for the queue number in AMQ, 32 bits for the last active period in which the application is run, 7 bits for the frequency counter, and 48 bits for the pointers to other descriptors. Hence, the total storage overhead of AMQ is 12.5KB.

3) Prediction buffer: Assuming 30 bits per entry and 10 entries, the size of the prediction buffer is only 300B. The AMQ and the prediction buffer are stored in the DRAM hot rank. In our evaluations, all the overheads incurred by these new components are considered. Overall, for a hybrid main memory with 128MB DRAM plus 1GB PCM, the area overhead of *NEMO* is less than 0.9% of the DRAM cache space, which is negligible.

4.10 Results

4.10.1 Methodology

Because our experiments need to study the behaviour of mobile applications for long running period, we use a combination of execution- and trace-driven simulations in our study. First, we

TABLE VI. System configuration.

Processor	ARM, quad-core, 2.0 GHz, out-of-order	
L1 Cache	32 KB, 4-way, LRU, 64B cache line, write-back, write allocate	
L2 Cache	512 KB, 16-way, LRU, 64B cache line, write-back, write allocate	
Memory Controller	Open page, FR-FCFS, 64-entry queues (per-rank), address mapping: page interleaving	
Main Memory	**DRAM**	128MB, LPDDR, 200MHz bus speed 1 channel, 4 ranks/channel, 4 banks/rank, 4k rows/bank, 32 columns/row
		DRAM Device: MT46H8M16LF
	PCM	1GB, 1 channel, 1 rank/channel, 8 banks/rank, 64k rows/bank, 32 columns/row
		t_{SET}= 150ns, t_{RESET}= 100ns, t_{RCD}= 120ns

use cycle-accurate simulators to collect main memory access traces (reads and writes with timestamps) from running benchmark workloads. Then, we create the mixes of traces and replay the mixes on our detailed memory system simulator. To collect memory traces, we model a quad-core processor on a scalable application-level architectural performance simulator based on Gem5 simulator [19] integrated with NVMAIN [62], a cycle accurate main memory simulator designed to simulate emerging non-volatile memories at the architectural level. The system configuration of our experiments is shown in TABLE VI. The DRAM configurations (i.e., timing and current parameters) are obtained from [53]. The PCM configurations are generated by NVSIM [16] and CACTI [8]. Note that the cell parameters used in NVSIM are based on the projections by [12]. For the main memory, we pick a small size to match the footprint of the workloads. The benchmark workloads used in this study are chosen from the Moby benchmark suite [25] as depicted in TABLE VII. We run each workload for the number of instructions given in the table. The selected workloads are reprehensive of a wide range of applications that are commonly used in mobile devices. In current and future mobile systems, various combinations of these workloads are likely to be executed by the user in an active period. Therefore, in this work, to mimic user's behaviour,

TABLE VII. Workloads [25].

Benchmark	IC*	Description	Input
BBench	2.48	Web Browser	Web pages
K9Mail	1.18	Email Client	Buffered emails
SinaWeibo	2.23	Social Network	Buffered texts
NeteaseNews	2.65	News Reader	Buffered news
KingSoftOffice	2.24	Document Editor	A doc file
AdobeReader	2.09	Document Editor	A PDF file
BaiduMap	3.53	Map Client	Buffered maps
MXPlayer	3.84	Video Player	A video file
TTPod	3.87	Audio Player	A music file
FrozzenBubble	0.28	Puzzle Game	Null

***IC: Instruction Count (Billions)**

we create our mix traces in three steps. In the first step, we select a set of workloads (i.e., five benchmarks) that are likely to be run together to create a mix trace. In the second step, a random subset of the selected workloads is combined to model an active period. The active periods are obtained from multi-programming executions. For each mix trace, a predefined usage frequency is assigned to each workload that determines how often the workload must be repeated. Though only a fixed set of workloads is used in a mix trace, the number and types of the workloads running in its active periods are not the same. Finally, the memory traces of six active periods that all use the same set of workloads are concatenated to create a mix trace. Overall, the system is assumed to be idle for 90% of time. Since the memory traces of the active periods are large, we consider only six active periods for each mix trace. TABLE VIII summarizes the mix traces.

We compare *NEMO* with hybrid main memories that each uses a different power management technique in DRAM during the idle periods. Schemes that only use a DRAM-based memory need to put all the DRAM ranks into self-refresh state during the idle periods, which consumes additional power. Also, a hybrid main memory provides significant performance and energy

TABLE VIII. Summary of Mix traces.

MIX 1	Workloads	K9Mail	SinaWeibo	BBench	NeteaseNews	KingSoftOffice
	Usage Frequencies	3 *(actives: 1,4,6)*	3 *(actives: 3,5,6)*	2 *(actives: 2,5)*	2 *(actives: 3,6)*	1 *(active: 4)*
MIX 2	Workloads	MXPlayer	K9Mail	BaiduMap	AdobeReader	FrozzenBubble
	Usage Frequencies	4 *(actives: 1,4,5,6)*	3 *(actives: 1,3,6)*	2 *(actives: 1,6)*	1 *(active: 2)*	1 *(active: 5)*
MIX 3	Workloads	BBench	TTPod	BaiduMap	NeteaseNews	FrozzenBubble
	Usage Frequencies	5 *(actives: 1,2,3,4,6)*	2 *(active: 1,5)*	1 *(active: 2)*	1 *(active: 2)*	1 *(active: 3)*
MIX 4	Workloads	MXPlayer	AdobeReader	K9Mail	SinaWeibo	TTPod
	Usage Frequencies	3 *(actives: 1,3,6)*	2 *(actives: 2,5)*	2 *(actives: 2,5)*	2 *(actives: 1,4)*	1 *(active:2)*
MIX 5	Workloads	NeteaseNews	KingSoftOffice	AdobeReader	BaiduMap	MXPlayer
	Usage Frequencies	2 *(active: 1,6)*	1 *(active: 2)*	1 *(active: 3)*	1 *(active: 4)*	1 *(active: 5)*
MIX 6	Workloads	BBench	SinaWeibo	KingSoftOffice	BaiduMap	TTPod
	Usage Frequencies	3 *(actives: 1,4,5)*	3 *(actives: 1,2,5)*	3 *(actives: 1,2,6)*	3 *(actives: 2,3,4)*	3 *(actives: 3,4,6)*

benefits compared to a conventional DRAM-based memory for mobile devices [17]. Hence, we only compare *NEMO* with other hybrid main memories.

1) *Self-Refreshing Idles (SRI)*: this scheme puts all the DRAM ranks into SR state when the mobile device is idle and is used as our baseline.

2) *NEMO Idle Optimized (NEMOI)*: this scheme works similar to *NEMO* during idle periods but does not perform *NEMO*'s active period optimizations.

3) *Ideal Idles (IDI)*: this is a non-practical scheme that assumes there is no need to perform refresh operations to maintain data in the DRAM cache during the idle periods.

4) *Ideal Actives (IDA)*: this is also a non-practical scheme that works similar to *NEMO* during the idle periods but for each active period (including the initial two active periods) turns on the minimum number of DRAM ranks needed to hold the entire working set of the workloads within that active period. In other words, IDA knows the exact number of DRAM ranks in advance. It should be noted that, the working set size of all the applications within a mix trace is larger than the DRAM cache size. However, the working set size of the currently running applications within an active period can be smaller than the DRAM cache size.

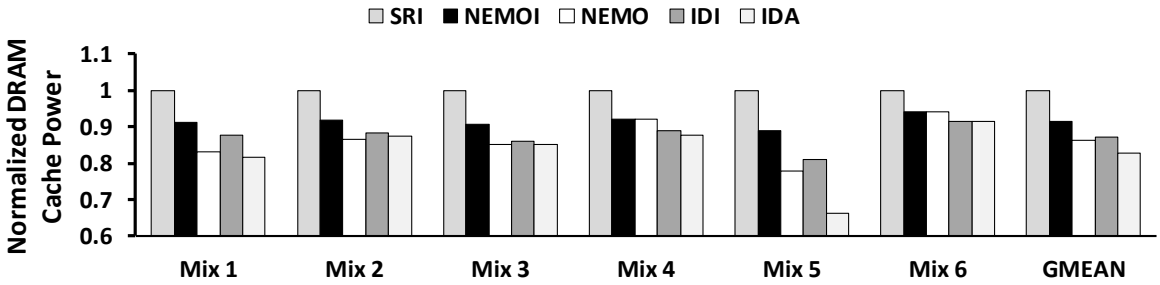


Figure 13. Normalized DRAM cache power consumption.

We use the same DRAM cache structure (fully-associative tagless cache) for all the schemes but the cache replacement policy for the SRI and IDI schemes is LRU. Moreover, for all the schemes, whenever there is no request to the memory system during the active periods, the DRAM ranks are put into active power down state, which has a very short resynchronization time, to reduce the active power consumption.

4.10.2 Power Evaluations

Figure 13 compares the power consumption of the DRAM cache for the different schemes explained before. The results are all normalized to baseline SRI. *NEMO* achieves up to 22.2% (13.8% on average) reduction in DRAM cache power consumption compared to SRI. This reduction is the result of idle and active mode optimizations performed by *NEMO*. Hence, we have removed the active periods optimizations in NEMOI to highlight the effect of each optimization separately. The NEMOI scheme reduces DRAM cache power by up to 11.3% (8.6% on average) compared to SRI. The maximum power that can be saved during the idle periods is up to 19.0% (12.8% on average), as shown with IDI scheme. The extra power consumed by NEMOI compared to IDI is mainly because of the hot rank migrations and the power that the hot rank consumes in

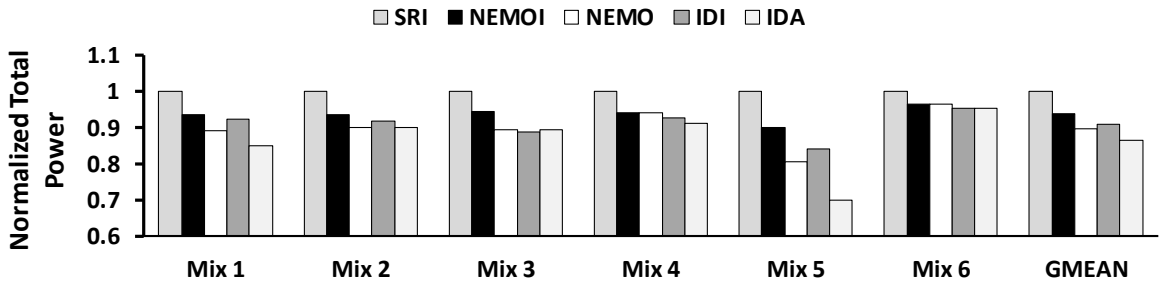


Figure 14. Normalized total power consumption of the hybrid main memory system.

self-refresh state during the idle periods. The power that the prediction logic consumes has also been considered in our evaluations. *NEMO* further reduces DRAM cache power by 5.7% on average compared to NEMOI. More specifically, compared to NEMOI, the power reduction by *NEMO* is 12.3% for Mix 5, the lightest Mix; 0.0% for Mix 6, the heaviest Mix; and up to 9.2% (5.3% on average) for the rest of the Mixes, the common cases. These reductions are actually the direct results of *NEMO*'s active period optimizations. The benefit of the active period optimizations is largely due to eliminating all the auto-refresh operations in the powered down DRAM ranks. Note that, *NEMO* turns on all the DRAM ranks for the first two active periods for training purpose. For most workloads, *NEMO*'s power consumption is close to that of IDA, which always turns on the necessary number of DRAM ranks even during the initial two active periods (*NEMO*'s warm up period). The only exception is for Mix 5, which is the lightest mix. *NEMO* turns on more ranks than needed, which consumes some extra active power. On average, *NEMO* consumes 3.9% more DRAM power than IDA, which indicates that *NEMO* predicts the usage of DRAM ranks during the active mode accurately.

Figure 14 shows the total power consumption of the hybrid main memory. The results are normalized to baseline SRI. The results show that *NEMO* reduces the total power consumption of

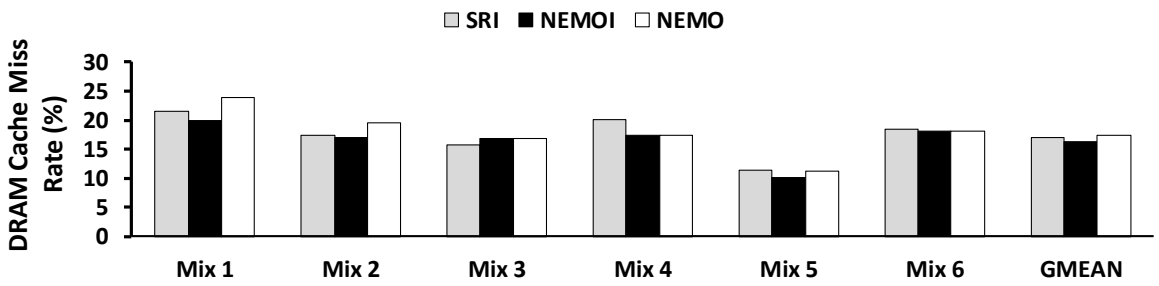


Figure 15. DRAM cache miss rate (%).

the main memory by up to 19.4% (10.2% on average) compared to SRI (ranging from 19.4% for Mix 5 to 3.5% for Mix 6). Moreover, *NEMO* consumes 4.2% less power than NEMOI. Generally, the power consumption of the hybrid main memory depends on the impact of *NEMO* on the DRAM cache miss-rate and the total number of writebacks.

Figure 15 depicts the values of DRAM cache miss rate (in all the six active periods) for the evaluated schemes. Since the miss rate values for the IDI scheme is similar to SRI (they both use the same cache management policy), we only include SRI in the figure. Similarly, since the miss rate values for the IDA scheme is similar to NEMOI, we only include NEMOI in the figure. The results show that *NEMO* increases the average DRAM cache miss rate slightly from 17.1% by SRI to 17.4% by *NEMO*. However, the average DRAM cache miss rate is decreased from 17.1% by SRI to 16.2% by NEMOI. The reason for this reduction is: 1) in most cases, a large number of NEMOI (or *NEMO*)'s evictions are inevitable for providing the cache space required for new applications data; 2) during the idle periods, NEMOI (or *NEMO*) only evicts colder pages that are less likely to be re-accessed in the future; 3) during the active periods, NEMOI (or *NEMO*) chooses its eviction victims more wisely. In other words, the eviction victims of SRI are more likely to be accessed again at some point later. The reason for the increase in the average DRAM cache miss rate by

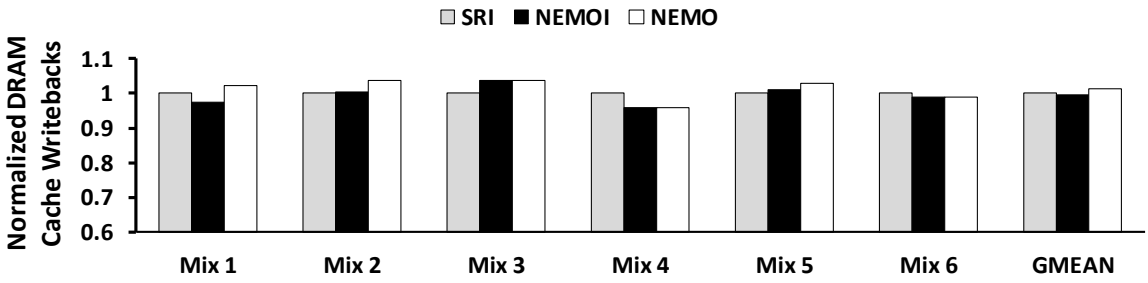


Figure 16. Normalized total number of writebacks.

NEMO compared to NEMOI is the active periods where the predicted number of DRAM ranks by *NEMO* is smaller than the actual number of ranks needed. It should be noted that even in those cases, *NEMO*'s DRAM cache replacement policy, which evicts only the dead pages, prevents any significant overhead. Moreover, our evaluation results reveal that the accuracy of the proposed predictor is 86.1% on average. This means that in most of the active periods of a Mix, the predicted number of ranks work similar to the situation when all the DRAM ranks (or in other words, the necessary number of ranks or more) are powered up.

Figure 16 shows the total number of DRAM cache rows written back to PCM by *NEMO*. The results are normalized to baseline SRI. Compared to SRI, the number of writebacks is increased by 1.2% on average by *NEMO*; but is decreased by 0.4% on average by NEMOI. The increase in *NEMO*'s writebacks compared to NEMOI (1.6% on average) is the price we pay for powering up fewer ranks. For Mix 5, the number of rows written back by NEMOI is more than those by SRI. The reason is: in this case, some of the evictions/writebacks performed by NEMOI (*NEMO*) during the idle periods are not necessary (to empty the required cache space) during the regular executions of *SRI*. However, since most of the applications are run only once (or twice, in two far separated active periods) in this mix, it does not have much negative impact on the miss rate. It

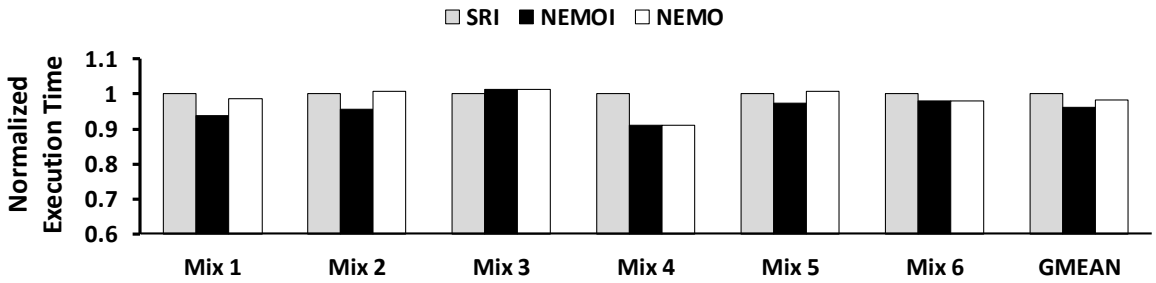


Figure 17. Normalized execution time.

should be noted that the increase in the number of writebacks is very small; thereby *NEMO* does not harm PCM lifetime.

4.10.3 Performance Evaluations

Figure 17 shows the performance of the evaluated schemes for the Mix workloads. The results are normalized to baseline SRI. Since for IDI, only the power state of DRAM in idle periods (not the performance) is different from SRI, we only include SRI in the figure. Though the IDA scheme powers up fewer DRAM ranks than NEMOI (i.e., the necessary number of ranks only) during some of the active periods, it does not affect the DRAM cache miss rate or number of writebacks. Therefore, the performance values of IDA are the same as those of NEMOI.

The results show that *NEMO* almost has no negative performance impact and even may improve performance. Compared with SRI, its performance is improved by 1.7% on average (up to 8.9%). The reasons for this improvement are: First, the number of writebacks performed during the active periods is always decreased for all the Mix workloads, although the total number of writebacks by *NEMO* is increased for some of the Mixes. Figure 18 shows the percentages of reduction in the number of active periods' writebacks by *NEMO* compared to SRI. The decreased

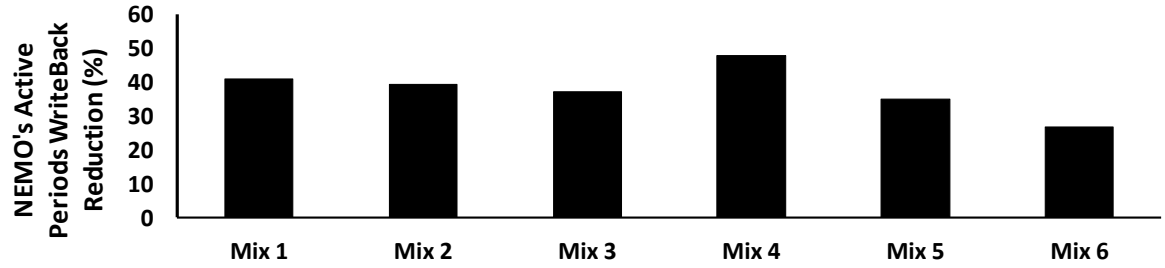


Figure 18. Percentage of reduction in the number of active periods' writebacks by *NEMO* compared to Baseline.

number of writebacks alleviates the bus utilization during the active periods, which is very helpful especially when the bus is mostly busy with responding to the requests generated by DRAM cache misses. Second, as we discussed before, *NEMO* increases the DRAM cache miss rate only very slightly. The execution time of *NEMO* is 2.3% longer than NEMOI due to misprediction of the number of necessary DRAM ranks. It should be noted that all the migrations performed by *NEMO* during the idle periods are off the critical path of memory accesses and do not incur any performance penalty.

4.10.4 Impact of Design Parameters

The hybrid main memory's total power consumption and the total execution time of *NEMO* with different number of DRAM hot ranks are shown in Figure 19. The results are normalized to *NEMO* with one hot rank. We have included only the average values in the figure.

Compared to *NEMO* with one hot rank, *NEMO* with zero hot ranks reduces main memory's power by only 1.6% on average, while it incurs a performance loss of 3.7% on average. This is because the evictions/writebacks of the hottest pages are unnecessary and increase the DRAM cache miss rate. Moreover, compared to *NEMO* with one hot rank, *NEMO* with two hot ranks

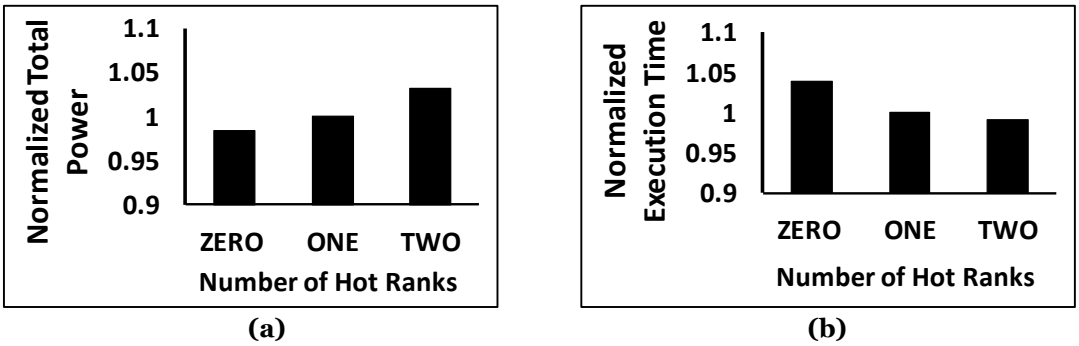


Figure 19. *NEMO*'s (a) normalized total power consumption; (b) normalized execution time; for different number of DRAM hot ranks.

improves performance by 0.8% but increases the total main memory's power consumption by 3.1% on average. This means that the pages stored in the second hot rank are not as critical to system's performance as those pages stored in the first hot rank. In other words, keeping a second DRAM hot rank powered up increases the power consumption with a small performance benefit. Overall, based on the results, *NEMO* with one DRAM hot rank provides the best balance between the main memory's power consumption and performance. That is why our scheme keeps only a single hot rank.

Figure 20 shows the hybrid main memory's total power consumption for different DRAM cache sizes. The results are normalized to baseline SRI with 64MB DRAM cache. The results show that NEMO can achieve, on average, 6.6%, 10.2%, and 18.1% reduction in the total power consumption for a hybrid main memory with a 64MB, 128MB, and 256MB DRAM cache, respectively, compared to baseline SRI. Using a larger DRAM cache for NEMO has several benefits. First, when the DRAM is larger, the DRAM hot rank is also larger and can keep more hot pages. Second, the obtainable power saving by the idle period optimizations becomes larger, because the larger the DRAM, the higher the self-refresh operations energy. Third, the active

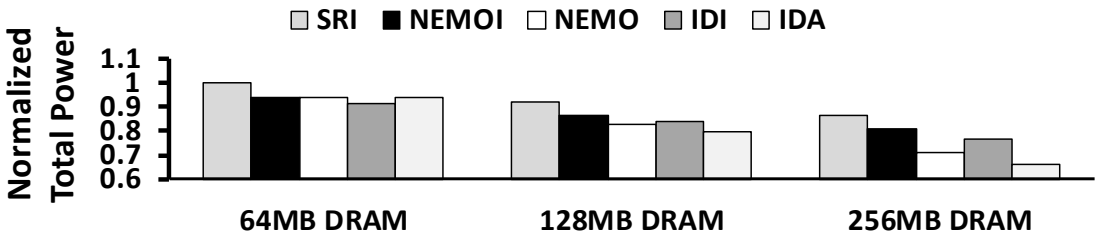


Figure 20. Normalized total power consumption of the hybrid main memory for different DRAM sizes.

period optimizations can also be more effective because more data can be placed in each DRAM rank and there will be more opportunities to power down some ranks. Besides, the background power that can be saved during the active periods is also larger for larger DRAM caches. Forth, when DRAM capacity and thus the number of requests served in DRAM increases, a larger portion of the hybrid main memory's energy will be reduced by improving the DRAM cache.

4.11 Conclusions

In this chapter, we proposed a novel scheme called *NEMO* to minimize the background energy of hybrid main memories used in mobile devices. To do so, *NEMO* takes advantage of the unique usage pattern of mobile devices, which are idle most of the times.

During the long idle periods, *NEMO* evicts the nonvaluable memory pages (those that are less likely to be reused in future) from the DRAM cache and collects the remaining hot memory pages in a single DRAM rank, called the hot rank. It then powers off all the DRAM ranks except for the hot rank. In addition, to minimize the background power during the active periods, it predicts the number of DRAM ranks that needs to be powered up in addition to the hot rank based on the

applications' launching pattern in the past. The experimental results revealed that *NEMO* could effectively reduce the memory power consumption without negative performance impact.

CHAPTER 5

A WRITEBACK-AWARE LLC MANAGEMENT SCHEME FOR PCM-BASED MAIN MEMORY SYSTEMS

5.1 Introduction

Due to its scalability limits, DRAM can no longer satisfy the memory capacity demands of the modern-day applications. Hence, PCM is gaining interest as DRAM replacement for building the future main memories [4, 65, 66, 70, 72]. However, PCM suffers from some major shortcomings including long write latency, high write energy consumption, and limited write endurance, which are all related to write operations. To deal with the overheads of write operations in PCM, there are two common types of solutions. First category is the optimizations on the PCM architecture to minimize the impact of writes. For example, modifying the request scheduling policy in the PCM main memory to alleviate the performance overheads of writes on reads [2, 68, 82, 91]; or modifying the PCM main memory architecture to reduce or balance the write loads on PCM cells and enhance their lifetime [86]. Second category is reducing the total number of writes sent to the PCM main memory by modifying the Last Level Cache (LLC)'s management policies [80]. This work falls into the latter category.

We propose *WALL* [63, 64], a novel dynamic **w**riteback-**a**ware **LL**C management scheme to improve performance, energy efficiency, and lifetime of a PCM-based main memory system by reducing the number of writebacks from LLC to PCM. In general, *WALL* consists of a writeback-aware set balancing mechanism and a writeback-aware replacement policy. Writebacks of the last

level cache are not uniformly distributed among its sets; some sets have far more writebacks than others, while some sets rarely see writebacks (please see Section 5.2 for detailed results). The proposed set balancing mechanism reduces the number of writebacks by employing the underutilized sets with infrequent writebacks as auxiliary storage units (inside LLC) for the evicted dirty lines from sets with many writebacks. Moreover, the proposed writeback-aware replacement policy tries to keep the dirty blocks that are frequently accessed after eviction in LLC. To do so, it allows the dirty eviction victims (i.e., dirty LRU block) to stay in the cache and be re-accessed; if the block becomes LRU block again without being accessed, it will be evicted from LLC then.

To implement the set balancing mechanism of *WALL*, we first propose a simple partner assignment strategy [63]. The simple partner assignment strategy classifies sets into three categories: 1) “writer”; sets with frequent writebacks; 2) “non-writer”; sets with infrequent writebacks; and 3) “neutral”; sets that are neither writer nor non-writer. Each writer set is partnered with a non-writer set until no non-writer sets are left un-partnered. Although this partner assignment strategy has simple implementation and works effectively for many workloads, it also has two limitations. First, it cannot always balance the number of writer and non-writer sets; thereby some writer or non-writer sets may remain without partners. Second, it does not guarantee the inclusion of writer sets with the largest number of writebacks or non-writer sets with the smallest number of writebacks in the partner assignment process. We further propose three novel partner assignment strategies called *contraction*, *expansion* and *ConExp* to alleviate the limitations of the simple partner assignment strategy [64]. Specifically, when the number of writer and non-writer sets are imbalanced, the expansion strategy includes some of the neutral sets, which are the most eligible to be considered writer or non-writer, in the partner assignment process. On the other hand, the contraction strategy addresses the second limitation

of the simple partner assignment strategy by first assigning partners to writer sets with the largest number of writebacks and non-writer sets with the smallest number of writebacks. The *ConExp* strategy is a combination of the contraction and expansion strategies to deal with both limitations of the simple partner assignment strategy.

We evaluate our proposed schemes by running SPEC CPU2006, NAS and PARSEC workloads on GEM5 [19] integrated with modified NVMAIN [62], which simulates the PCM-based main memory system. The experimental results indicate that our schemes can reduce the total number of LLC writebacks by 30.9%, on average, compared to a baseline scheme, which uses the LRU replacement policy. As a result, for a system with eight cores and a 4GB PCM main memory, it can enhance PCM lifetime by 1.29×, on average, and reduce the memory energy consumption by 23.1%, on average.

5.2 Motivation

In this section, we explain the motivation of our work by investigating LLC sets' writeback behaviour. To this end, we have run three workloads, selected from different benchmark suites (*sp* from NAS [60], *gcc* from SPEC CPU2006, and *streamcluster* from PARSEC [6]) on our simulated system (please see Section 5.9.1 for more details).

Figure 21 shows the cumulative distribution of writebacks over LLC sets. The results reveal that majority of the writebacks from the LLC to main memory are generated by less than half of the LLC sets. For *sp*, 25% of the LLC sets, with the largest number of writebacks (i.e., frequent writeback sets), are responsible for 41.1% of the total number of writebacks. On the other hand, 12.5% of the sets, with the smallest number of writebacks (i.e., infrequent writeback sets), are accountable for only 5.3% of the total number of writebacks. In general, some sets tend to write

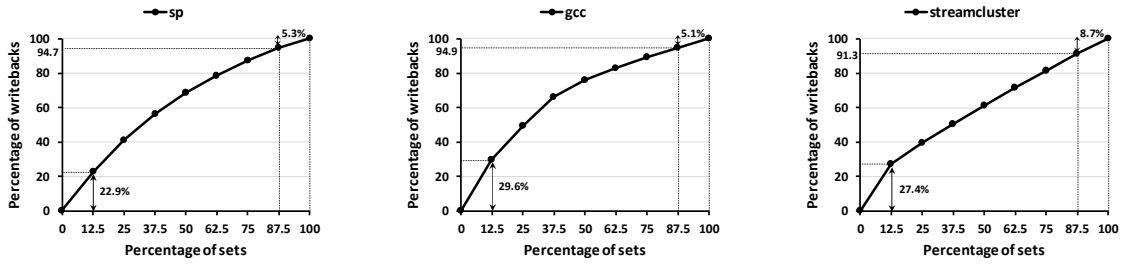


Figure 21. Cumulative distribution of writebacks over LLC sets. (*NOTE*: sets are sorted in a descending order based on their total number of writebacks).

back much more frequently than others. For example, since 22.9% of the writebacks for *sp* are performed by 12.5% of the frequent writeback sets, a frequent writeback set writes to main memory about four times more frequently than an infrequent writeback set, on average. Similarly, for *gcc*, 25% of the frequent writeback sets perform 49.6% of the writebacks while the same percentage of the infrequent writeback sets are responsible for only 10.8% of the total number of writebacks. Finally, for *streamcluster*, 20.7% of the writebacks are sourced from 6.3% of the frequent writeback sets.

Based on this observation, we propose a set balancing mechanism to reduce the number of writes to a PCM-based main memory system. More specifically, *WALL* aims to prevent a noticeable percentage of write traffic from reaching the PCM main memory by taking the non-uniform distribution of LLC set writebacks into account. The idea of avoiding the eviction of LLC sets' highly reused dirty blocks, which are likely to become eviction victims soon after being re-inserted in the cache (i.e., frequent writeback blocks), has been discussed in a recent study called *WADE* [80]. The WADE scheme partitions the blocks of each LLC set into two groups, "frequent writeback blocks" and "non-frequent writeback blocks" and tries to keep the frequent writeback blocks in the set to reduce the number of writes to the main memory. However, one major

shortcoming of WADE is that for every set, irrespective of its characteristics, it considers the non-frequent writeback blocks of the set as the only replacement candidates; while other sets might have clean, underutilized lines. Moreover, WADE is rather complex. In our study, a simple but effective writeback-aware replacement policy is also proposed to keep the frequent writeback blocks in the cache. Overall, *WALL* can reduce the number of LLC writebacks by considering the writeback behaviour of sets and their blocks at the same time.

5.3 *WALL* Overview

The *WALL* scheme comprises of a writeback-aware set balancing mechanism and a writeback-aware replacement policy. The set balancing mechanism classifies LLC sets into three categories, "writers" (i.e., frequent writeback sets), "non-writers" (i.e., infrequent writeback sets) and "neutral". To reduce the number of write requests to PCM, the non-writer sets are used to store the evicted dirty lines from the writer sets. Specifically, each writer set is partnered with a non-writer set (until there is no non-writer set left) and upon eviction of an LRU dirty line from the writer set, the line will be inserted into the set's partner instead of being written back to the main memory. Besides, our writeback-aware replacement policy further reduces the number of LLC writebacks by keeping the frequent writeback blocks in the cache.

5.4 Writeback-Aware Set Balancing Scheme

To decide whether a set is writer or non-writer, *WALL* monitors the number of writebacks, and accesses of each LLC set for a time window. Generally, a set is "writer" if the number of writebacks from it exceeds a certain threshold (i.e., τ_{high_wb}), but a set with relatively small number of writebacks is not necessarily a good partner set. A non-writer set must have enough

space to store the evicted dirty blocks of its writer partner without noticeable performance penalty. To measure the degree to which a set can hold its working set, we employ a saturating arithmetic miss counter (i.e., saturation counter) similar to that in a previous study [73]. For a *K*-way set associative cache, the working range of the saturation miss counter is from 0 to 2*K*-1 [73]. Upon every access, the saturation counter is incremented if the access results in a miss and decremented otherwise. To count the number of writebacks, we also use a saturating writeback frequency counter for each set that is incremented upon every writeback from the set. At the end of a monitoring period, we divide the writeback values by two (i.e., shift the counters right by one bit). This actually reduces the impact of set's writeback behaviour in the past on the type of the set in the current time period.

WALL considers a set "non-writer" if both of saturation and writeback counters are smaller than specific thresholds (i.e., $\boldsymbol{\tau_{sat}}$, $\boldsymbol{\tau_{low\_wb}}$). For a set with writeback frequency counter of *W* and saturation counter of *M*, the set is considered writer if ($W \geq \tau_{high\_wb}$), non-writer if ($M \leq \tau_{sat}$ & $W \leq \tau_{low\_wb}$) and neutral otherwise. We divide the total execution time of programs into epochs of 10^7 accesses to the LLC. On entering an epoch, the partnership between two sets can be easily broken if the writer set has no blocks in its non-writer partner. The thresholds are re-calculated and if there is a pair of writer and non-writer sets with no partners, they will be assigned to each other. Note that for a set with partner (i.e., a writer set with blocks in its partner or a non-writer set that holds some of its partners' blocks), we do not change the set type. Our experiments show that set types rarely change after an initial set type identification epoch and most partnerships remain intact. On the other hand, to avoid overfill a non-writer set, if the saturation counter of a non-writer set reaches $\alpha \times \tau_{sat}$, where α is 2 in our experiments, the insertion of blocks from its writer partner will be suspended until the set's saturation counter retrieves to values smaller than τ_{sat}. To reduce implementation complexities, we assume each writer set is partnered with only

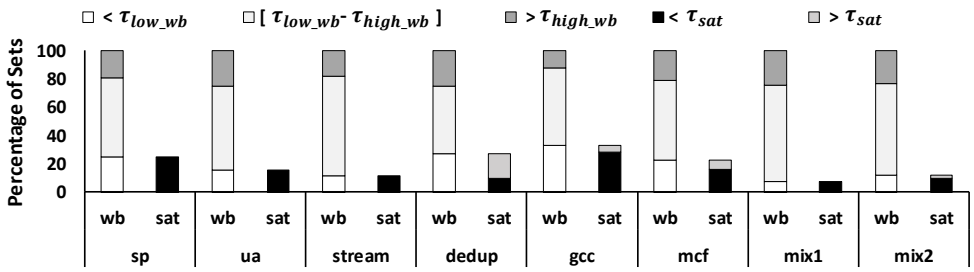


Figure 22. Distribution of LLC sets based on the thresholds (wb: writebacks; sat: saturation; *NOTE*: *sat* bars are only for the sets with wb < τ_{low_wb}).

one non-writer set. In other words, the number of the paired writer and non-writer sets is equal to the size of the smaller group of writer and non-writer sets. We later discuss how accesses to sets and their partners are handled in Section 5.6.

To determine the writeback thresholds, three simple steps are followed. First, the arithmetic mean of all the writeback values is computed and referred to as the overall average. Then, τ_{low_wb} is computed as the arithmetic mean of the writeback values smaller than the overall average. Finally, τ_{high_wb} is computed as the arithmetic mean of the writeback values larger than the overall average. Note that since these thresholds cannot guarantee an equal number of writer and non-writer sets, some writer sets may remain without a partner at the end. Moreover, we assume τ_{sat} = $K/4$ (i.e., K is the set associativity). Figure 22 depicts the distribution of sets based on the discussed thresholds for the workloads shown in TABLE XI (see Section 5.9.1). The results are based on the values obtained after the initial epoch of 10^7 accesses to the LLC. It should be noted that the saturation bars (i.e., bars with horizontal axis title of "sat") in the figure are only for the sets with writeback counters smaller than τ_{low_wb}. Our results show that the selected thresholds can distinguish different types of sets from each other effectively.

5.5 Writeback-Aware Replacement Policy

In addition to assigning partners to writer sets, *WALL* enables a writeback-aware replacement policy inside the LLC sets to further reduce the number of writebacks to the PCM. To keep the "frequent writeback blocks" in the cache, such blocks need to be identified first. We propose a much simpler yet effective method compared to the prediction scheme discussed in WADE [80], because such prediction schemes are usually complex and costly in terms of area, energy, and/or performance overheads.

The intrinsic definition of a frequent writeback block is a block that is frequently reused each time after being evicted from the cache. Generally, our scheme avoids the eviction of such blocks by giving the dirty victims a second chance to stay in the cache and be accessed again. To keep track of the dirty blocks that have been given a second chance, a one-bit flag called *FV* (i.e., Former Victim) is considered for each block. We assume the baseline replacement policy is LRU. When a replacement is needed in an LLC set, the dirty status bit of the LRU line (i.e., the eviction victim) is checked; if the LRU block is clean, it will be evicted from the cache but if the block is dirty, two scenarios are possible. First, the block is not a former victim (*FV* = 0). In this case, the line will be moved to the MRU position of the access stack and will be marked as a former victim (i.e., its *FV* flag will be set to '1'), this process will be repeated until finding an eviction victim. Second, the block is a former victim (*FV* = 1) and has become the eviction victim for the second time without being accessed. In this case, the block will be evicted from the cache. If a cache line with *FV* = 1 is accessed, its *FV* bit will be reset to '0'. The reason is that such block is likely to be a frequent writeback block. It should be noted that all these steps happen in parallel with the resolution of the miss, thereby there is no performance penalty.

We use the proposed replacement policy for the non-writer and neutral sets. Since writer sets usually have high miss rate values and often choose dirty blocks as eviction victims, for those sets, we use the baseline LRU replacement policy. However, on eviction of an LRU dirty block from a writer set with a partner, the block will be inserted into the set's partner.

5.6 Set Balancing Simple Partner Assignment and Access Management

For writer sets with large number of writebacks, changing the replacement policy may cause a non-trivial increase in the sets' miss rates. Hence, the writeback-aware replacement policy is applied to those sets that are not writer (i.e., neutral and non-writer sets). Instead, *WALL* virtually increases the associativity of a writer set by assigning a non-writer partner to it. We first propose a simple partner assignment strategy and access management for set-balancing LLC. The reason that we have excluded the neutral sets from the partnering process is that it is not beneficial to write from one set to another set with similar writeback or miss frequencies.

The partner of a writer set is selected randomly from the non-writer sets. To keep tracking the partners, a small direct-access remap table is introduced. The indices of the sets' partners are saved in the remap table, which is indexed by set indices. For a set with no partner, its own index is stored. When an LRU dirty line is evicted from a writer set with a partner, it will be inserted into the set's partner. To show whether a block in a non-writer set is repositioned from the set's writer partner or not, a one-bit flag called *RB* (i.e., Repositioned Block) is considered for each block. The remap table is also augmented with one-bit flag *P* (i.e., Partnered) for each set to show whether a writer set has any blocks in its partner or not. Upon an access to a writer set, if the access results in a miss, the remap table is checked, if *P* is '1', the set's partner will be accessed for the block; otherwise, main memory will be accessed as usual. If the access also misses in the set's

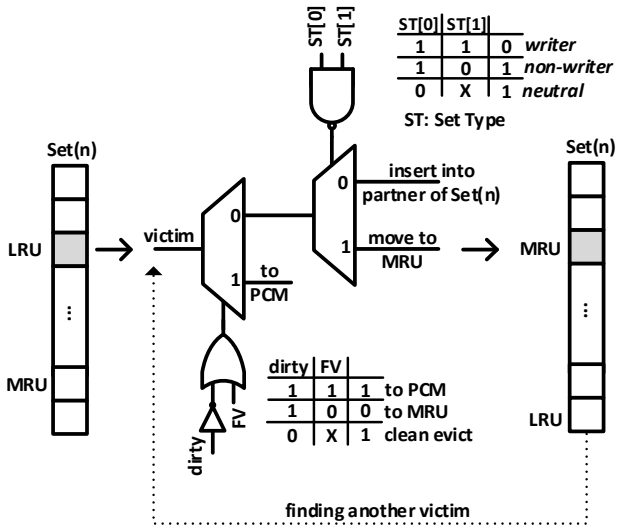


Figure 23. Design of WALL.

partner, main memory must be accessed. If all the repositioned blocks of a writer set get evicted from its partner, *P* flag of the writer set will be reset to '0'.

The design of *WALL* is depicted in Figure 23. On an eviction of a block from an LLC set, we first decide whether the block needs to be written back to the PCM or not. If not, depending on the set type, the block either remains in the set as MRU or will be written back into the set's partner. To specify the type of each set, a 2-bit register called *ST* is considered per set (i.e., the set is writer if *ST* = "11", non-writer if *ST* = "10", and neutral if *ST* = "00" or "01").

5.7 Extended Partner Assignment Strategies

The simple partner assignment strategy of *WALL* explained in the previous section cannot guarantee an equal number of writer and non-writer sets. In other words, some writer sets may remain without partners at the end of the partner assignment process. In this section, to efficiently

exploit the available storage space of the non-writer sets and to further reduce the number of writebacks from the writer sets, we propose three more partner assignment strategies called expansion, contraction and ConExp, which is a combination of **con**traction and **exp**ansion strategies.

5.7.1 Expansion Partner Assignment Strategy

The expansion strategy modifies the writeback thresholds to balance the number of writer and non-writer sets. Specifically, when the number of non-writer sets is considerably smaller than the writer sets, the expansion strategy finds the most eligible neutral sets to be added to the non-writer sets. On the other hand, when the number of writer sets is smaller, the expansion strategy assigns the remaining (without partners) non-writer sets, to the neutral sets that can benefit the most from the partnership.

To avoid enlarging the group of writer or non-writer sets unreasonably (i.e., causing more imbalance in the sizes of the two groups), we define an expansion condition; the expansion strategy is applicable only when the size of the larger group is at least $\beta\times$ the size of the smaller group. Our evaluations show that $\beta = 1.5$ provides the best balance between the number of writer and non-writer sets. The writeback thresholds of the expansion strategy are represented as $\tau_{exp\_low\_wb}$ and $\tau_{exp\_high\_wb}$. When the expansion condition is true, considering a neutral set with writeback frequency counter of W and saturation counter of M, the two possible scenarios are as follows: 1) If non-writer sets are fewer, the neutral set is considered "semi non-writer" if ($M \leq \tau_{sat}$ & $W \leq \tau_{exp\_low\_wb}$). To determine $\tau_{exp\_low\_wb}$, the arithmetic mean of the writeback values between τ_{low_wb} and the overall average is computed. 2) If writer sets are fewer, the neutral set is considered "semi writer" if $W \geq \tau_{exp\_high\_wb}$. The value of $\tau_{exp\_high\_wb}$ is also computed as the

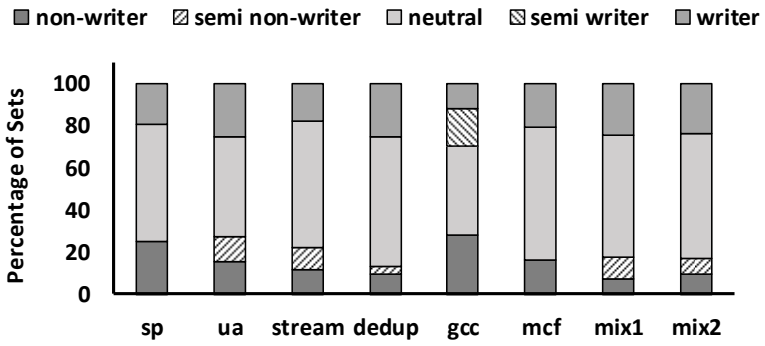


Figure 24. Distribution of LLC sets after applying the expansion strategy.

arithmetic mean of the writeback values between the τ_{high_wb} and the overall average. Figure 24 shows the distribution of sets with the expansion strategy after the initial monitoring epoch.

The expansion strategy starts by assigning the original writer and non-writer sets as partners. Then, it continues by assigning the remaining writer/non-writer sets to the semi non-writer/semi writer sets until no semi non-writer/non-writer set remains without a partner.

5.7.2 Contraction Partner Assignment Strategy

Since the number of writer and non-writer sets are usually not the same, some sets remain without a partner. On the other hand, it is beneficial to always keep the writer sets with the largest number of writebacks, or "super writer sets", and non-writer sets with the smallest number of writebacks, or "super non-writer" sets, included in the partner assignment process.

The contraction strategy distinguishes the super writer and super non-writer sets from the rest of the sets using two writeback thresholds called $\tau_{con\_high\_wb}$ and $\tau_{con\_low\_wb}$. Specifically, a

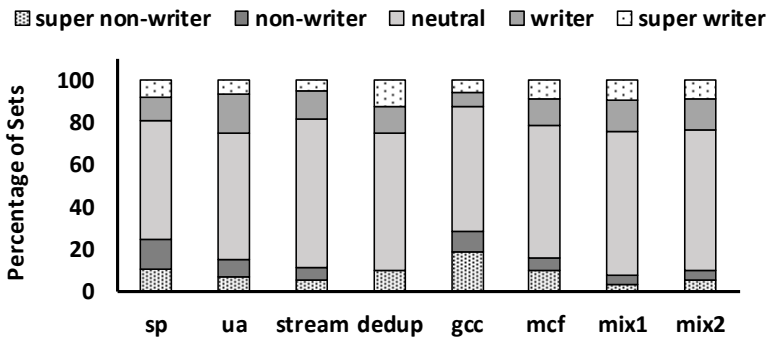


Figure 25. Distribution of LLC sets after applying the contraction strategy.

non-writer set with writeback counter of W is considered super non-writer if $W \leq \tau_{con\_low\_wb}$ and a writer set with writeback counter of W is considered super writer if $W \geq \tau_{con\_high\_wb}$. To determine the $\tau_{con\_low\_wb}$, the arithmetic mean of the writeback values smaller than the τ_{low_wb} is computed. Also, $\tau_{con\_high\_wb}$ is computed as the arithmetic mean of the writeback values larger than the τ_{high_wb}. Figure 25 illustrates the distribution of sets with the contraction strategy after the initial monitoring epoch. The contraction strategy starts from assigning partners to super writer and super non-writer sets and then continues by assigning partners to the remaining writer and non-writer sets.

5.7.3 Contraction-Expansion (ConExp) Strategy

There are two main shortcomings with the expansion strategy. First, since the semi non-writer sets are less effective than the non-writer sets in reducing the number of writebacks, assigning them as partners to the super writer sets should be avoided. Second, although the expansion strategy tries to balance the number of writer and non-writer sets, the super writer or super non-

TABLE IX. Total Storage Overhead.

Type	Storage	Type	Storage
FV per block	16 KB	Saturation counter (6-bit) per set	3 KB
RB per block	16 KB	Writeback counter (8-bit) per set	4 KB
P per set	0.5 KB	Remap table	6 KB
ST per set	1 KB	**TOTAL**	**46.5 KB**

writer sets may still be excluded from the partner assignment process. To alleviate these problems, we propose the ConExp strategy, which is a combination of the contraction and expansion strategies. ConExp assigns partners, in order, to the super sets, then to the original writer and non-writer sets, and finally to the semi sets, until no non-writer or semi non-writer set remains without a partner. It is worth mentioning that the semi and super writer sets are represented (treated the same) as the writer sets and the semi and super non-writer sets as the non-writer sets. Hence, these strategies do not require any additional hardware overheads.

5.8 Overhead Analysis

TABLE IX summarizes the storage overhead of *WALL*. The total storage overhead of *WALL* is less than 0.6% of the LLC capacity. It is worth mentioning that this overhead is about half of that of WADE [80].

Calculating the writeback thresholds and updating the remap table (i.e., pairing writer and non-writer sets) are only performed once at the end of each epoch. Hence, their performance impacts are negligible.

TABLE X. System Configuration.

Processor and on-chip Caches	
Cores	8 cores, out-of-order, 2.0 GHz
L1-I/D	Split 32KB I/D-cache/core, 4-way, 8-MSHR, 2-cycle hit
L2	256KB/core, 8-way, 12-MSHR, 12-cycle hit
L3 (LLC)	SRAM: Shared, 8MB, 32-way, 32-MSHR, 35-cycle hit
Coherency	MOESI directory, 2×4 grid packet NoC, XY routing
Main Memory	
PCM	4GB, 4 Channel, 1 rank/channel, 4 banks/rank, 400 MHz t_{SET}= 150ns, t_{RESET}= 100ns, t_{RCD}= 120ns Cell endurance = 32×106 writes
MC	Four controllers, Open page, 32-entry queues (one read queue and one write queue), Write drain threshold: high = 80%, low = 50%, Address mapping: page interleaving

5.9 Results

5.9.1 Methodology

For this work, we model an 8-core processor using the GEM5 full-system simulator integrated with NVMAIN. The system configuration of our experiments is shown in TABLE X. The PCM configurations are generated by NVSIM [16] and CACTI [8], the cell parameters used in NVSIM are based on the projections by [12]. The benchmarks used in this study are chosen from NAS [60], SPEC CPU2006 and PARSEC [6] as depicted in TABLE XI. The selected benchmarks are some of the memory-intensive workloads from the three benchmark suites. For all the workloads, we use either sampled reference or native input sets to represent a real-world execution scenario and run the applications for two billion instructions, after two billion instructions for cache warm-up phase.

We compare *WALL* with 1) *Baseline* that uses the LRU replacement policy, 2) *Baseline double-way*, a baseline cache of the same size with double the associativity, and 3) *WADE*, which is the

TABLE XI. Evaluated workloads characteristics.
(RPKI/WPKI: main memory Reads/Writes Per 1000 Instructions)

Workload	RPKI	WPKI	Workload	RPKI	WPKI
from the NAS benchmarks (*8-Thread*)					
sp	4.98	2.55	ua	3.12	2.67
from the PARSEC benchmarks (*8-Thread*)					
stream	24.4	0.21	dedup	11.5	8.32
from the SPEC CPU2006 benchmarks					
8×gcc	7.42	1.59	8×mcf	43.5	9.02
mix1	2.91	1.90	mix2	12.7	4.21
mix1: 4×lbm, 4×bzip			mix2: 4×cactusADM, 4×leslie3D		

scheme proposed in [80]. In PCM, we always prioritize reads over writes if write queue is less than 80% full.

5.9.2 LLC Writeback Reduction

Figure 26 shows the writeback reduction of *WALL* for the different types of LLC sets. The results are normalized to the baseline scheme. It is worth mentioning that any writeback of a writer set's lines from its partner is considered for the original writer set.

The *WALL* scheme when using the simple partner assignment strategy achieves an average of 26.6% writeback reduction, compared to the Baseline scheme. The three main reasons for this reduction are: 1) The efficacy of the set type identification process. 2) The capability of the writeback-aware set balancing scheme in reducing the writer sets' writebacks; the number of writebacks originated from writer sets is reduced by 39.5%, on average (i.e., from 33.4% to 20.1% of the Baseline total writebacks), while the writebacks originated from non-writer sets are increased slightly from 10.4% to 13.1% of the Baseline total number of writebacks. 3) The proposed writeback-aware replacement policy has been able to reduce the writebacks originated from the

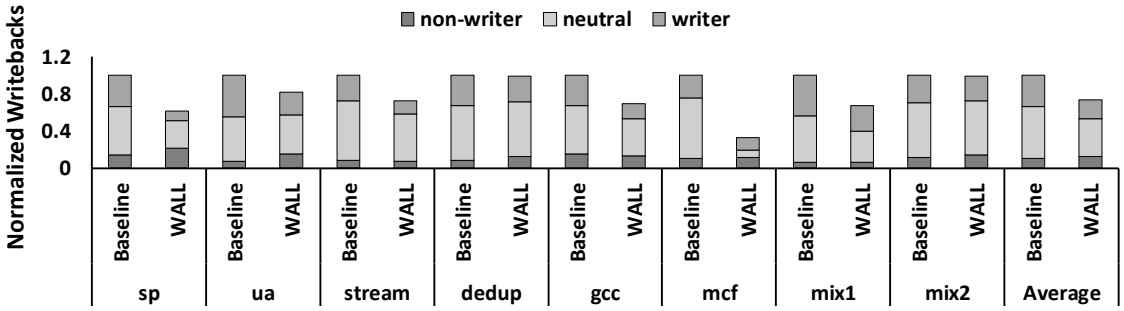


Figure 26. *WALL*'s normalized LLC writebacks reduction.

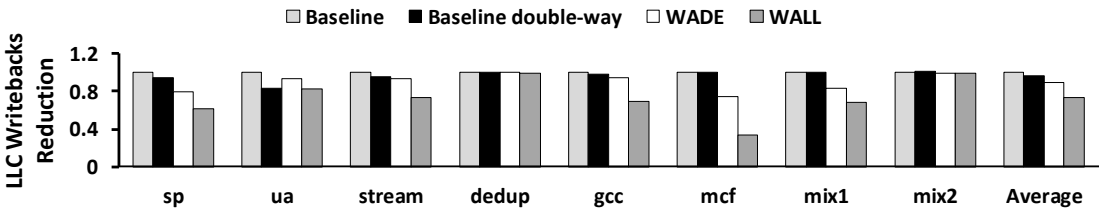


Figure 27. Writebacks reduction of evaluated schemes.

neutral sets by 28.6%, on average (i.e., from 56.2% to 40.1% of the Baseline total writebacks). It is worth noting that for some of the benchmarks, the proposed replacement policy has also been able to reduce the number of writebacks originated from the non-writer sets.

Figure 27 compares the normalized writebacks reduction of the evaluated schemes. Compared with Baseline double-way and WADE, *WALL* reduces the number of writebacks by 23.3% and 16.4%, on average, respectively. Based on the results, duplicating the set associativity is not very helpful in reducing the number of LLC writebacks; our baseline implementation is 32-way and increasing the associativity beyond that does not cause a significant improvement.

Figure 28 compares the writeback reduction of the extended partner assignment strategies. The results are normalized to the simple partner assignment strategy. The expansion strategy can

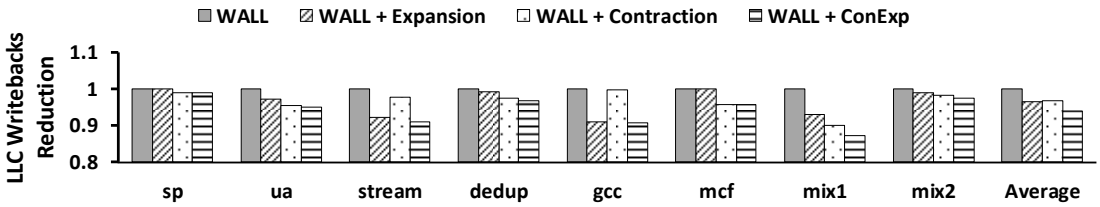


Figure 28. Writebacks reduction of *WALL* with different partner assignment strategies.

reduce the number of writebacks by 3.6%, on average (by up to 9.2% for gcc). This reduction is caused by including more writer (or semi writer) sets in the partner assignment process. The contraction strategy can reduce the number of writebacks by 3.4%, on average (by up to 4.6% for ua). The reason for this reduction is the fact that sets with highest write frequency (super writer sets) are always assigned a partner, which is in the super non-writer sets (i.e., one that has the smallest write frequency). ConExp, which uses the benefits of both strategies, can reduce the number of writebacks by 5.9%, on average (by up to 12.7% for mix1), compared to the simple partner assignment strategy. In other words, ConExp strategy can reduce the number of writebacks by 30.9%, on average, compared to the Baseline scheme.

5.9.3 LLC Miss Rate

Figure 29 shows the normalized MPKI (Misses Per Kilo Instructions) of *WALL* for different types of LLC sets. The accesses that result in a hit in a writer set's partner are considered for the original writer set.

Figure 30 shows the normalized MPKI of the evaluated schemes. Results reveal that *WALL* when using the simple partner assignment strategy reduces the MPKI by 2.4%, on average, compared to the Baseline. This reduction is mainly because *WALL* stores the frequently reused

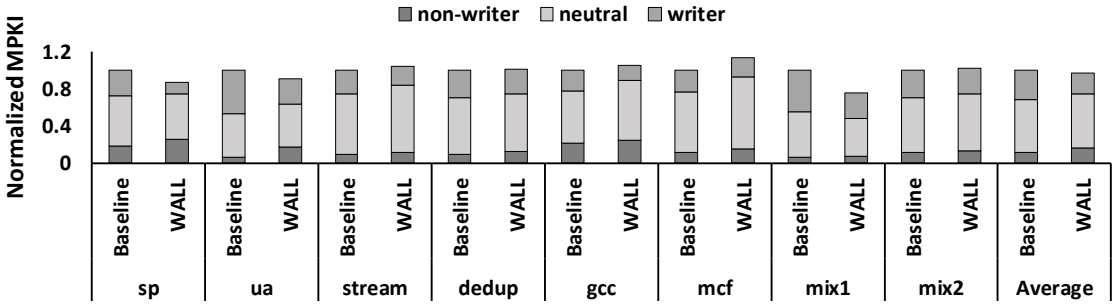


Figure 29. *WALL*'s normalized MPKI.

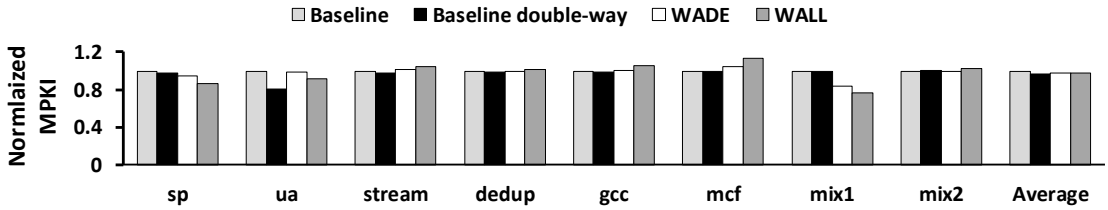


Figure 30. MPKI of evaluated schemes.

dirty blocks of writer sets in the non-writer partners and performs similarly to doubling the associativity of those cache sets. More specifically, the MPKI of writer sets is reduced by 27.8%, on average (i.e., from 30.7% to 22.1% of the Baseline total MPKI). On the other hand, the MPKI is increased from 12.0% to 16.2% of the Baseline total MPKI for the non-writer sets and from 57.3% to 59.1% for the neutral sets, on average, respectively. For some cases, the writeback-aware replacement policy has incurred miss penalties by evicting the clean lines that are later reused more frequently than the saved dirty blocks (i.e., those that have been given a second chance). For other cases, the dirty lines kept in LLC by the writeback-aware replacement policy are re-referenced more than the lines evicted by them instead. It should be noted that the penalty is small because we give only a second chance to the dirty victims of the sets. The set-balancing

scheme reduces the MPKI of writer sets at the cost of increasing the MPKI of non-writer sets. However, since the non-writer sets are usually underutilized, the increase is small. Compared with the LRU replacement policy (used for both Baseline and Baseline double-way), our scheme has higher management cost and increases the MPKI slightly for some benchmarks. However, our evaluations reveal that it does not have a noticeable negative impact on the cache miss rate. Compared with Baseline double-way, *WALL* increases the MPKI by 1.0%, on average. However, *WALL* reduces the MPKI by 0.3%, on average, compared to WADE because WADE does not distinguish different set types.

5.9.4 Energy Comparison

Figure 31 shows the normalized energy of the PCM main memory for the evaluated schemes. *WALL* when using the simple partner assignment strategy can save main memory's energy by 19.2%, on average, compared to the Baseline. Compared with Baseline double-way and WADE, *WALL* reduces the energy consumption of the PCM main memory by 16.5% and 11.3% on average, respectively. The energy consumption of writing to PCM is much higher than that of reads. Hence, the energy saving is mainly due to the reduction in the number of write requests issued to PCM by *WALL*. That is also the reason that Baseline double-way exhibits a higher power consumption compared to *WALL*; Baseline double-way experiences a smaller MPKI, but a higher rate of writebacks. Unlike WADE, which uses complex prediction schemes, the energy consumption of the logic components added to LLC by *WALL* is negligible.

Figure 32 shows the normalized energy of the main memory for the extended partner assignment strategies. Compared to the simple partner assignment strategy, the expansion and the contraction strategies can save main memory's energy consumption by 2.7% and 3.0%, on average, respectively. The ConExp strategy can reduce the main memory's energy by 4.8%, on

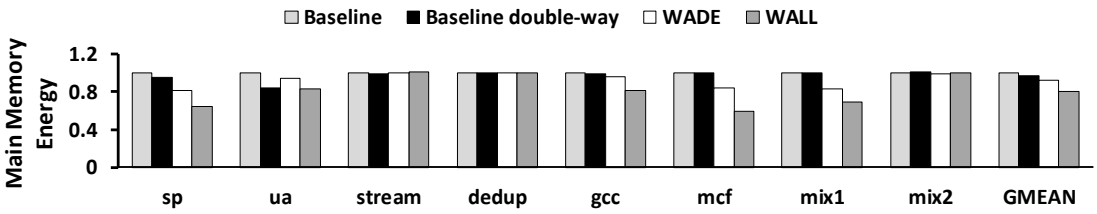


Figure 31. Normalized main memory energy.

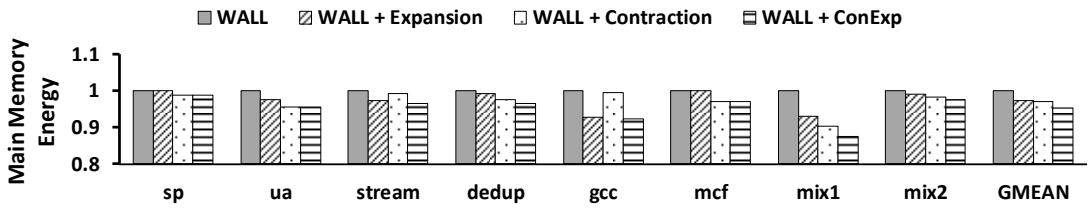


Figure 32. Main memory energy of *WALL* with different partner assignment strategies.

average, compared to the simple partner assignment strategy. In other words, ConExp strategy can reduce the energy consumption of the PCM main memory by 23.1%, on average, compared to the Baseline scheme.

5.9.5 Performance Comparison

Figure 33 compares the normalized system IPC of the evaluated schemes. The overhead of the second searches in partners of the *writer* sets is considered in our experiments. *WALL* when using the simple partner assignment strategy improves performance by 6.7% on average, compared to the Baseline scheme. Compared with Baseline double-way and WADE, *WALL* improves system performance by 4.9% and 3.2% on average, respectively. The latency of writing to PCM is much higher than that of reads. Hence, the reduced average access latency of the PCM main memory is

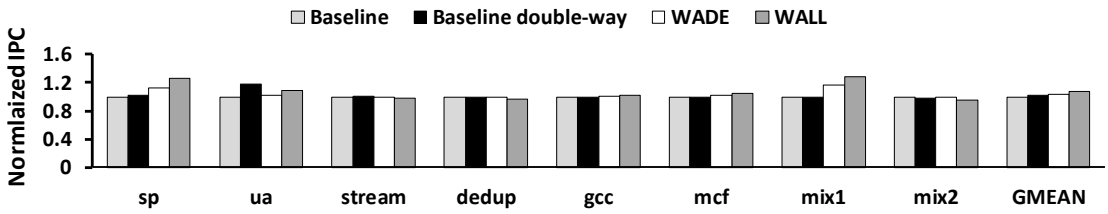


Figure 33. Normalized IPC.

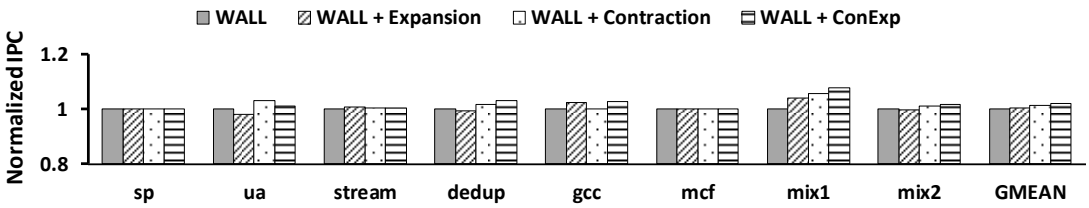


Figure 34. Normalized IPC of *WALL* with different partner assignment strategies.

the major reason for the performance improvement. The results indicate that *WALL* can reduce the PCM average access latency by 12.6%, on average, compared to the Baseline. The reduction is due to reducing the PCM write traffic and thus the queuing delay of the PCM read requests.

Figure 34 illustrates the normalized system IPC of *WALL* with the extended partner assignment strategies. Compared to the simple partner assignment strategy, the expansion and contraction strategies can improve the system performance by 0.7% and 1.5%, on average, respectively. ConExp, which uses the benefits of the both strategies, can improve performance by 2.1%, on average, compared to the simple partner assignment strategy. Compared to Baseline, ConExp strategy can improve performance by 8.7%, on average.

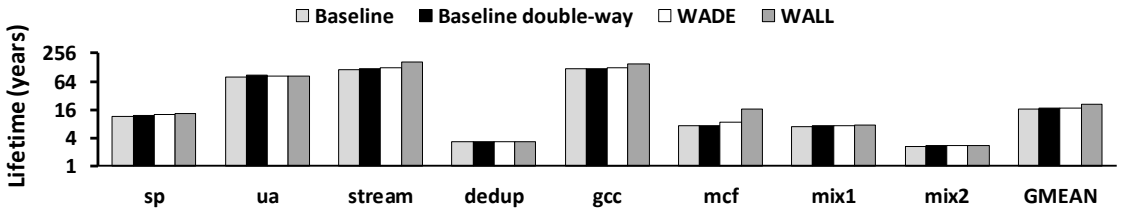


Figure 35. Lifetime enhancement (years, log scale).

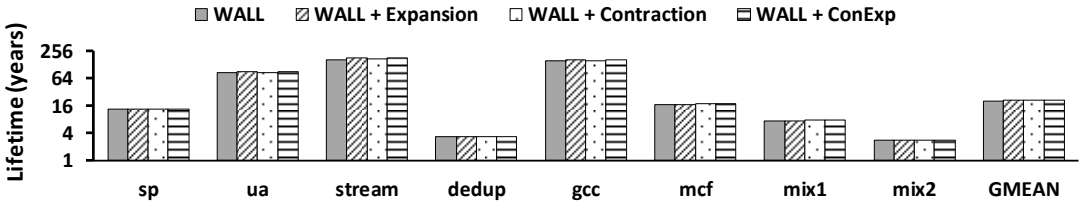


Figure 36. Lifetime enhancement of *WALL* with different partner assignment strategies.

5.9.6 PCM Lifetime Enhancement

Figure 35 compares the PCM lifetime enhancement of the evaluated schemes. The lifetime is calculated based on the analytical model in [70] explained in the Background Chapter Section 2.4.1. It should be noted that PCM lifetime is inversely proportional to the writes per cycle or write rate (GBps). *WALL* when using the simple partner assignment strategy can enhance PCM lifetime by 1.25×, on average, compared to the Baseline. Compared with Baseline double-way and WADE, *WALL* enhances the PCM lifetime by 1.21× and 1.17× on average, respectively. Alleviating the write traffic sent out to the PCM main memory by *WALL* is the reason for the lifetime enhancement.

Figure 36 shows the PCM lifetime enhancement of the extended partner assignment strategies. ConExp can enhance PCM lifetime by 1.04×, on average, compared to the simple

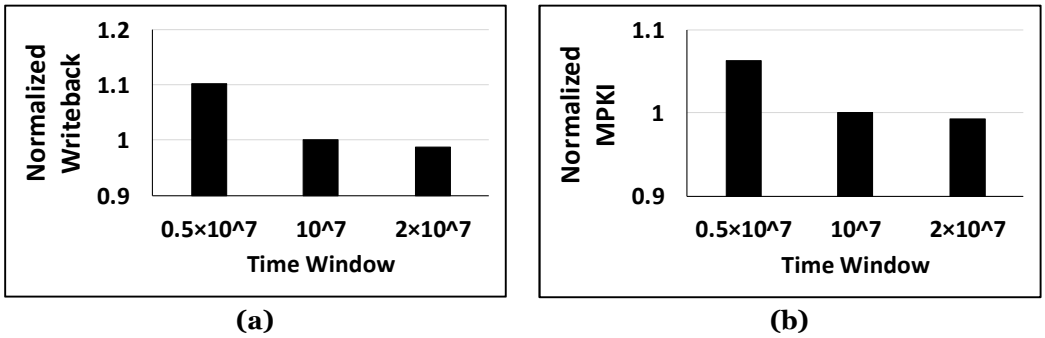


Figure 37. *WALL* results for various window sizes: (a) writeback reduction of *WALL* for three different time window sizes; (b) MPKI of *WALL* for three different time window sizes.

partner assignment strategy. In other words, ConExp strategy can enhance PCM lifetime by 1.29× compared to the Baseline scheme.

5.9.7 Impact of Time Window Size

For experimental results presented so far, the set types are checked and adjusted with an epoch of 10^7 LLC accesses, because our experiments indicate that this provides the best balance between the writeback reduction and storage overhead. Next, we discuss the impact of time window size. We choose to use the number of LLC accesses instead of the number of cycles because *WALL* works based on the number of writebacks/accesses of the sets and epochs with the same number of cycles may have very different number of writebacks/accesses. In other words, to obtain a consistent approach among the epochs, we choose the number of LLC accesses rather than cycles. Figure 37 compares the average writeback and miss rate values of *WALL* + ConExp for three different time window sizes.

The results are normalized to 10^7 LLC access time windows. Our goal is to reduce the number of writebacks while keeping the storage overhead at minimum. The evaluations show that using

time windows of 0.5×10^7 accesses results in 10.3% more writebacks and 6.3% higher miss rate compared to time windows of 10^7 accesses. The reason is that time windows of 0.5×10^7 accesses are not long enough, resulting in incorrect identification of some set types. Identifying a neutral set as non-writer and assigning a writer partner to it can increase its writeback and miss rate values because the set will not have enough space for its writer partner's blocks. Moreover, identifying a writer set as neutral can also increase its writeback and miss rate values because writer sets need to be managed differently as we explained in Section 5.4. Finally, identifying a neutral set as writer or a non-writer set as neutral may prevent us from using the set-balancing scheme efficiently.

Our results also show that for time windows of 2×10^7 accesses, larger writeback counters are required; otherwise counters will saturate causing sets types to be identified incorrectly. However, even when larger writeback counters are used for time windows of 2×10^7 accesses, the reduction in the number of writebacks is small (i.e., only 1.2%) compared to time windows of 10^7 accesses. Hence, the selected time windows of 10^7 LLC accesses gives us the best balance between the writeback reduction and storage overhead.

5.10 Conclusions

In this chapter, a novel writeback-aware LLC management scheme is proposed to reduce the number of LLC writebacks to a PCM based main memory to improve its energy efficiency and lifetime.

We first investigated the non-uniformity of the LLC sets writebacks and proposed a writeback-aware set-balancing mechanism based on that. To implement the set-balancing mechanism, we first proposed a simple partner assignment strategy. Then, to further optimize our proposed

WALL scheme, we proposed three novel partner assignment strategies, called contraction, expansion, and ConExp to pair sets with different behaviours more efficiently compared to the simple partner assignment strategy. In addition, we proposed a simple but effective writeback-aware replacement policy to keep the frequently reused dirty lines of the sets in the cache. Our evaluation results revealed that *WALL* can achieve a significant reduction in the total number of writebacks to PCM; thereby improving the system performance, energy efficiency and PCM lifetime.

CHAPTER 6

A DYNAMIC PAGE SWAP MANAGEMENT SCHEME FOR HYBRID DRAM/NVM MAIN MEMORY SYSTEMS

6.1 Introduction

It is now well known that DRAM can no longer satisfy the increasing memory capacity and bandwidth demands of many modern-day applications in the era of "Big Data" [27, 50]. Non-volatile memory technologies such as PCM, ReRAM and STT-RAM have been explored as potential replacements to DRAM due to their higher density, better scalability, and lower leakage power [79, 81, 83]. However, NVMs have also a number of drawbacks. First, these technologies have higher access latency and energy compared to DRAM [15]. Second, NVM cells have limited write endurance, which can adversely affect their lifetime [15, 85]. Hybrid memory systems, which incorporate both NVM and DRAM, enable systems to benefit from the large capacity of an NVM and lower access latency and energy of a DRAM. In a hybrid memory system, DRAM can be used either as a hardware-managed cache for NVM, or as part of a flat address space hybrid main memory. To be able to benefit from the total visible capacity of both DRAM and NVM (i.e., to avoid data replications) and their aggregate bandwidth, some recent studies have focused on flat address space hybrid main memories [14, 36, 37, 67, 74, 75]. In this work, we consider a hardware-managed flat address space hybrid main memory. A flat address space hybrid main memory can also be managed by the OS. However, page migrations under OS control can incur significant performance penalties [75].

In flat address space hybrid memories, DRAM has a limited capacity. Moreover, memory access behaviour of programs changes during execution. Hence, data may need to be migrated (i.e., swapped) between DRAM and NVM to take advantage of performance benefits of DRAM. Since hardware requires meta-data storage to keep track of the migrated data, migrations are typically performed at a coarse granularity (e.g., memory pages). Page migrations are costly. Therefore, a main challenge in flat memories is correctly deciding which migrations are beneficial to performance. Typically, such decisions have to be made dynamically at run-time [37, 67, 74]. Existing schemes consider the changing memory access patterns of programs when making migration decisions. However, they all suffer from one major limitation. In those studies, the memory space is statically partitioned into "swap groups" and only DRAM (i.e., fast memory) and NVM (i.e., slow memory) pages that belong to the same swap group can be swapped with each other. However, within a given interval, a swap group may contain more "high frequently accessed" pages than the number of DRAM segments (i.e., each segment contains a memory page) assigned to it (i.e., "swap group associativity"). This can cause frequent back and forth migrations of those pages between DRAM and NVM. Meanwhile, pages in another swap group may all be "low frequently accessed". Hence, most of the page migrations, which are very costly in terms of performance and energy, can be avoided by dynamically adjusting the structure of the swap groups based on programs behaviour.

In this study, to address the limitation of statically structured swap groups, we propose *DynaSwap* to dynamically associate swap groups with each other in such a way that a swap group with many high frequently accessed pages can benefit from the DRAM space of a swap group with low frequently accessed pages. In other words, unlike previous studies that create swap groups solely based on the physical address of memory pages, we try to create swap groups based on their access patterns. Assigning enough DRAM segments to a swap group based on its demand (i.e., its

access patterns in the current interval) can improve performance and energy efficiency of the memory system by reducing the number of unnecessary swaps. Our experimental results show that *DynaSwap* can efficiently utilize DRAM capacity and improve the overall performance and main memory energy efficiency by 30.1% and 13.5% on average, respectively, compared to a state-of-art baseline design.

6.2 *DynaSwap* Overview

In our design, swap groups are dynamically merged with each other so that a swap group with many frequently accessed pages can benefit from the DRAM resources of a swap group with rarely accessed pages. Moreover, to be able to adjust the structure of the swap groups and initiate page swaps ahead of time and off critical path, we employ the LSTM (i.e., Long Short-Term Memory)-based address predictor proposed in [3] to predict sequence of future LLC miss (i.e., main memory access) addresses. By doing so, the access latency of the incoming requests can be reduced. The address predictor relies on the deep recurrent neural network models to predict a sequence of future LLC miss addresses using a sequence of past LLC miss addresses. More specifically, the address predictor uses a sequence of 10 past memory accesses (for each access, the concatenation of the miss causing instruction's PC and the miss address is used) as input to predict a sequence of 10 future memory accesses (please refer to [3] for more details).

6.3 Baseline Organization

In this work, we use an organization similar to PoM [75] as our baseline. However, since the direct-mapped structure of PoM forces pages of a swap group to compete for a single DRAM segment, we first extend the PoM organization so that it can support associativity. To keep track of the swapped pages (i.e., pages that are relocated from their original OS-allocated location), a

Page Remapping Table (*PRT*) is required. Upon every access to the main memory, the remapping table needs to be accessed first to determine where to fetch the requested data from (i.e., the original or the hardware-remapped address). Hence, to avoid incurring significant performance penalties, a PRT cache (*PRTc*) is also usually employed.

We define a *K*-way set-associative swap group as a swap group that consists of *K* segments in DRAM and *M*×*K* segments in NVM (i.e., assuming NVM is *M* times larger than DRAM). Since in a PoM-like PRT structure, the information about all pages of a swap group reside within a single PRT entry, increasing the associativity of swap groups beyond a certain point comes at the cost of a significant increase in the PRTc area and latency overheads. Hence, for our baseline organization, we consider an associativity of *two* and then propose a scheme that dynamically increases (i.e., doubles) the associativity of swap groups based on their demand.

6.4 Page Classification and Monitoring

We classify the memory pages into two categories: high- and low-frequently accessed. By doing so, in a swap group, a high frequently accessed page in NVM can simply be swapped with a low frequently accessed page in DRAM.

The access frequency of a page in future can be estimated based on its access frequency in the past [36, 37]. More specifically, a memory page that observes a burst of accesses at some point during the execution of a program, will probably observe the same burst of accesses if it is re-used after a while in future. Hence, we consider a page "high frequently accessed" if it is now *being actively used* and *has seen a large number of accesses in the past*. To monitor pages access patterns, we use a Page Monitoring Table (*PMT*). Each PMT entry contains the Page Physical Address (*PPN*) and an Access Counter (*AC*). *AC* is a saturating counter that is incremented by one

upon observing an LLC miss to the page. Since PMT is too large to be stored on-chip, we keep a PMT cache (*PMTc*) in the memory controller. PMTc can also be used as a temporal filter to identify periods when pages are actively used; a page is being actively used if its entry is resident in PMTc.

To categorize pages, we augment each PRT entry with two one-bit flags, called Actively Accessed (*AA*) and Frequently Accessed (*FA*) per page location. The *AA* flag of a page is set to '1' upon inserting its entry into the PMTc and reset to '0' upon its eviction. Moreover, the *FA* flag of a page is set to '1' when its *AC* reaches a certain threshold (*access threshold*). The access threshold is set so that the attainable savings of a swap would be more than its costs. Upon eviction of a page entry from PMTc, the *FA* flag of the page is reset to '0' only if the value of *AC* is smaller than the access threshold. The value of *AC* is then divided by 2 to maintain information on history of accesses. The *AA* and *FA* flags show whether the page is being *actively* and *frequently* used or not; a page is considered high frequently accessed if both its *AA* and *FA* flags are '1'. On the other hand, we prefer low frequently accessed pages with both *AA* and *FA* equal to '0' as swap candidates.

6.5 Swap Group Classification

Our experiments show that within a given interval during the execution of a program, only a small portion of the main memory is frequently accessed. For example, when running *mcf* from SPEC2006 benchmark suite, in every interval of one million LLC misses, only 24.4% of the accessed swap groups (i.e., swap groups that are accessed at least once during program execution) have received more than 200 accesses on average (please refer to Section 6.9.1 for details of our simulation environment). In other words, while a small percentage of the swap groups service a large portion of the accesses within an interval, other swap groups are not frequently accessed.

Based on this observation, we classify the swap groups into two categories; SATURATED (*SAT*) and NON-SATURATED (*NON-SAT*).

A swap group is recognized as "SAT" if its number of high frequently accessed pages is larger than the swap groups associativity. Otherwise, the swap group is recognized as NON-SAT. In other words, while a NON-SAT swap group does not utilize its DRAM resources efficiently, a SAT swap group requires more DRAM segments than its baseline associativity. Hence, we double the associativity of a SAT swap group by associating a NON-SAT swap group as "partner" to it. This way, the SAT swap group can use the DRAM segments of its partner as auxiliary fast storage units for storing its high frequently accessed pages. To be able to create a partnership that can provide enough DRAM space for a SAT swap group, we only use NON-SAT swap groups with no high frequently accessed pages as partners.

Figure 38 illustrates an example of how a swap group is recognized as SAT and assigned with a NON-SAT partner. In this example, page *P1* with *FA* = 1 (i.e., a page that is likely to receive a large number of accesses) is in NVM within swap group *SG1*. It is predicted to be accessed soon (by the LSTM-Based Address Predictor). Hence, *DynaSwap* tries to prefetch *P1* into DRAM. However, since all pages in DRAM segments of *SG1* are high frequently accessed (i.e., there are no swap candidates inside *SG1*), *SG1* is recognized as SAT. *SG2,* which is a NON-SAT swap group with no high frequently accessed pages, is assigned as the partner to *SG1*. Then, *P1* is swapped with *P2*, which is a low frequently accessed page in *SG2's* DRAM.

6.6 Swap Group Partner Assignment

To be able to find a NON-SAT partner for each SAT swap group, we keep a one-bit flag called Swap group Type (*ST*) per swap group in the memory controller; *ST* = 0 represents a NON-SAT

swap group with no partner. The value of *ST* is initialized to '0' and is set to '1' if any page in the swap group becomes high frequently accessed or the swap group is assigned as a partner. *ST* of a swap group is reset to '0' when its partnership ends, and all of its pages become low frequently accessed. Whenever a swap group is recognized as SAT, we randomly choose a swap group with *ST* = 0 as its partner. For a 512MB DRAM with 4KB memory pages and swap group associativity of two, the area overhead of keeping *ST* flags is 8KB. Moreover, we keep the index of a swap group's partner in a register in the swap group's PRT entry. For a swap group with no partner, its own index is stored in the register.

To identify the pages that belong to partner of a swap group, we augment PRT with a one-bit flag called Displaced Page (*DP*) per page location: *DP* = 1 indicates that the page is repositioned from its original swap group. For displaced pages (i.e., with *DP* = 1) we use slow swaps. For example, in Figure 38, if a page, say *P3*, in *SG2* needs to be moved to DRAM, and it has to go to the place currently occupied by *P1*, we first swap *P1* and *P2* to put *P1* back into its original location. If *P2* becomes high frequently accessed while it is in NVM, it is swapped with *P1* back into its original location in *SG2*'s DRAM. For these swaps, we perform an *optimized slow swap* proposed in [37].

We enable a swap group to push pages of its partner out of its DRAM space if it needs its own DRAM resources at any point. Specifically, if 1) a page in a NON-SAT swap group with a partner becomes high frequently accessed and 2) no low frequently accessed pages can be found in the swap group's DRAM, the page will be swapped with a displaced page from the swap group's partner (i.e., page with *DP* = 1). Based on our experiments, this scenario does not happen frequently. Moreover, if a page with *DP* = 1 in DRAM becomes low frequently accessed, we proactively migrate it back into NVM in its original swap group when the memory is idle. The partnership between two swap groups is broken if no pages with *DP* = 1 remain in any of the swap

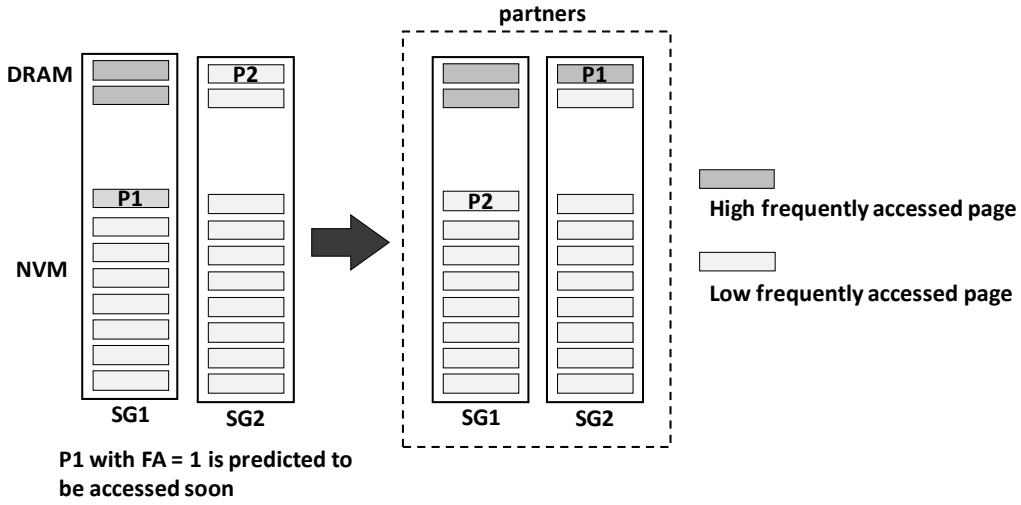


Figure 38. An example illustrating how two swap groups are partnered.

groups. It is worth mentioning that if a SAT swap group with a partner becomes NON-SAT, *DynaSwap* does not break the partnership between the two swap groups. Instead, it prefetches the high frequently accessed pages of the swap group into its own DRAM space (not its partner's) until it becomes SAT again.

6.7 Swap Group Access Management

The PRTc (PRT in case of a PRTc miss) needs to be accessed on every main memory access [75]. Upon an access to a page, if we cannot find the page (i.e., *tag* of the page with *DP* = 0) in its corresponding swap group's PRTc entry, we also need to access the PRTc entry of the swap group's partner (i.e., for page's *tag* with *DP* = 1). It should be noted that we access the PRTc entries at the time when the page is predicted to be accessed. Hence, these accesses are not performed on the critical path of program execution when the prediction is accurate, which is the case most of the

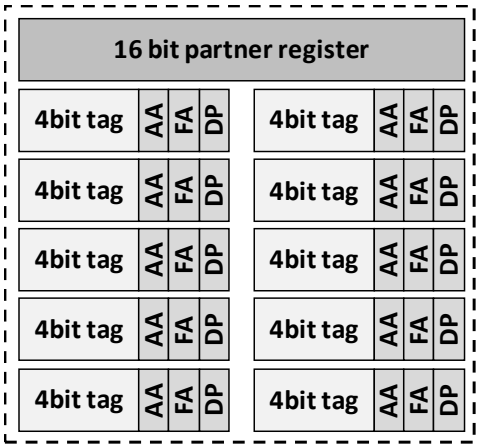


Figure 39. The structure of a PRT entry of *DynaSwap*; fast swaps are supported.

time (please refer to Section 6.9.2 for more details). The structure of a PRT entry, when fast swaps are supported, is shown in Figure 39. In this figure, the partner register size is based on a 512MB DRAM with page size of 4KB and swap group associativity of two.

If a page with *FA* = 1 and *DP* = 0 in NVM is predicted to be accessed soon, we first try to swap the page with a low frequently accessed page or a page with *DP* = 1 in the swap group's DRAM. If we cannot find any swap candidate inside the swap group itself, we look for one in the swap group's partner (we assign a partner to the swap group if it does not have any).

6.8 Storage Overhead

The total storage overhead of *DynaSwap* is shown in TABLE XII. The total storage overhead of the memory controller is 104KB, which is very small. Moreover, *DynaSwap* occupies only less than 0.8% of the DRAM space.

TABLE XII. *DynaSwap*'s total storage overhead.

in Memory Controller	**Storage**
PRTc	32 KB
PMTc	64 KB
Swap Type (*ST*) flags	8 KB
Total in-MC Overhead	**104 KB**
in DRAM	**Storage**
PRT	688 KB
PMT	3.28 MB
Total in-DRAM Overhead	**3.95 MB**

6.9 Results

6.9.1 Methodology

We model a quad core processor and a 2.5 GB main memory composed of a 2 GB ReRAM and a 512MB DRAM using Gem5 full-system simulator [19] integrated with Ramulator [35], a cycle accurate main memory simulator. We extend Ramulator to support flat address space hybrid memories. The latency and energy values of main memory and LLC are generated using CACTI [20] and DESTINY [61]. Note that the ReRAM cell parameters used in DESTINY are based on the projections by [32]. The system configuration of our experiments is shown in TABLE XIII. The benchmarks used in this study are chosen from SPEC2006 as shown in TABLE XIV. For all workloads, we use either sampled reference or native input sets to represent a real-world execution scenario and run the applications for 0.5 billion instructions, after a 1.5 billion instructions warm-up phase. To generate the information required for the LSTM-based address predictor including the miss causing instructions' PCs and LLC miss addresses, we capture the memory trace of applications using the Exec debug flag in Gem5. We also use the generated dynamic traces for our evaluations.

TABLE XIII. System configuration.

Processor	4-core, 2.5 GHz, out-of-order	
L1 Cache	Split I/D, Private, 32KB per core, 4-way, LRU, 2-cycle access latency	
L2 Cache	256KB, 8-way, 8-cycle access latency	
L3 Cache	4MB Shared, 16-way, 31-cycle access latency	
Memory Controller	One memory controller per channels Read Queue: 64 entries Write Queue: 64 entries, Write drain threshold: α (high) = 80%, β(low)=50%.	
Main Memory	DRAM	512MB, 1GHz: DDR- 4 channels, 1 rank, 8 banks, 1K rows, 64-bit bus width
		t_{RAS}= 28 cycles, t_{RCD}= t_{CAS} = t_{RP} = 11 cycles, t_{WR} = 12 cycles
	ReRAM	2GB, 2channels, 1 rank, 8 banks, 8K rows, 64-bit bus width
		t_{RCD}= 18 cycles, t_{CAS}= 11 cycles

TABLE XIV. Workloads.

Workloads: from the SPEC CPU2006 benchmarks	
mcf×4 leslie3D×4	milc×4 lbm×4
mix1: leslie3D×2, omnetpp×2	mix2: libquantum×2, mcf×2
mix3: bwaves-soplex-omnetpp-libq.	mix4: milc-bwaves-soplex-lbm

We compare our scheme with two state-of-the-art hybrid main memory systems: PoM [75] and MemPod [67].

1) PoM: we configure PoM based on the specifications in [75] and adjust some of the parameters based on our configuration. Based on our memory timing model, we set the value of *K* to 8. For the SRC, which is the equivalent of our PRTc, we use a 32KB cache similar to *DynaSwap*.

2) MemPod: for the MEA algorithm used in MemPod for its swap decisions, we use 64 MEA counters and 50μs MEA intervals similar to the original paper. We also use a 32KB cache for the remapping table.

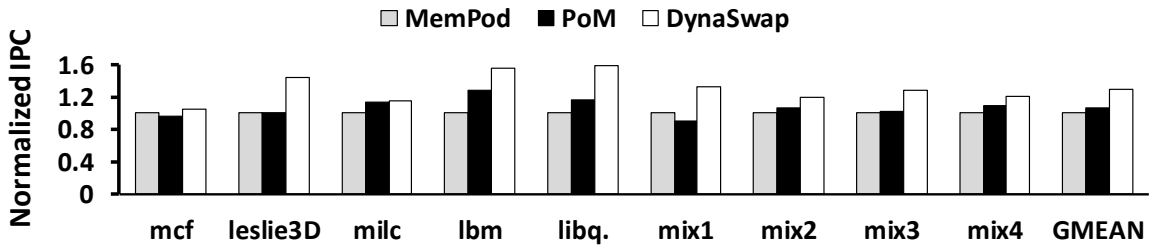


Figure 40. IPC of the evaluated schemes.

3) DynaSwap: the configuration details of our scheme are described in TABLE XII and TABLE XIII. Both PMT and PRT are in DRAM. PMTc and PRTc are in the memory controller. In our design, we use a 64KB, 8-way set associative PMTc and a 32KB, 4-way set associative PRTc.

6.9.2 Performance Evaluations

Figure 40 compares the IPC values of the evaluated schemes. The results are normalized to MemPod. Results reveal that *DynaSwap* can outperform PoM and MemPod by 23.6% and 30.1% on average, respectively.

The performance benefits of our proposed scheme come from two main factors. First, *DynaSwap* accurately prefetches a page from ReRAM into DRAM if it is likely to receive a large number of accesses in the near future (i.e., its FA flag is '1'). It also performs majority of PRTc accesses off the critical path of program execution thanks to the LSTM-Based address predictor high accuracy (i.e., 82.4% on average). It should be noted that in case of a PRTc miss, the latency of accessing PRT in DRAM is not negligible. Hence, the sooner we handle a PRTc miss, the better. Figure 41 shows the accuracy of our page swap scheme. We consider a swap accurate if the number of accesses to the swapped page in DRAM is high enough to justify the swap cost; if the swapped

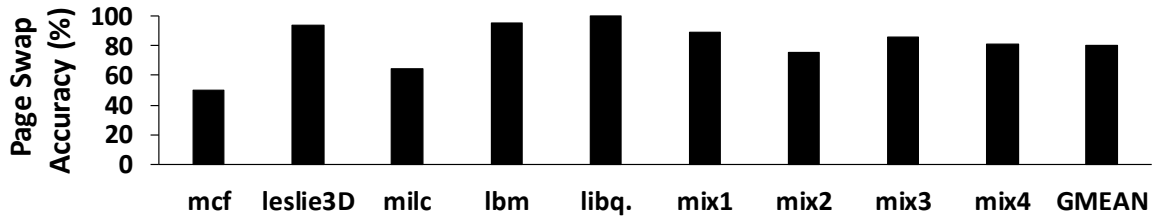


Figure 41. *DynaSwap* page swap accuracy.

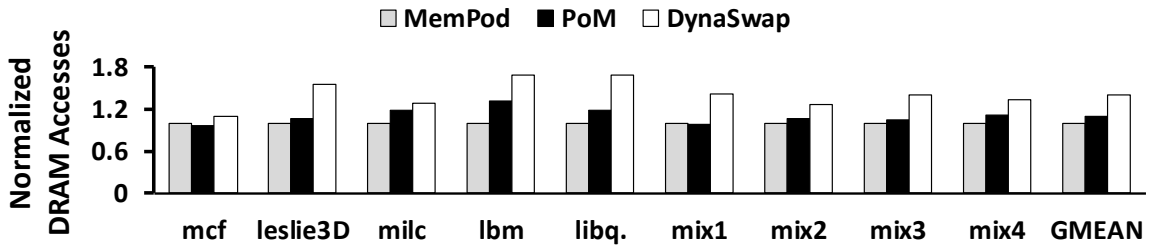


Figure 42. Normalized number of DRAM accesses.

page receives at least 12 accesses in DRAM, we recognize the swap as accurate. As shown in the figure, *DynaSwap* is accurate in vast majority of applications with an average swap accuracy of 80.1%.

Second, our scheme dynamically increases the associativity of a swap group to accommodate more highly accessed pages in DRAM. Figure 42 shows the fraction of main memory accesses serviced in DRAM by the evaluated schemes. The results are normalized to MemPod. Results reveal that *DynaSwap* can increase the number of accesses serviced in DRAM by 29.8% and 40.2% on average, compared to PoM and MemPod, respectively. In addition, while serving majority of the requests in DRAM (i.e., 85.7% on average), *DynaSwap* also performs fewer swaps compared to PoM and MemPod. It is also worth noting that similar to [37], to avoid saturating

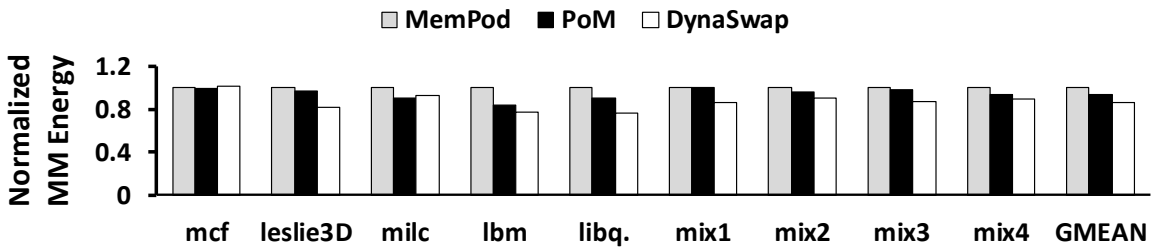


Figure 43. Normalized main memory energy of the evaluated schemes.

DRAM channels and not using the NVM channels, we use a simple heuristic that declines the swaps if 95% of the main memory requests of an application have been satisfied by DRAM.

6.9.3 Energy Evaluations

Figure 43 shows the total hybrid main memory energy consumption for the evaluated schemes. The results are normalized to MemPod. Results show that *DynaSwap* can reduce the energy consumption of the hybrid main memory by 7.6% and 13.5% on average, compared to PoM and MemPod, respectively. Generally, by moving the pages that are likely to receive a large number of accesses into DRAM and also increasing the associativity of swap groups dynamically, our design increases the number of accesses serviced in DRAM by 29.8% and 40.2% on average, compared to PoM and MemPod (see Figure 42). Hence, it reduces the number of accesses serviced in NVM, which has much higher dynamic energy compared to DRAM. More specifically, as we mentioned before, the page swap accuracy of *DynaSwap* is high enough (80.1%, on average, as shown in Figure 41) to justify the costs of swaps. In other words, a page is prefetched into DRAM if it is likely to receive a large number of accesses in the near future, which can be determined based on the access frequency of page in the past. On the other hand, to accommodate more such

pages in DRAM, which has much lower access energy compared to NVM, *DynaSwap* increases the associativity of swap groups dynamically.

6.10 Conclusion

In this chapter, we proposed *DynaSwap*, a scheme that improves system performance and energy efficiency by efficiently utilizing DRAM space in a flat address space hybrid DRAM/NVM main memory.

DynaSwap dynamically associates swap groups with each other in such a way that a swap group with many high frequently accessed pages can benefit from the DRAM space of a swap group with low frequently accessed pages. To do so, it classifies the swap groups into two categories; SAT and NON-SAT. A swap group is recognized as SAT if its number of high frequently accessed pages is larger than the swap groups associativity. Otherwise, the swap group is recognized as NON-SAT. Since a SAT swap group requires more DRAM segments than its baseline associativity, we double the associativity of a SAT swap group by associating a NON-SAT swap group as partner to it. Our experimental results revealed that *DynaSwap* can efficiently utilize DRAM capacity and improve the overall performance and main memory energy efficiency by 30.1% and 13.5% on average, respectively, compared to a state-of-art baseline design.

CHAPTER 7

CONCLUSION

DRAM can no longer satisfy the memory capacity demands of the modern-day applications due to its scalability limit and considerable amount of static and refresh power consumption. Non-Volatile Memory (NVM) technologies such as Phase Change Memory (PCM) have recently emerged as promising alternatives to DRAM. Compared to DRAM, NVMs have better scalability, higher density and zero standby power. However, NVMs generally suffer from higher access latency and energy (especially for the write operations) and limited write endurance. To benefit from the large capacity of NVM and the lower access latency and energy of DRAM, hybrid DRAM/NVM main memories, which incorporate both DRAM and NVM, have been proposed. In this thesis, we presented novel schemes for improving the energy efficiency of hybrid main memories or alleviating the write-related overheads of PCM-based memories. We first focused on reducing DRAM refresh and background power in a hybrid DRAM/NVM main memory by proposing two schemes called *Refree* and *NEMO*. Then, we presented *WALL*, a scheme that improves the energy efficiency and lifetime of a PCM-based main memory by reducing the number of writebacks from the LLC to PCM. Finally, we presented a scheme called *DynaSwap* to efficiently utilize DRAM space in a flat address space hybrid main memory and improve its performance and energy efficiency.

The *Refree* scheme eliminates refresh operations of the DRAM cache in a hybrid DRAM/PCM main memory to improve system performance and energy efficiency. *Refree* takes all the refresh-reducing factors including rows access pattern and retention time into consideration. Specifically, a row that is accessed at least once within its retention time does not need to be refreshed. On the

other hand, most of the rows in a DRAM device are strong and have very long retention times. Hence, a row that is not accessed within such long retention time can be recognized as not frequently accessed or dead and does not need to be refreshed and kept in the DRAM cache either. *Refree* then evicts an inactive row from the DRAM cache instead of refreshing it and writes it back to PCM if the row is dirty. The experimental results revealed that *Refree* can effectively reduce the main memory power consumption with small performance impact. The effectiveness of *Refree* would further improve for future systems with larger DRAM sizes.

The *NEMO* scheme minimizes the background energy of hybrid main memories used in mobile devices. Specifically, *NEMO* takes advantage of the unique usage pattern of mobile devices, which are idle most of the times. During the long idle periods, *NEMO* evicts the nonvaluable memory pages (those that are less likely to be reused in future) from the DRAM cache and collects the remaining hot memory pages in a single DRAM rank, called the hot rank. It then powers off all the DRAM ranks except for the hot rank. In addition, to minimize the background power during the active periods, it predicts the number of DRAM ranks that needs to be powered up in addition to the hot rank based on the applications launching pattern in the past. The experimental results revealed that *NEMO* could effectively reduce the memory power consumption without negative performance impact.

The *WALL* scheme reduces the number of LLC writebacks to a PCM based main memory to improve its energy efficiency and lifetime. In that work, we first investigated the non-uniformity of the LLC sets' writebacks and proposed a writeback-aware set-balancing mechanism based on that. To implement the set-balancing mechanism, we first proposed a simple partner assignment strategy. Then, to further optimize our proposed *WALL* scheme, we proposed three novel partner assignment strategies, called contraction, expansion, and ConExp to pair sets with different behaviours more efficiently compared to the simple partner assignment strategy that pairs writer

and non-writer sets randomly. In addition, we proposed a simple but effective writeback-aware replacement policy to keep the frequently reused dirty lines of the sets in the cache. Our evaluation results revealed that *WALL* can achieve a significant reduction in the total number of writebacks to PCM; thereby improving the system performance, energy efficiency and PCM lifetime.

The *DynaSwap* scheme tries to efficiently utilize DRAM space in a flat address space hybrid DRAM/NVM main memory to improve system performance and energy efficiency. Specifically, *DynaSwap* dynamically associates swap groups with each other in such a way that a swap group with many high frequently accessed pages can benefit from the DRAM space of a swap group with low frequently accessed pages. To do so, it classifies the swap groups into two categories; SAT and NON-SAT. A swap group is recognized as SAT if its number of high frequently accessed pages is larger than the swap group's associativity. Otherwise, the swap group is recognized as NON-SAT. Since a SAT swap group requires more DRAM segments than its baseline associativity, we double the associativity of a SAT swap group by associating a NON-SAT swap group as partner to it. Our experimental results revealed that *DynaSwap* can efficiently utilize the DRAM capacity and improve the overall system performance and main memory energy efficiency.

APPENDIX

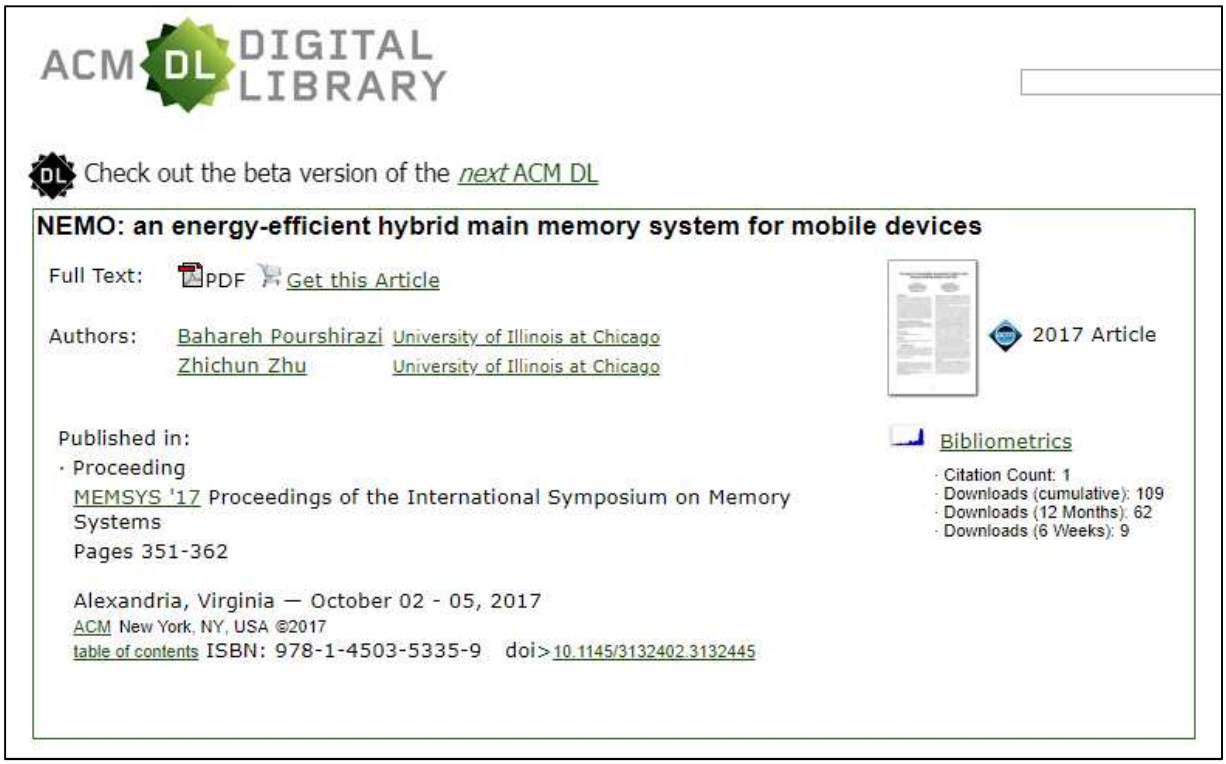
COPYRIGHT PERMISSIONS

The contents of Chapter 3 have been published in IEEE International Parallel and Distributed Processing Symposium (IPDPS2016) [66].

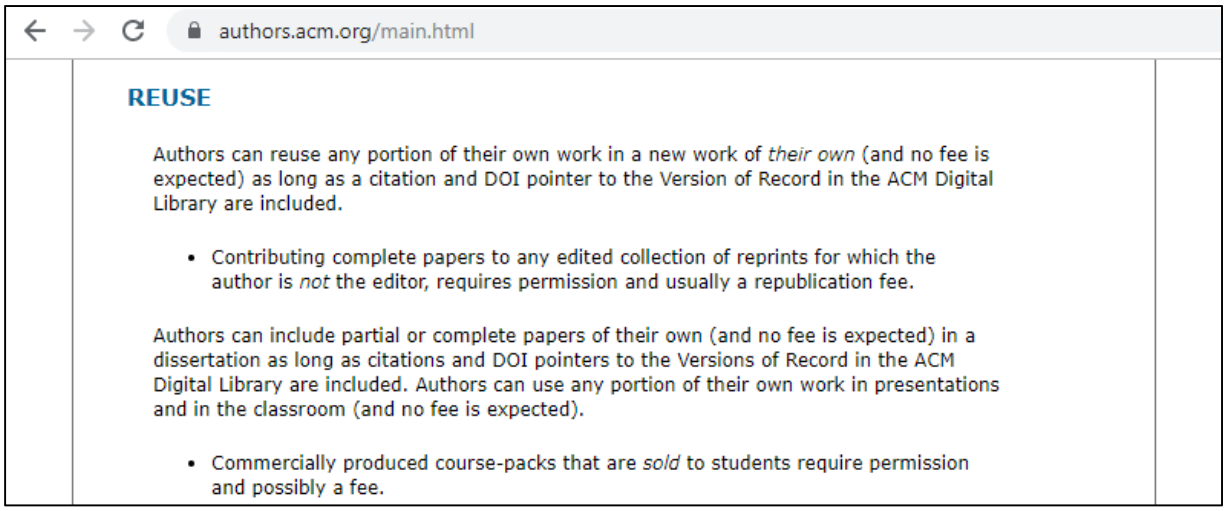
The contents of Chapter 4 have been published in The International Symposium on Memory Systems (MEMSYS2017) [65].

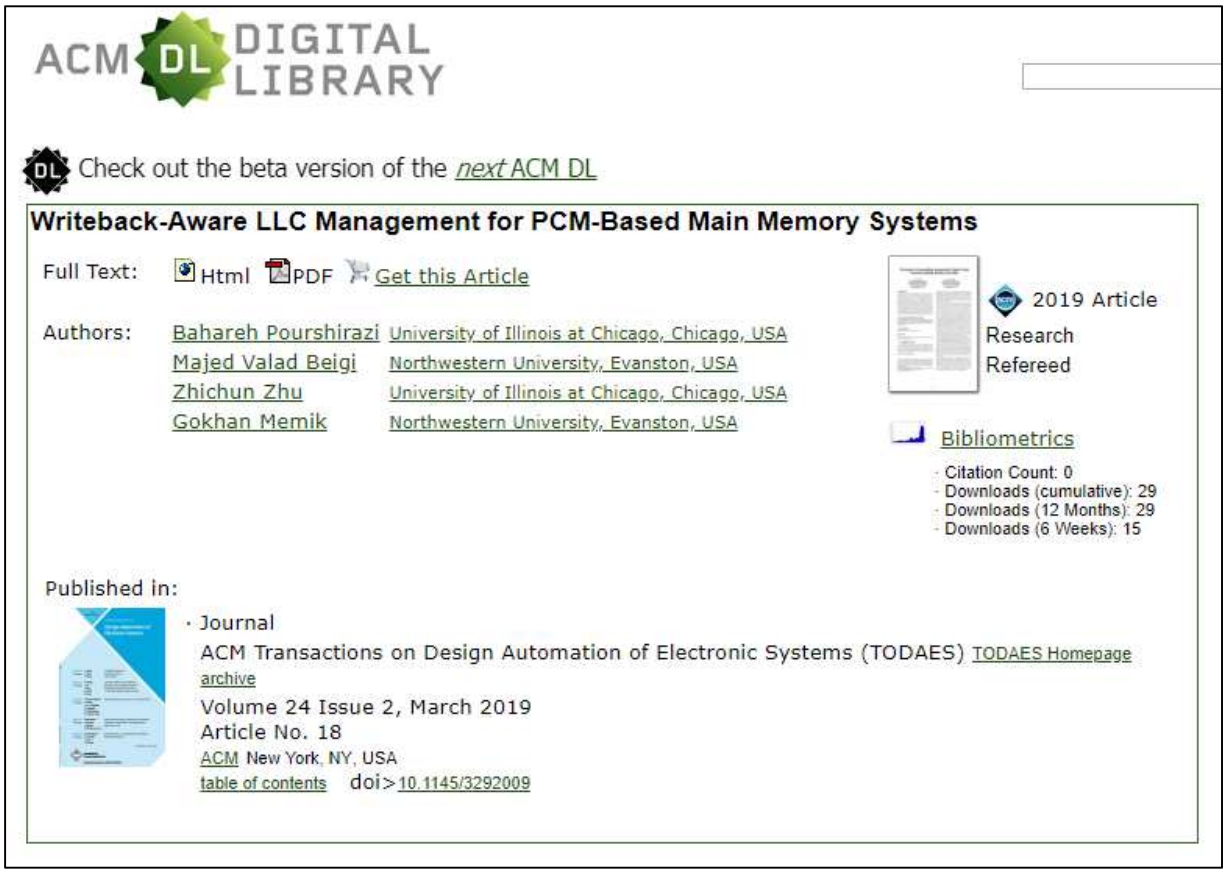
The contents of Chapter 5 have been published in Design, Automation and Test in Europe Conference (DATE2018) [63] and in ACM Transactions on Design Automation of Electronic Systems (TODAES2019) [64].

The thesis/dissertation reuse permission of the publishers of the papers mentioned above are presented in order in the following.

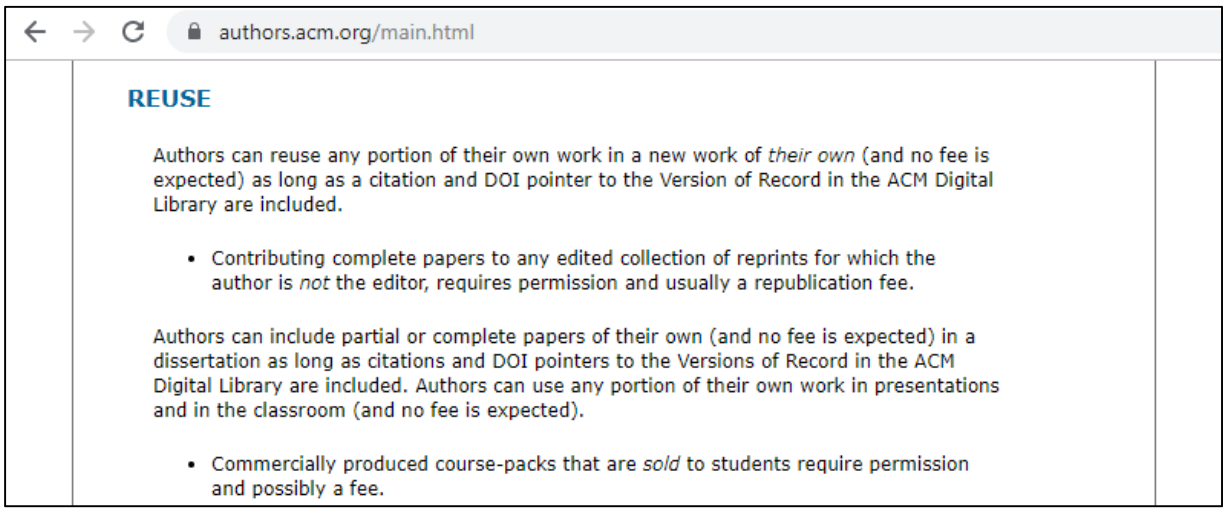


The following statement regarding "*ACM Author Rights*" is posted on *ACM* official website (https://authors.acm.org/main.html).





The following statement regarding "*ACM Author Rights*" is posted on *ACM* official website (https://authors.acm.org/main.html).



CITED LITERATURE

[1] Agrawal, A., Jain, P., Ansari, A., and Torrellas, J.: Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. In Proceedings of 19th International Symposium on High Performance Computer Architecture (HPCA), pages 400 - 411, 2013.

[2] Arjomand, M., Kandemir, M. T., Sivasubramaniam, A., and Das, C. R.: Boosting Access Parallelism to PCM-Based Main Memory. In Proceedings of 43rd Annual International Symposium on Computer Architecture (ISCA), pages 695-706 2016.

[3] Beigi, M. V.: Thermal-aware Optimizations for Emerging Technologies in 3D-Stacked Chips. PhD Dissertation, Northwestern University, 2019.

[4] Beigi, M. V. and Memik, G.: TAPAS: Temperature-aware Adaptive Placement for 3D Stacked Hybrid Caches. In Proceedings of International Symposium on Memory Systems (MEMSYS), pages 415-426, 2016.

[5] Bhati, I., Chishti, Z., Lu, S. L., and Jacob, B.: Flexible Auto-Refresh: Enabling Scalable and Energy-Efficient DRAM Refresh Reductions. In Proceedings of 42nd Annual International Symposium on Computer Architecture (ISCA), pages 235 - 246, 2015.

[6] Bienia, C., Kumar, S., Singh, J. P., and Li, K.: The PARSEC benchmark suite: Characterization and architectural implications. In Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 72-81 2009.

[7] Bock, S., Childers, B. R., Melhem, R., and Mossé, D.: Concurrent page migration for mobile systems with OS-managed hybrid memory. In Proceedings of the 11th ACM Conference on Computing Frontiers (CF), 2014.

[8] CACTI-6.5. Available: http: //hpl .hp.com: research /cacti/

[9] Carroll, A. and Heiser, G.: An Analysis of Power Consumption in a Smartphone. In Proceedings of USENIX annual technical conference, 2010.

[10] Chang, M.-T., Rosenfeld, P., Lu, S.-L., and Jacob, B.: Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM. In Proceedings of 19th International Symposium on High Performance Computer Architecture (HPCA), pages 143 - 154, 2013.

[11] Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., and Xie, Y.: PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In Proceedings of 43rd Annual International Symposium on Computer Architecture (ISCA), pages 27-39, 2016.

[12] Choi, Y., Song, I., Park, M.-H., Chung, H., Chang, S., and Cho, B.: A 20nm 1.8v 8gb pram with 40mb/s program bandwidth. In Proceedings of International Solid-State Circuits Conference (ISSCC), pages 46 - 48, 2012.

[13] Chou, C., Nair, P., and Qureshi, M. K.: Reducing Refresh Power in Mobile Devices with Morphable ECC. In Proceedings of International Conference on Dependable Systems and Networks (DSN), pages 355 - 366, 2015.

[14] Chou, C. C., Jaleel, A., and Qureshi, M. K.: CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In Proceedings of 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 1-12, 2014.

CITED LITERATURE (Continued)

[15] Deng, Z., Zhang, L., Mishra, N., Hoffmann, H., and Chong, F. T.: Memory cocktail therapy: a general learning-based framework to optimize dynamic tradeoffs in NVMs. In Proceedings of 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 232-244, 2017.

[16] Dong, X., Xu, C., Xie, Y., and Jouppi, N. P.: Nvsim: A circuitlevel performance, energy, and area model for emerging nonvolatile memory. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 31: 994 - 1007, 2012.

[17] Duan, R., Bi, M., and Gniady, C.: Exploring memory energy optimizations in smartphones. In Proceedings of International Green Computing Conference and Workshops (IGCC), pages 1-8, 2011.

[18] Franey, S. and Lipasti, M.: Tag Tables. In Proceedings of 21st International Symposium on High Performance Computer Architecture (HPCA), pages 514 - 525, 2015.

[19] gem5-simulator, Available: http: //gem5.org/

[20] Ghosh, M. and Lee, H.-H. S.: Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In Proceedings of 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 134 - 145, 2007.

[21] Huang, H., Shin, K. G., Lefurgy, C., Rajamani, K., Keller, T., Hensbergen, E. V., and Rawson, F.: Cooperative Software–Hardware Power Management for Main Memory. In Proceedings of Workshop on Power-Aware Computer Systems (PACS), 2004.

[22] Ham, T. J., Chelepalli, B. K., Xue, N., and Lee, B. C.: Disintegrated control for energy-efficient and heterogeneous memory systems. In Proceedings of 19th International Symposium on High Performance Computer Architecture (HPCA), pages 424 - 435, 2013.

[23] Hamamoto, T., Sugiura, S., and Sawada, S.: On the retention time distribution of dynamic random access memory (DRAM). IEEE Transactions on Electrical Devices,1300 - 1309, 1998.

[24] Huang, H., Pillai, P., and Shin, K. G.: Design and implementation of power-aware virtual memory. In Proceedings of Annual conference on USENIX Annual Technical Conference (ATEC), 2003.

[25] Huang, Y., Zha, Z., Chen, M., and Zhang, L.: Moby: A mobile benchmark suite for architectural simulators. In Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 45-54, 2014.

[26] Isen, C. and John, L.: ESKIMO: Energy savings using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM subsystem. In Proceedings of 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 337 - 346, 2009.

[27] ITRS: International technology roadmap for semiconductors. 2013.

[28] JEDEC: DDR3 SDRAM Specification. 2010.

[29] JEDEC: DDR4 STANDARD. 2012.

[30] Karlson, A. K., Meyers, B. R., Jacobs, A., Johns, P., and Kane, S. K.: Working Overtime: Patterns of Smartphone and PC Usage in the Day of an Information Worker. In Proceedings of International Conference on Pervasive Computing, pages 398-405, 2009.

[31] Karp, R. M., Shenker, S., and Papadimitriou, C. H.: A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems (TODS), 28: 51-55, 2003.

[32] Kawahara, A., Azuma, R., Ikeda, Y., Kawai, K., Katoh, Y., Tanabe, K., Nakamura, T., Sumimoto, Y., Yamada, N., Nakai, N., Sakamoto, S., Hayakawa, Y., Tsuji, K., Yoneda, S., Himeno, A., Origasa, K.-i., Shimakawa, K., Takagi, T., Mikawa, T., and Aono, K.: An 8Mb multi-layered cross-point ReRAM macro with 443MB/s write throughput. In Proceedings of International Solid-State Circuits Conference (ISSCC), 2012.

[33] Kim, K. and Lee, J.: A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs. IEEE Electron Devices Society, 45: 846 - 848, 2009.

CITED LITERATURE (Continued)

[34] Kim, Y., Imani, M., Patil, S., and Rosing, T. S.: CAUSE: Critical application usage-aware memory system using non-volatile memory for mobile devices. In Proceedings of International Conference on Computer-Aided Design (ICCAD), pages 690-696, 2015.

[35] Kim, Y., Yang, W., and Mutlu, O.: Ramulator: A Fast and Extensible DRAM Simulator. Computer Architecture Letters (CAL): 15:45-49, 2015.

[36] Knyaginin, D., Papaefstathiou, V., and Stenstrom, P.: ProFess: A Probabilistic Hybrid Main Memory Management Framework for High Performance and Fairness. In Proceedings of 24th International Symposium on High Performance Computer Architecture (HPCA), pages 143-155, 2018.

[37] Kokolis, A., Skarlatos, D., and Torrellas, J.: PageSeer: Using Page Walks to Trigger Page Swaps in Hybrid Memory Systems. In Proceedings of 25th International Symposium on High Performance Computer Architecture (HPCA), pages 596-608, 2019.

[38] Kultursay, E., Kandemir, M., Sivasubramaniam, A., and Mutlu, O.: Evaluating STT-RAM as an energy-efficient main memory alternative. In Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 256 - 267, 2013.

[39] Lee, B. C., Ipek, E., Mutlu, O., and Burger, D.: Architecting phase change memory as a scalable DRAM alternative. In Proceedings of 36th annual international symposium on Computer architecture (ISCA), pages 2-13, 2009.

[40] Lee, B. C., Zhou, P., Yang, J., Zhang, Y., Zhao, B., Ipek, E., Mutlu, O., and Burger, D.: Phase-change technology and the future of main memory. In Proceedings of IEEE MICRO, pages 131-141, 2010.

[41] Lee, H. G., Baek, S., Nicopoulos, C., and Kim, J.: An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems. In Proceedings of 29th International Conference on Computer Design (ICCD), pages 381 - 387, 2011.

[42] Lee, M., Seo, E., Lee, J., and Kim, J.-s.: PABC: Power-Aware Buffer Cache Management for Low Power Consumption. IEEE Transactions on Computers 56: 488 - 501, 2007.

[43] Lee, S., Bahn, H., and Noh, S. H.: Characterizing Memory Write References for Efficient Management of Hybrid PCM and DRAM Memory. In Proceedings of 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 25-27, 2011.

[44] Lee, Y., Kim, J., Jang, H., Yang, H., Kim, J., Jeong, J., and Lee, J. W.: A fully associative, tagless DRAM cache. In Proceedings of 42nd Annual International Symposium on Computer Architecture (ISCA), pages 211-222 2015.

[45] Lim, K., Ranganathan, P., Chang, J., Patel, C., Mudge, T., and Reinhardt, S.: Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In Proceedings of International Symposium on Computer Architecture (ISCA), pages 315-326, 2008.

[46] Liu, J., Jaiyen, B., Veras, R., and Mutlu, O.: RAIDR: Retention-Aware Intelligent DRAM Refresh. In Proceedings of 39th Annual International Symposium on Computer Architecture (ISCA), pages 1-12, 2012.

[47] Liu, S., Pattabiraman, K., Moscibroda, T., and Zorn, B. G.: Flikker: Saving DRAM Refresh-power through Critical Data Partitioning through Critical Data Partitioning. In Proceedings of Sixteenth International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 213-224 2011.

[48] Loh, G. H. and Hill, M. D.: Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In Proceedings of 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 454-464, 2011.

[49] Lu, Y., Shanghai, C., Wu, D., He, B., Tang, X., Xu, J., and Guo, M.: Rank-Aware Dynamic Migrations and Adaptive Demotions for DRAM Power Management. IEEE Transactions on Computers, 65: 187-202, 2015.

CITED LITERATURE (Continued)

[50] Mandelman, J. A., Dennard, R. H., Bronner, G. B., DeBrosse, J. K., Divakaruni, R., Li, Y., and Radens, C. J.: Challenges and future directions for the scaling of dynamic random-access memory (dram). IBM Journal of Research and Development, 46: 187-212, 2002.

[51] Meza, J., Chang, J., Yoon, H., Mutlu, O., and Ranganathan, P.: Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. IEEE Computer Architecture Letters, 11: 61-64, 2012.

[52] Micron-Technology: 4Gb: x4, x8, x16 DDR3 SDRAM. 2011.

[53] Micron-Technology: 128Mb: x16, x32 Mobile LPDDR SDRAM. 2007.

[54] Micron-Technology: Calculating Memory System Power for DDR3. 2007.

[55] Micron-Technology: EDB1316BD Datasheet.2016.

[56] Micron-Technology: LPDDR2 System Power Calculator. 2013.

[57] Micron-Technology: Various methods of DRAM refresh. 1999.

[58] Moshnyaga, V. G., Vo, H., Reinman, G., and Potkonjak, M.: Reducing Energy of DRAM/Flash Memory System by OS-Controlled Data Refresh. In Proceedings of International Symposium on Circuits and Systems (ISCS), pages 2108 - 2111, 2007.

[59] Nair, P., Chou, C. C., and Qureshi, M. K.: Refresh pausing in DRAM memory systems. ACM Transactions on Architecture and Code Optimization (TACO), 11: 10:1-10:25, 2014.

[60] NAS. *The NAS parallel benchmarks*.

[61] Poremba, M., Mittal, S., Li, D., Vetter, J. S., and Xie, Y.: DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches. In Proceedings of Design, Automation & Test in Europe Conference (DATE), pages 1543-1546, 2015.

[62] Poremba, M., Zhang, T., and Xie, Y.: NVMain 2.0: Architectural Simulator to Model (Non-)Volatile Memory Systems. Computer Architecture Letters (CAL): 140 - 143, 2015.

[63] Pourshirazi, B., Beigi, M. V., Zhu, Z., and Memik, G.: WALL: A Writeback-Aware LLC Management for PCM-based Main Memory Systems. In Proceedings of Design, Automation & Test in Europe Conference (DATE), pages 449-454, 2018.

[64] Pourshirazi, B., Beigi, M. V., Zhu, Z., and Memik, G.: Writeback-Aware LLC Management for PCM-based Main Memory Systems. ACM Transactions on Design Automation of Electronic Systems (TODAES), 24: 18:1-18:19, 2019. DOI: 10.1145/3292009

[65] Pourshirazi, B. and Zhu, Z.: NEMO: An Energy-Efficient Hybrid Main Memory System for Mobile Devices. In Proceedings of the International Symposium on Memory Systems (MEMSYS), 2017. DOI: 10.1145/3132402.3132445

[66] Pourshirazi, B. and Zhu, Z.: Refree: A Refresh-Free Hybrid DRAM/PCM Main Memory System. In Proceedings of International Parallel and Distributed Processing Symposium (IPDPS), pages 566-575, 2016.

[67] Prodromou, A., Meswani, M., Jayasena, N., Loh, G., and Tullsen, D. M.: MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories. In Proceedings of 23th International Symposium on High Performance Computer Architecture (HPCA), pages 433-444, 2017.

[68] Qureshi, M. K., Franceschini, M. M., and Lastras-Montaño, L. A.: Improving read performance of Phase Change Memories via Write Cancellation and Write Pausing. In Proceedings of 16th International Symposium on High-Performance Computer Architecture (HPCA), 2010.

[69] Qureshi, M. K. and Loh, G. H.: Fundamental Latency Trade-offs in Architecting DRAM Caches. In Proceedings of 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 235-246 2012.

CITED LITERATURE (Continued)

[70] Qureshi, M. K., Srinivasan, V., and Rivers, J. A.: Scalable high performance main memory system using phase-change memory technology. In Proceedings of 36th annual international symposium on Computer architecture (ISCA), pages 24-33, 2009.

[71] Ramos, L. E., Gorbatov, E., and Bianchini, R.: Page placement in hybrid memory systems. In Proceedings of International conference on Supercomputing (ICS), pages 85-95 2011.

[72] Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y. C., Shelby, R. M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H. L., and Lam, C. H.: Phase-change random access memory: A scalable technolog. IBM Journal of Research and Development 52: 465 - 479, 2008.

[73] Rolán, D., Fraguela, B. B., and Doallo, R.: Adaptive line placement with the set balancing cache. In Proceedings of 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 529-540, 2009.

[74] Ryoo, J. H., Meswani, M. R., Prodromou, A., and John, L. K.: SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization. In Proceedings of 23th International Symposium on High Performance Computer Architecture (HPCA), pages 349-360, 2017.

[75] Sim, J., Alameldeen, A. R., Chishti, Z., Wilkerson, C., and Kim, H.: Transparent Hardware Management of Stacked DRAM as Part of Memory. In Proceedings of 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 13-24, 2014.

[76] Song, W., Kim, Y., Kim, H., Lim, J., and Kim, J.: Personalized optimization for android smartphones. ACM Transactions on Embedded Computing Systems (TECS), 13: 60:1-60:25, 2014.

[77] Strukov, D. B.: Endurance-write-speed tradeoffs in nonvolatile memories. Applied Physics A, 122: 1-4, 2016.

[78] Udipi, A. N., Muralimanohar, N., Chatterjee, N., Balasubramonian, R., Davis, A., and Jouppi, N. P.: Rethinking DRAM design and organization for energy-constrained multi-cores. In Proceedings of 37th annual international symposium on Computer architecture (ISCA), pages 175-186 2010.

[79] Wang, Z., Jiménez, D. A., Xu, C., Sun, G., and Xie, Y.: Adaptive Placement and Migration Policy for an STT-RAM-Based Hybrid Cache. In Proceedings of 20th International Symposium on High Performance Computer Architecture (HPCA), 2014.

[80] Wang, Z., Shan, S., Cao, T., Gu, J., Xu, Y., Mu, S., Xie, Y., and Jiménez, D. A.: WADE: Writeback-aware dynamic cache management for NVM-based main memory system. ACM Transactions on Architecture and Code Optimization (TACO), 10: 51:1-51:21, 2013.

[81] Wong, H.-S. P., Raoux, S., Kim, S., Liang, J., Reifenberg, J. P., Rajendran, B., Asheghi, M., and Goodson, K. E.: Phase Change Memory. IEEE 98: 2201 - 2227, 2010.

[82] Xia, F., Jiang, D., Xiong, J., Chen, M., Zhang, L., and Sun, N.: DWC: dynamic write consolidation for phase change memory systems. In Proceedings of 28th ACM international conference on Supercomputing (ICS), pages 211 - 220, 2014.

[83] Xu, C., Niu, D., Muralimanohar, N., Balasubramonian, R., Zhang, T., Yu, S., and Xie, Y.: Overcoming the Challenges of Crossbar Resistive Memory Architectures. In Proceedings of 21th International Symposium on High Performance Computer Architecture (HPCA), 2015.

[84] Yoon, H., Meza, J., Ausavarungnirun, R., Harding, R. A., and Mutlu, O.: Row Buffer Locality Aware Caching Policies for Hybrid Memories. In Proceedings of 30th International Conference on Computer Design (ICCD), pages 337-344 2013.

[85] Zhang, L., Neely, B., Franklin, D., Strukov, D., Xie, Y., and Chong, F. T.: Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs. In Proceedings of 43rd Annual International Symposium on Computer Architecture (ISCA), pages 519-531, 2016.

[86] Zhang, M., Zhang, L., Jiang, L., Liu, Z., and Chong, F. T.: Balancing Performance and Lifetime of MLC PCM by Using a Region Retention Monitor. In Proceedings of 23th International Symposium on High Performance Computer Architecture (HPCA), pages 385-396, 2017.

[87] Zhang, W. and Li, T.: Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 101 - 112, 2009.

[88] Zhong, K., Liu, D., Liang, L., Zhu, X., Long, L., Wang, Y., and Sha, E.: Energy-Efficient In-memory Paging for Smartphones. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 99: 1577 - 1590, 2015.

[89] Zhong, K., Zhu, X., Wang, T., Zhang, D., Luo, X., Liu, D., Liu, W., and Sha, E. H.-M.: DR. Swap: energy-efficient paging for smartphones. In Proceedings of International symposium on Low power electronics and design (ISLPED), 2014.

[90] Zhou, P., Zhao, B., Yang, J., and Zhang, Y.: A durable and energy efficient main memory using phase change memory technology. In Proceedings of 36th annual international symposium on Computer architecture (ISCA), pages 14-23 2009.

[91] Zhou, P., Zhao, B., Yang, J., and Zhang, Y.: Throughput Enhancement for Phase Change Memories. IEEE Transactions on Computers (TC), 63: 2080 - 2093, 2014.

[92] Zhou, Y., Philbin, J., and Li, K.: The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In Proceedings of The General Track: 2001 USENIX Annual Technical Conference (ATEC), pages 91-104, 2001.

VITA

Bahareh Pourshirazi
Web: https://www.linkedin.com/in/bahareh-pourshirazi-7871999a
Email: bahareh.pourshirazi@gmail.com

EDUCATION

University of Illinois at Chicago — Chicago, IL
Ph.D. in Computer Engineering, GPA: 4.0/4.0 — 2013–2019

Shahid Beheshti University — Tehran, Iran
M.S. in Computer Engineering, GPA: 18.84/20 — 2010-2013
B.S. in Computer Engineering, GPA: 17.04/20 — 2006-2010

PUBLICATIONS Conference (C), Journal (J)

J3 **Pourshirazi, B.**, Beigi, M. V., Zhu, Z., and Memik, G.: Writeback-Aware LLC Management for PCM-based Main Memory Systems. ACM Transactions on Design Automation of Electronics Systems (TODAES), 2019.

C6 **Pourshirazi, B.**, Beigi, M. V., Zhu, Z., and Memik, G.: WALL: A Writeback-Aware LLC Management for PCM-based Main Memory Systems. In Proceedings of IEEE/ACM Design, Automation, and Test in Europe Conference (DATE), Dresden, Germany, 2018.

C5 **Pourshirazi, B.**, and Zhichun Zhu: NEMO: An Energy-Efficient Hybrid Main Memory System for Mobile Devices. In ACM International Symposium on Memory Systems (MEMSYS), Washington, DC, 2017.

C4 **Pourshirazi, B.**, and Zhu, Z.: Refree: A Refresh-Free Hybrid DRAM/PCM Main Memory System. In Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, 2016.

J2 Beigi, M. V., Safaei, F., and **Pourshirazi, B.**: Application-aware virtual paths insertion for NoCs. Elsevier Journal of Microelectronics (Elsevier-Micro), vol. 45, issue.4, pages. 454-462, 2014.

C3 Beigi, M. V., Safaei, F., Belghadr, A., and **Pourshirazi B.**: A Dependable and Power Efficient NoC Architecture. In Proceeding of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Brazil, 2013.

C2 Beigi, M. V., Safaei, F., and **Pourshirazi, B**.: An Energy-Efficient Reconfigurable NoC Architecture with RF-Interconnects. In Proceedings of EuroMicro conference on Digital System Design (DSD), Spain, 2013.

J1 Beigi, M. V., Safaei, F., and **Pourshirazi, B.**: DBR: A Simple, Fast and Efficient Dynamic

Network Reconfiguration Based on Deadlock Recovery Scheme. International Journal of VLSI design and Communication Systems (VLSICS), vol.3, no.5, 2012.

C1 **Pourshirazi, B.** and Jahanian, A.: RF-Interconnect resource assignment and placement algorithms in application specific ICs to improve performance and reduce routing congestion. In Proceedings of EuroMicro conference on Digital System Design (DSD), Turkey, 2012.

AWARDS AND HONORS

Wexler award given to outstanding entry-level students	2013
Ontario trillium scholarship given to the best doctoral students from around the world to study in Ontario	2013
Ranked third in M.S. program among all graduate students of computer engineering, Shahid Beheshti University, Tehran, Iran	2013
Memocode World Design Contest **fourth place** in the world	2012
Gained the privilege to start M.S. in Shahid Beheshti University as a talented student (**ranked second** with GPA 17.04/20 in B.S.)	2011
Qualified for Robocup World Championship 2008, China	2008
Iran open 2008 technical committee member (Virtual Robots League)	2008
Robocup Competitions **third place** in Iran Open 2008 (Virtual Robots League)	2008

INDUSTRY EXPERIENCES

AMD Research, Santa Clara, CA
Research Co-op — Jan. 2019 – May 2019
Project: Developing and implementing a technique to reduce data movement between GPU and CPU for deep neural networks using Keras backend with TensorFlow framework

Institution for Research in Fundamental Sciences (IPM), Tehran, Iran
Research Intern — Summer 2009