THE DESIGN OF A HIGH CAPACITY AND ENERGY EFFICIENT PHASE CHANGE MAIN MEMORY

by

Alexandre Peixoto Ferreira

BS, Universidade de Brasília, 1988

MS, Universidade Federal de Santa Catarina, 1998

Submitted to the Graduate Faculty of

the Department of Computer Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2011

UNIVERSITY OF PITTSBURGH

COMPUTER SCIENCE DEPARTMENT

This dissertation was presented

by

Alexandre Peixoto Ferreira

It was defended on

March 24, 2011

and approved by

Daniel Mossé, Department of Computer Science

Bruce Childers, Department of Computer Science

Rami Melhem, Department of Computer Science

Mazin Yousif, T-Systems

Dissertation Advisors: Daniel Mossé, Department of Computer Science,

Bruce Childers, Department of Computer Science

THE DESIGN OF A HIGH CAPACITY AND ENERGY EFFICIENT PHASE CHANGE MAIN MEMORY

Alexandre Peixoto Ferreira, PhD

University of Pittsburgh, 2011

Higher energy-efficiency has become essential in servers for a variety of reasons that range from heavy power and thermal constraints, environmental issues and financial savings. With main memory responsible for at least 30% of the energy consumed by a server, a low power main memory is fundamental to achieving this energy efficiency. DRAM has been the technology of choice for main memory for the last three decades primarily because it traditionally combined relatively low power, high performance, low cost and high density. However, with DRAM nearing its density limit, alternative low-power memory technologies, such as Phase-change memory (PCM), have become a feasible replacement. PCM limitations, such as limited endurance and low write performance, preclude simple drop-in replacement and require new architectures and algorithms to be developed.

A PCM main memory architecture (PMMA) is introduced in this dissertation, utilizing both DRAM and PCM, to create an energy-efficient main memory that is able to replace a DRAM-only memory. PMMA utilizes a number of techniques and architectural changes to achieve a level of performance that is par with DRAM. PMMA achieves gains in energy-delay of up to 65%, with less than 5% of performance loss and extremely high energy gains. To address the other major shortcoming of PCM, namely limited endurance, a novel, low-overhead wear-leveling algorithm that builds on PMMA is proposed that increases the lifetime of PMMA to match the expected server lifetime so that both server and memory subsystems become obsolete at about the same time.

We also study how to better use the excess capacity, traditionally available on PCM devices, to obtain the highest lifetime possible. We show that under specific endurance distributions, the naive choice does not achieve the highest lifetime. We devise rules that empower the designer to select algorithms and parameters to achieve higher lifetime or simplify the design knowing the impact on the lifetime. The techniques presented also apply to other storage class memories (SCM) memories that suffer from limited endurance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

PREFACE

I would like to dedicate this work to my wife for displaying the patience and dedication that allowed this work to be in existence.

I would also acknowledge the dedication, encouragement and all the extremely helpful advice dispensed to me by my advisors during my stay at the university.

My peers at the university are a very important piece of the fabric that made my stay an extremely enjoyable and valuable experience. They were and still are a source of inspiration.

Last but not least, I would like to thank the staff of the Computer Science department for being helpful, friendly and professional in every situation. They always went above and beyond to help and I appreciate and recognize their efforts. I wish to them all the success and recognition they deserve.

1.0 INTRODUCTION

The design of main memory has become the newest challenge for system designers due to aggressive requirements of capacity, power consumption, performance and form factor. Although main memory has steadily grown in size and performance, application demand has grown even faster, with multicore CPUs and virtualization compounding this pressure. In servers, the challenge is more pronounced since capacity and performance objectives for memory subsystems are much higher.

Main memory uses a significant portion of system power [5, 27] in servers, as shown in Figure 1. With the growth of memory demand, it has become especially challenging to construct a memory subsystem that has feasible power consumption, heat dissipation, and form factor. Main memory size is growing from the 10s to 100s of gigabytes of current systems to terabytes or more in the new designs [19, 9]. A terabyte of memory using today's main memory technology (e.g., Fully Buffered DRAM [32, 18, 15] or DDR3 SDRAM [33]) is above the cost, size and energy allowable for most systems. With commercially available memory devices based on 8 GBytes FBDRAM (fully buffered DRAM)[32], it would take 125 memory modules (DIMMs) to construct a one terabyte memory! Other memory technologies like DDR3-DRAM face even larger problems with density and form factor [33]. In terms of power consumption, the situation is even bleaker. The power consumption of just the memory chips for one terabyte memory implemented as FBDRAM is 1.25 KW [32, 32, 15]. Using DDR3-DRAM, the power consumption in the memory devices is smaller, 400W [33], but more system support is required since only a small number of DIMMs can be connected to each bus. These power demands are up to 10 times more than the demands of memory in current machines, which are already considered too power hungry [5, 3, 2, 1, 4, 28].

For over three decades, DRAM has been chosen as the technology for main memory because it allowed construction of cheaper, larger or faster main memories than existing alternative tech-

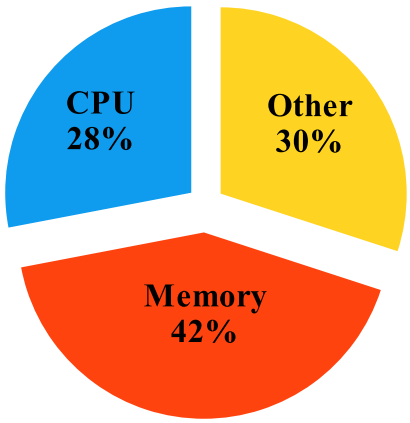

Figure 1: Server Power Distribution [27].

nologies such as SRAM, MRAM, Flash and others. The primary advantage of DRAM was an unmatched combination of density, energy-efficiency, speed and cost per bit. The DRAM storage mechanism, which stored the memory cell value as charge in a capacitor, is susceptible to a number of problems. The main limitations of DRAM technology are:

- Destructive reads: Each read removes the stored charge in the capacitor, requiring the information to be restored.
- Limited data retention: The stored charge is also lost by leakage currents. Current DRAM cells can maintain data for tens of milliseconds. Hence, a refresh process is required to create the appearance of unlimited storage time.
- Susceptibility of singe-event upsets (SEUs, or errors): The stored charge in each cell is small and the ionization process caused by cosmic radiation (alpha particles) can insert or remove enough charge to change stored value.

In order to achieve good reliability and compensate for DRAM susceptibility to singe-event upsets, error correction codes (ECC) are employed. In current systems, the addition of ECC increases memory size by 12.5% with the corresponding increase in energy and power consumption [31], assuming a common configuration of 8 ECC bits for each 64bits word. Due to form factor, power

consumption, and fault tolerance overhead (ECC), it is clearly very expensive to build scalable main memory systems with DRAM for a typical Chip MultiProcessor (CMP) system. Until recently, DRAM has been able to reap the benefits of smaller semiconductor technologies, with size being the primary gain, but it is expected to reach serious roadblocks for smaller geometries. As memory cells become smaller, less charge is stored in each cell, and they suffer more from charge loss. The combination of these two effects cause DRAM cells to be unusable below 22nm [13, 14].

Memory technologies that can be considered alternatives to DRAM are still struggling with either size, speed or maturity. First, NAND Flash is a mature technology and is a very serious contender for file storage since it allies non-volatility with density, but its use as main memory has serious limitations because of large latency (in the order of microseconds for reads and even more for writes), block oriented access (512 bytes or more) and limited endurance. Endurance is also touted as a primary scaling problem for NAND Flash, decreasing from 100K erase cycles to less than 3K erase cycles for 25nm or smaller geometries [13, 14]. Second, NOR Flash uses a similar storage mechanism as NAND Flash. NOR Flash advantages are in speed (in the hundreds of nanoseconds) and byte addressability, but it offers limited capacity and similar endurance when compared to NAND Flash. Lastly, resistive memories such as STT-RAM [10] are promising but still immature. Existing resistive devices [12, 10] (commercial and prototypes) are still orders of magnitude smaller in capacity than current DRAM devices, creating a large size and cost disadvantage.

Phase-Change Memory (PCM) has been proposed to replace DRAM in main memory [24, 48, 41]. PCM [38, 22, 23] is a new alternative memory technology that uses the physical state of the material and the state's impact on resistivity to store a bit, rather than relying on electrical charge. PCM works by changing the material from a crystalline to an amorphous state and vice-versa (hence, it is non volatile [11] and radiation resistant [30]). Each state has a specific electrical resistance, which requires small power consumption for a read operation. PCM's storage mechanism, being non-volatile, requires no power for idle mode and very low standby power for reading: a small current is used to read the cell by measuring the resistance. However, a high current is needed to change the cell. The cell state is modified by melting the phase-change material and using different cooling profiles to achieve the desired final state: crystalline with long cooling time and amorphous with a short cooling time. Hence, PCM has asymmetric timing and energy

for reads/writes. Writes are five to ten times slower than reads and consume substantially more energy due to the need to melt material in the cell to change its physical state. Also, because the cooling profile is different if the bit is 1 or 0, the write duration and energy used depends on the value written.

Contrary to NAND Flash memory and similar to NOR Flash, PCM is bit addressable. Other operations, such as erasing a block before writing it as in NAND and NOR Flash, are not required.

PCM suffers from limited endurance because a write to a cell reduces slowly the cell's ability to reliably achieve physical state changes [16, 21]. PCM prototypes show that a PCM cell supports at least 10^7 writes [26, 45], which is at least three orders of magnitude better than Flash. PCM prototypes have shown high density (up to 1Gbit) [26, 45] and high read performance and bandwidth (cycle time around 85ns and 266MBytes/s) but with very limited write bandwidth (around 9MBytes/s) [26].

In this dissertation, Sections 1.1,1.2, 1.3 below, describe the problem we are solving, present the solution utilized and the results obtained, respectively. Chapter 2 introduces the PCM technology and present the related work. Chapter 3 describes the proposed architecture, PMMA. In Chapter 4, a novel wear-leveling is described. Chapter 5 proposes how to better select the technique to use excess capacity present on wear-prone memories. Chapter 6 present the conclusions and future work.

1.1 PROBLEM STATEMENT

Computing system are part of our lives today, not only in a recognizable form but ranging from the large clusters hidden in datacenters to embedded systems hidden in common objects. Achieving greater energy-efficiency is a requirement that appears in almost any system design, either because the system is energy-limited (using stored energy) or power-limited. In the path to that goal, large strides have been taken specially in dealing with power consumption in CPUs, but the path is still not complete. Main memory is today one of the major energy consumers and new memory technologies have been proposed. It is our goal to propose an architecture, techniques and algorithms that allow one of the most promising new memory technologies, PCM, to create a suitable replace-

ment to a DRAM main memory that has large gains in energy-efficiency. It is also part of this goal to integrate this architecture in current systems with minimal or no modifications to the CPU, OS and applications.

1.2 SOLUTION PROPOSED

Our main memory architecture, Phase-Change Main Memory Architecture, or PMMA, utilizes PCM, DRAM and SRAM in combination to achieve a high performance and energy-efficient replacement for a DRAM-only main memory. This new architecture is designed to replace the DRAM memory controller, requiring no modifications to existing CPU, DRAM devices or PCM devices that are commercially available or expected to become available soon. PMMA hides all aspects of the architecture. It exposes only what is necessary for a memory request to be made and executed. A number of algorithms and techniques are implemented in the memory controller to achieve the desired level of performance, energy-efficiency and lifetime.

1.3 SUMMARY OF RESULTS

PMMA is designed to achieve high performance with high energy-efficiency. As a validation tool, a highly configurable, energy and timing accurate main memory simulator was constructed. This simulator is able to simulate both PMMA and a DRAM-only main memory architectures with multiple variations in both architectures. A number of techniques were applied to mitigate PCM limitations, specifically performance and endurance, and also reduce the overhead of the architecture. The goal of having performance parity to a DRAM-only main memory main memory with better energy-efficiency was achieved. PMMA has only a performance impact of only 5% but consumes 50% less energy than a DRAM-only main memory. The use of a hybrid architecture is essential to obtain the performance and energy-efficiency. The size of PMMA memory manager is reduced by the use of larger pages and an asymmetric read-write partitioning.

The lifetime of PMMA was studied by creating an endurance management simulator, which simulates multiple wear-leveling algorithms. The endurance simulator was essential to determine the lifetime of the memory under worst case scenarios, by using multiple memory intensive applications running continuously. A lifetime of 8 years was achieved thanks to our novel low overhead swap-based wear-leveling algorithms. The algorithm uses random victim selection with a global counter that is within 25% of the lifetime of an idealized algorithm with an overhead of only 0.2% additional writes. The proposed wear-leveling algorithm is crucial to the lifetime and energy-efficiency of PMMA.

A study of how to achieve higher lifetime in wear-prone memories, PCM in particular, can be limited by process variation in memory endurance. Different models of cell lifetime variation, for example, bimodal and uniform are analyzed and rules are proposed that allow a designer to select parameters or techniques best suited to the objectives, either higher lifetime, lower overhead or lower cost. The specific endurance distribution, size of total memory size, desired lifetime or amount to be reserved as excess capacity are the necessary parameters to identify the technique.

2.0 BACKGROUND AND RELATED WORK

Phase-Change memory takes advantage of materials that have two or more stable states with different physical characteristics. These materials can change states reliably, in a very short time and with each state presenting large differences in easily measurable physical properties, optical reflectivity and electrical resistivity as the most common ones [43]. One of most common materials used in PCM is GST, a chalcogenide alloy of germanium, antimony and tellurium (GeSbTe). GST can exist in a amorphous state with low reflectivity and high resistivity and crystalline states that have high reflectivity (30% higher) and low resistivity (five orders of magnitude smaller). GST has been popular as optical storage medium due to its optical properties. The high speed crystallization process in GST, requiring less than 100ns, and the large changes in resistivity are the primary properties that makes PCM memory possible.

The process of changing the state requires controlling the temperature of the material to achieve the desired state. It uses the property of a phase-change material that will crystallize if it is heated above the crystallization threshold but below the melting point and it will become amorphous if melt-quenched [43](technique used to avoid crystallization by fast reduction of temperature). The state of the material determines the stored value so the read operation requires the measurement of the resistance of the phase-change material by applying a small current. The write operation require state of the phase-change material to be modified, and consequently a much larger current needs to be applied and a much longer timing is necessary. A large and short current pulse can be used to melt-quench the material and a smaller but longer current pulse can achieve crystallization. Both pulses can be independent of the current state of the material because GST will present almost the same resistance when the voltage applied is above a threshold, as shown in the curve in Figure 2 extracted from [43]. This property allows PCM to avoid an erase operation. The right

part of Figure 2 shows an example of two PCM memory cells with each in a different state, SET and RESET. A cell is in RESET when it is in the amorphous state and SET in a crystalline state.

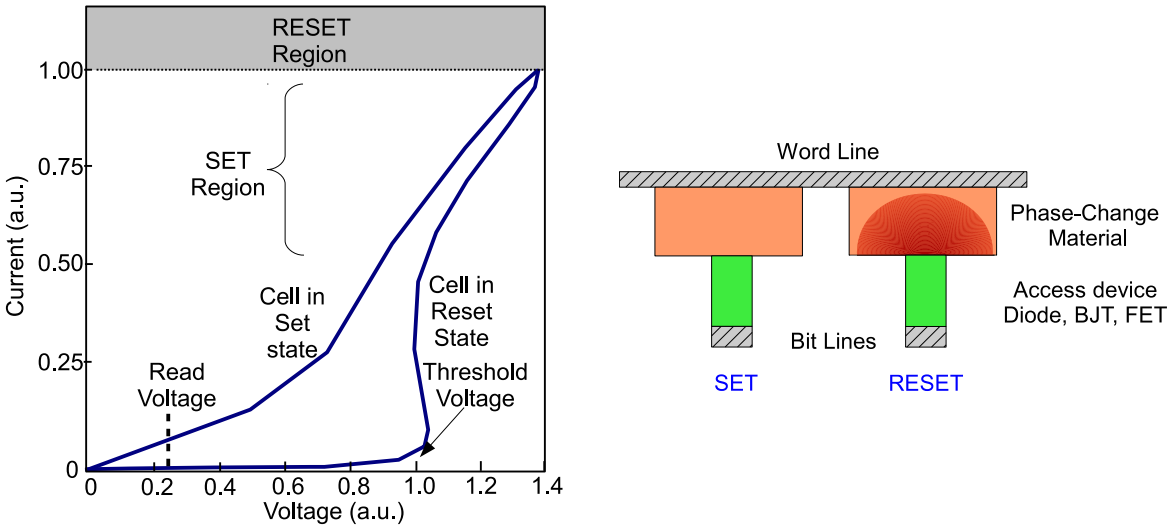

Figure 2: Example current-voltage behavior and of PCM memory cells, from [43].

A number of PCM properties can be extracted by looking at the way information is stored:

- Bit-addressability

  Each bit can be independently changed and there is no intrinsic requirement to change bits simultaneously.

- Non-volatility

  No energy is necessary to keep the state.

- Energy and latency asymmetry between reads and writes

  Reads are faster and much more energy-efficient since the value of current needed is much smaller and it can be applied for a very short period of time. Writes require a much larger current for a much longer time to achieve crystallization or to melt-quench the material.

- Radiation tolerance [30]

  Since the energy necessary to change the physical state of the material is much higher than the energy necessary to provoke ionization, a much more energetic radiation is necessary to modify the stored information.

- Bit value energy and latency asymmetry

  RESETs are faster and more energy-efficient than SETs.

- Sensitivity to higher operational temperatures

  Larger ambient temperature can crystallize the material, which makes operation and storage at higher temperatures challenging for PCM [43, 37].

- MLC (Multi-Level cells)

  MLC (Multi-Level cells) can be implemented in PCM by using partial crystallization to create intermediate states allowing larger memory densities. MLC is expected to be necessary to make PCM competitive to Flash and DRAM [13, 14].

PCM cell failure mechanisms determine the endurance, which ranges from 10^6 to 10^8 writes [13, 14]. Two of the common underlying causes of failures is contamination of the phase change material and delamination. Phase-change material contamination happens only in the melted stage since only in this stage contaminants can diffuse into the material. The cell can delaminate, separating the phase-change material from the metal or silicon that borders it, and create a void that is essentially an open circuit. Even though the achieved endurance of PCM cells are orders of magnitude higher than the 10^4 to 10^5 for Flash, it is not enough to be ignored in high performance memory subsystems. One of the most common cause for contamination or delamination is a phenomenon called overprogramming [46]. Overprogramming happens when a cell is being RESET and too much current is applied. In that situation, even phase-change material that is in the border of the cell is melted and now delaminate or contaminants can propagate through the border. This effect is cumulative and causes early failures. Using a small current is not an option because it will not melt a sufficient volume of material to reliably RESET the cell. Since these current values are not too far apart, prototype devices use a loop to program the cell, where at each interaction the cell value is compared to the expected value and, if it is different, a higher level of current is used to try to program the cell in the next interaction. This mechanism is very effective to make the endurance higher than 10^6 writes but has a large cost in terms of latency [26, 11, 21].

Even though the PCM concept has been introduced in the 1960s [38, 23], only recently large sized prototypes and commercial devices have become available [26, 35]. The size of the available PCM devices is still limited (512Mbits [26] and 128Mbits [35]) when compared to commercial DRAM devices (1GBits and 2GBits are common), but are much larger than competing technologies, such as MRAM [12] and STT-RAM [10]. PCM devices still use 90nm and 65nm technology

Attribute	DRAM	PCM	NAND Flash
Non-Volatile	No	Yes	Yes
Idle Power	100mW/GByte	1 mW/GByte	10 mW/GByte
Erase / Page Size	No / 64Bytes	No / 64Bytes	Yes / 256KB
Write Bandwidth per die	1-6GBytes/s	50-100 MB/s	5-40 MB/s
Page Write Latency	20-50 ns	1 μs	500 μs
Page Read Latency	20-50 ns	50 ns	25 μs
Endurance	10^{16}	10^{7}	10^{5}
Maximum Density	4Gbits	4Gbits	64Gbits

Table 1: PCM, DRAM and NAND Performance and Power Comparison, extracted from [13, 14, 37].

that is generations behind the of 32/28/22nm that is already in use for DRAM, so it is expected a rapid expansion in the size of PCM devices [13, 14].

As Table 1 shows, writes are main cause of PCM limitations since not only they are slower (10X to 20X) but also require a lot more energy (10X to 50X) when compared to reads. Two main PCM system limitations are closely related and caused by writes, high latency for writes and limited endurance. Due to these limitations, many researchers proposed techniques to avoid or mitigate those limitations. These techniques are explored in the next sections.

2.1 EXISTING SOLUTIONS FOR ENDURANCE

PCM has an endurance of 10^6 to 10^8 writes per cell [14]. If a specific cell is written once a second, it will take only 115 days for that cell to pass 10^7 writes [1]. This shows that, even with a very low write rate per cell of one write per second, PCM endurance is not enough to sustain a lifetime of 7 to 8 years. which is the expected lifetime of a server assuming obsolescence and capacity limits.

[1] Even if 10^8 can be guaranteed, this would be only 3 years at the rate of 1 write per second.

A number of techniques have been proposed to increase lifetime of PCM devices and PCM memory systems. These techniques varies from write minimization (avoiding unnecessary writes), to various forms of wear-leveling, to changes in bit encoding. Some important techniques are presented in the following subsections.

2.1.1 Write minimization

The goal of the techniques presented below is to remove writes that are redundant or modify the value to be written to reduce the number of bits to be altered. This can be done at the cell, row or even at higher levels of aggregation.

1. Read-before-write

   In this technique, a read is executed before the write and only the cells that the value differs from value to be written are updated, and the cells that already have the final value are not modified. Many PCM devices [11, 21, 26] implement this at the cell level. As described in Page 9, to avoid overprogramming multiple writes may be necessary and only the cells that are not correct (differ from the desired value) are written. In [25], it is proposed for removal of redundant writes.

2. Partial writes

   In this technique [25], higher level caches (the one closest to the main memory) track dirty word or line and pass that information to the main memory. The additional information allows PCM memory controller and devices to ignore the unmodified portions of the cache. In [41], a similar technique, called Line-Level Writes, is proposed that stores dirty information per cache line.

3. Flip-N-Write

   Flip-N-Write, proposed by [7], uses an additional bit in each row to determine if the value is stored in a normal encoding or inverted. The decision of which way to store is taken based on the lowest number of changed bits from the previous stored data and the new value. This technique reduces the number of bits to write at least by half, increasing write bandwidth to the memory. PCM write performance is power limited so more bits can be written simultaneously

within the same power budget. Less energy is consumed per transaction increasing energy-efficiency of the memory.

2.1.2 Wear-Leveling

In a system in which that all the cells have the same endurance, an uneven distribution of wear can cause cells that have more wear to fail prematurely, causing the whole memory to fail even though most of the cells are still healthy. Wear-leveling has the objective of distributing the wear evenly over the cells, avoiding premature cell failures and consequently increasing the memory lifetime. The memory access pattern of applications does not normally follow a regular pattern of an even distribution of writes over the whole memory, but it is highly skewed (see Chapter 3) directing a higher number of writes to a small number of memory locations. The memory subsystem have to counteract that access pattern to avoid premature failures and achieve a higher lifetime. Wear-leveling is applied to achieve this goal and can be implemented using different algorithms and granularities. A common implementation disconnects the physical address from a logical address using a mapping function that translates a logical address to the corresponding physical address. This mapping function allows the translation to be changed, pointing a logical address to different physical locations, throughout the lifetime of the memory effectively distributing the writes to the logical address over a larger number of physical address. Some of the wear-leveling algorithms use a configurable mapping function, others implement a mapping table. Some of the algorithms require the physical memory to be larger than what is addressable by a logical address (excess capacity), where others do not require but support this implementation. The wear-leveling algorithms proposed to be used in PCM memories are described below.

1. Row Shifting

   This technique, proposed in [48], implements wear-leveling at a row level. The mapping of the physical cells and the logical cells in a row is shifted by a certain amount at each write, This technique is designed to mitigate the difference in wear of cells in a row. In [48], it is shown that writes to a specific row in general do not modify all bits in the row but tend to affect only a small subset (low order bits for example). By shifting the row, the physical location of a particular bit changes, hence distributing the writes to a single bit over a number of physical

cells. In [48], it is proposed to use coarser shift instead of small ones (closer to 1bit shift), since modified bits tend to cluster.

2. Segment Swapping

   In segment swapping, each bank of memory is subdivided into fixed size segments. Each segment is composed of a number of rows. The mapping of a logical segment and a physical segment is flexible, by the use of a mapping table, allowing any logical segment to map to any physical segment in that bank. Segment swapping exchanges both the mapping between logical segments and the data, preserving the information but exchanging the physical segment that now holds the data. The translation of logical segment to a physical segment is kept in a mapping table. Segment swapping is proposed in [48] to create a more coarse level wear-leveling. The key to the technique is how to select the segments that will be swapped. The segments chosen in [48] are the one with the number of writes since the last swap is above a threshold and the least written one. Segments that are 1MByte in size are used, in [48], to reduce the overhead in counters and the size of the mapping table. The memory is unable to respond to requests whenever a segment is being swapped, so even the number of swaps is small (every $2 \cdot 10^6$ writes) the latency of copying this amount of data can be significant, since 2MBytes have to be transferred.

3. Fine-Grained Wear-Leveling

   In [41], this technique is used to achieve intra-page wear-leveling. Each page of 4KBytes is divided into 16 sub-pages and a shift number is added to the logical sub-page number to map to the corresponding physical sub-page. In [41], this shift number is changed only when the page is allocated by the operating system and it is not changed until the page is freed. The shift number is allocated randomly whenever it is changed.

4. Start-Gap

   Start-gap, proposed in [40], is a very low overhead wear-leveling technique. The low overhead is achieved by using a programmable mapping function instead of a mapping table. The mapping function uses two counters, the start and gap counters. The start counter marks which physical address corresponds to the logical address 0 and the gap marks the start of the gap, a sequence of physical addresses that are not currently mapped to any logical address. The physical pages work as a circular buffer with the mapping determined by a start and gap counters.

Periodically, based on number of writes, the gap counter is incremented and the information stored at the previous end of the gap is copied to the physical address pointed by the gap counter. This action remaps the logical address that corresponds to the gap entry to a new physical address. Since the physical memory is treated as a circular buffer, a particular logical address will point to all physical addresses during the lifetime of the memory.

2.1.3 ECC

Some memories utilize some form of error correction, and in NAND Flash and PCM, ECC has been proposed as way to deal with cell failures. ECC reserves additional bits to store redundant information allowing errors to be detected and corrected. ECC is heavily used in servers DRAM main memory where memory errors cannot be tolerated. In a write, the error correcting code is generated and stored in addition to the original information. In a read, the stored error correcting code is compared to the one generated on the fly, if the original error correcting code and the newly generated do not agree, an error has occurred. The ECC is capable of correcting errors up to a determined number of incorrect bits and detecting but not correcting for a larger number of incorrect bits. The limits are determined by the number of additional bits used for ECC. In [44], ECC is determined to be detrimental to PCM and Error-Correcting Pointers (ECP) is proposed. ECC is detrimental to PCM because it requires more bits to be written for each write, increasing power consumption and decreasing write bandwidth. ECC also increases the number of bits modified in each write, because a single data bit changed will affect some or all bits in the error correcting code. ECC is used to detect bits that changed after writes which is useful in DRAM or flash, but not so useful in case of PCM since after a bit is successfully written it will not fail. ECP repurposes the excess bits to be used as pointers and replacements bits. When a write to a memory cell fails, a set of pointer and replacement bit is allocated and the correct value is stored in the replacement bit and the pointer contains the number of the failed bit. This removes the need of computing the error correcting code both in the write and the read, and reading the correct bit is a matter of swapping the incorrect bits pointed by the ECP code. ECP also avoids increasing the number of bits to be written by changing the pointer only when a new failure is detected otherwise the pointer and replacement bits are kept at 0. It is shown that ECP leads to a longer lifetime than ECC.

2.2 EXISTING SOLUTIONS FOR PERFORMANCE

PCM write latency is one of the major limitations of the technology. Even though PCM reads are expected to be two times slower than DRAM reads, PCM writes are one or more orders of magnitude slower than PCM reads [37]. Write bandwidth is also severely limited by power, since each cell written consumes a significant amount of power, limiting the total number of cells that can be written simultaneously. Read bandwidth does not suffer from that limitation and it is one to two orders of magnitude higher than write bandwidth. One of the simple solutions that has been adopted is write avoidance: to reduce the number of writes to PCM, by both reducing the total number of write operations and the number of bits that are written. Write avoidance has a significant energy and latency impact. Other performance improving techniques have been proposed such as write cancellation and a improved PCM requests scheduler.

2.2.1 Write Avoidance

The techniques listed in the Section 2.1.1 can also improve performance. Flip-N-Write can double the write bandwidth to the PCM devices by reducing the number of bits that are simultaneously written to the memory. Partial writes and Read-before-write when used at a line/row level potentially can remove unnecessary writes. The drawback of partial writes is the requirement of additional dirty bits to store the dirty status in a higher granularity. Read-before-write requires a read operation before a write and in worst case can have a small increase in latency since reads are much faster than writes.

2.2.2 Request Preemption and Pausing

PCM Request preemption is used by [49, 39] to improve performance by reducing the impact of high latency operations. Both reads and writes can be preempted. The advantage of preemption is reduction of latency for higher priority requests even if another operation is already ongoing for that bank. Read preemption increases energy consumption since the operation has to be reexecuted in the future. Writes are the large latency offender in PCM, they are up to 10x slower than reads and are often not in the critical path. Write preemption (write cancellation [49]) has the potential

to improve latency but incurs in energy and endurance impacts. The additional energy and wear are consequences of the reexecution of the write since the previous write was not completed and the stored value differs from the expected. Both approaches [49, 39], try to limit the impact by not allowing preemptions to occur to the same request too many times. In [39], an alternative operation is proposed, called write pausing, that pauses the write operation between steps in the loop (over-programming) and resume the write later. This operation does not incur in energy overhead, since the operation is nor repeated but restarted from the point it was stopped. The request is paused so no additional latency is necessary beyond the delay caused by the requests interposed to the paused one.

2.2.3 PCM QoS Scheduling

A new memory controller is proposed in [49] that implements a scheduling algorithm that is more amenable to PCM. The scheduling algorithms proposed in [49] are a modification of PAR-BS [36]. PAR-BS is a fair scheduling algorithm designed for CMP systems and it is modified to include write and read preemption in an effort to improve latency for higher priority requests. The preemption of requests that can be close to finishing does not improve latency significantly and can be detrimental to energy efficiency. Threshold limits are implemented to preclude preemption to requests that will take less than the threshold to finish. The advantage of specializing the memory controller to PCM is the ability to explore the difference in internal operations between PCM and DRAM. DRAM, contrary to PCM, requires that a read be followed by a write in a row, since reads are destructive where PCM reads are harmless. PCM can have preemptable reads or writes, where preemption in DRAM are not possible since that would provoke information loss in reads and a single write buffer prevents simultaneous operation (read with a write date buffered). This is possible in PCM since read data do not have to be stored so the information can be preserved.

2.3 EXISTING PCM MAIN MEMORY ARCHITECTURES

PCM use as main memory has been proposed by [41, 25, 48] and others. The search has started for a possible DRAM replacement since it may be reaching the scaling limit [13] and PCM appears as a possible candidate. PCM has limitations and needs architectural solutions to achieve performance and endurance that are acceptable to system designers.

In [48], PCM is used as main memory for a 3D stacked chip to obtain power and energy savings, characteristics that are essential in a 3D stacked chip. In this proposal, MLC (Multi-Level Cells) PCM chips are stacked over a CPU chip and interconnected using a high bandwidth interface by taking advantage of the 3D construction. The high read bandwidth can offset the higher read latency of PCM when compared to DRAM. The use of MLC increases write bandwidth by reducing the number of cells that need to be written. PCM has very low idle power reducing the thermal requirements of the 3D chip.

In [25], a PCM device is redesigned to be more amenable as main memory. The main change are in the increase of row size for reads and the inclusion of a write buffer that coalesces writes. PCM density is increased by use of a 2-bit MLC. The asymmetry of reads and writes and the endurance impact are mitigated by using the techniques mentioned at Section 2.1;

The architecture proposed by [41] utilizes both DRAM and PCM to create main memory. DRAM is used as a large cache and PCM as a large main memory. The DRAM cache is used to mitigate the write impacts on PCM, large latency and limited endurance. The larger latency that is characteristic of PCM is reduced by using 4x larger main memory and accounting for the reduction of virtual page faults. The architecture is presented in a very high level and it is assumed that PCM will be 4x denser than DRAM. The use of DRAM as a large cache towards improving lifetime and endurance of a larger PCM storage is very compelling solution and the architecture presented on this thesis share this characteristic. The details of each implementation can make substantial negative or positive impact on performance and energy-efficiency. Using the algorithms described on the following chapter, we show that even at the same size, a large main memory constructed with PCM can be competitive with PCM.

3.0 PCM MAIN MEMORY ARCHITECTURE - PMMA

In the previous chapter, we surveyed existing solutions for mitigating PCM performance and endurance limitations. A number of existing architectures that explore the use PCM as main memory were also presented. In this chapter we introduce a new main memory architecture that uses PCM as main memory. This architecture provides large energy and power savings with very little performance impact as we will show in Section 3.3.

The new Phase Change Main Memory Architecture (PMMA) addresses the shortcomings associated with PCM, namely endurance, write vs. read asymmetries, and performance. PMMA utilizes a set of architectural and algorithmic solutions, described in the next sections, to achieve the desired lifetime (comparable to a server lifetime, around 8 years) and to mitigate the performance impact of PCM. Large main memory subsystems for servers are the main application for PMMA, since power is a major limitation in server design and one of the major advantages of PMMA. PMMA is designed to replace DRAM main memory without impact on the CPU architecture and to leverage existing DRAM and PCM devices. PCM devices are assumed to have an internal operation similar to existing PCM prototypes [26, 45].

3.1 PMMA

PMMA is a hybrid architecture that uses multiple memory technologies, such as PCM, DRAM and SRAM, to achieve high performance and high energy-efficiency. PMMA uses PCM as main memory, DRAM as a high performance local cache and SRAM as buffering and metadata storage in the controller. In this work, it is assumed that the memory controller is off-chip, even though PMMA can be integrated as an in-chip memory controller. Figure 3 contrasts a current DRAM

architecture (left) with PMMA (right). The PMMA design goal of transparency to the CPU is achieved by utilizing the same interface to the CPU as the one that exists between the DRAM memory controller and the CPU in a conventional architecture.

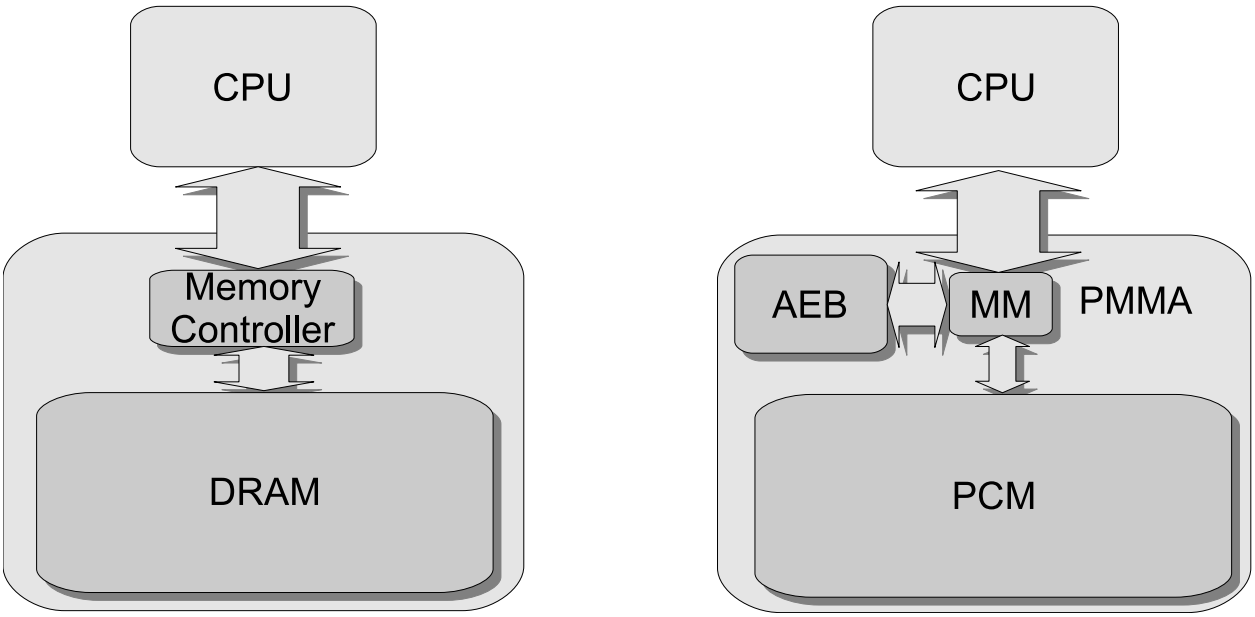

Figure 3: Typical DRAM architecture and proposed PMMA Architecture

PMMA has two auxiliary components, in addition to PCM devices: the Acceleration and Endurance Buffer (AEB) and the Memory Manager (MM). The AEB is much smaller but faster memory than PCM. The MM is responsible for managing the internal operation of PMMA. Its functionality is a superset of a memory controller in a conventional DRAM architecture. The MM uses the AEB and PCM as randomly accessible storage and controls the information flow between CPUs, PCM and AEB. One of the major functions of the MM is to manage the limited endurance of the PCM cells by implementing algorithms to enhance the lifetime of the system. PMMA's PCM memory is designed to have at least a lifetime equivalent to the lifetime of a server (7 to 8 years). PMMA supports multiprocessor operation with multiple pending requests per CPU and a split-transaction bus (requests and responses are sent asynchronously).

We describe the PMMA components in detail next. Figure 4 shows PMMA's internal structure for the AEB, MM, and PCM.

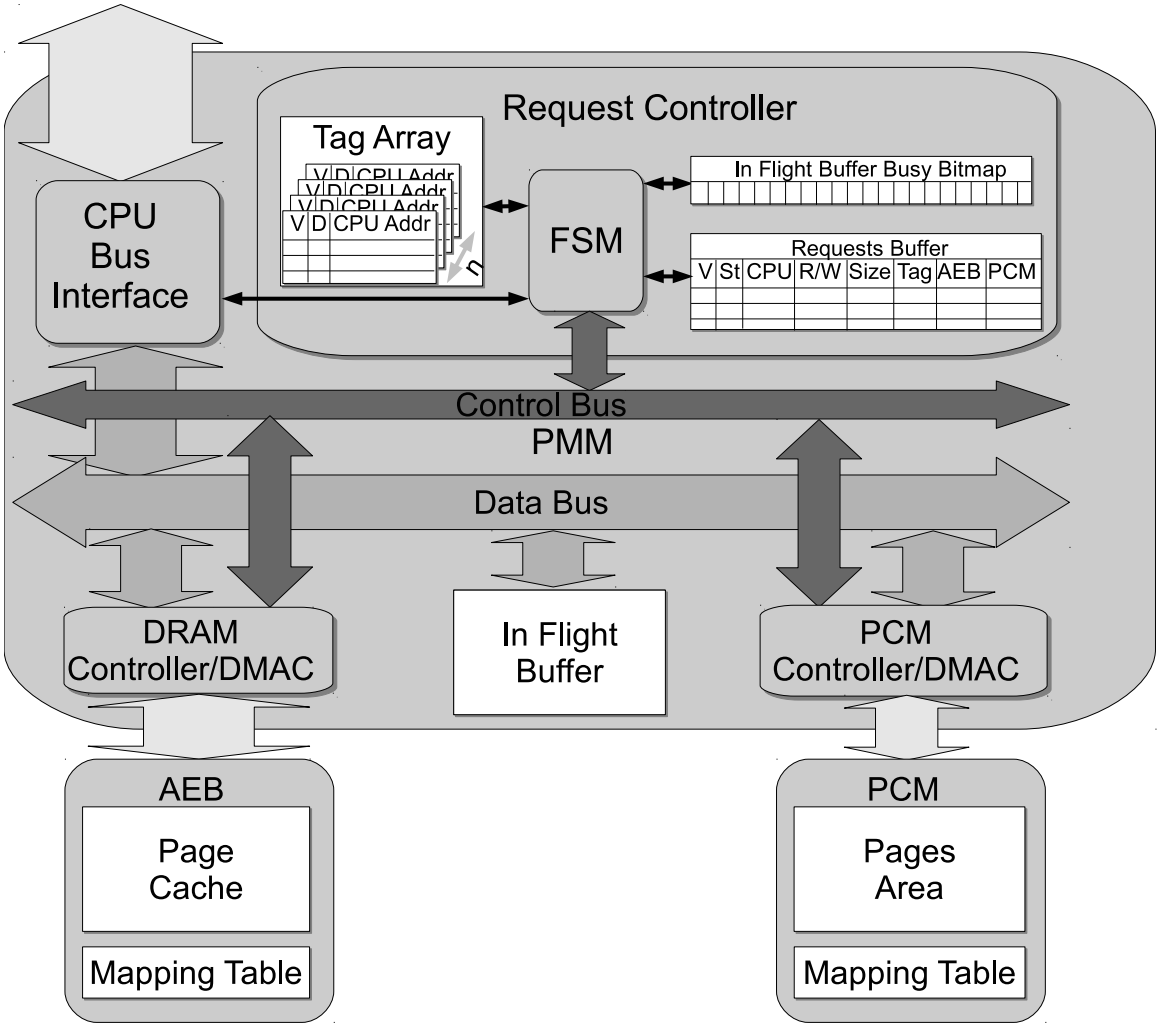

Figure 4: MM Architecture

3.1.1 PCM: Phase Change Memory

PCM memory devices are managed by the MM. PCM devices are connected to the MM, as shown in Figure 4, via the PCM controller and data/control buses. The PCM controller has a direct memory access controller (DMAC) that allows it to initiate transactions directly to the data/control buses without requiring intervention by the request controller. A memory bus is used to interconnect the PCM devices to the MM; current prototypes of PCM [11, 26, 35] use an interface similar to SDR/DDR DRAM. Although it would not influence the design of PMMA, we expect that DDR DIMM form factor and controller interface will likely be reused by PCM modules to ease integration with existing systems.

The interconnection of PCM devices to the MM is an important design aspect and heavily influences performance and power consumption. The design space for the interconnection is large: the number and width of buses and the number of DIMMs per bus are the primary parameters. More buses result in more parallelism and bandwidth, but increase the complexity of the PCM controller and the number of chip pins. The use of wider buses increases available bandwidth but also increases interconnection cost (i.e., more pins). Using more DIMMs per bus allows more physical memory per controller, which reduces system cost, but there is an inverse relation between bus speed and the number of allowed connections (DIMMS). Since PCM uses a low speed bus (PCM is intrinsically a slower technology [26, 45]), its memory controller can support more DIMMs per controller than a conventional DRAM architecture [20]. In our initial design, we favor a single PCM bus to reduce controller costs and implementation complexity.

Figure 4 also shows what is stored in the PCM devices. The *pages area* holds the actual data stored. The data is arranged in relatively large chunks, called pages. PMMA uses large page sizes (1KByte to 4KBytes) to reduce cost and improve performance, as described and evaluated in Section 3.2. The *mapping table* area holds the translation table that is used to map a CPU physical page address to a PCM physical page address (as described in Section 2.1.2). This table is also stored in the AEB for performance reasons but it is replicated in the PCM for persistence.

3.1.2 AEB: Acceleration and Endurance Buffer

The AEB is a fast memory module that uses DRAM as a write buffer and high-speed (relative to PCM) data cache. By design, it is much smaller (8 to 64 times smaller) than a conventional DRAM main memory. We limit the AEB to a single, fast DIMM to reduce physical bus size, load and capacitance, while enabling a high bus frequency. A number of high speed memory busses, such as Rambus XDR [42], have been proposed that would be a good fit as the AEB interconnection to MM. The high speed nature of the new bus technologies limits the number of devices that can be interconnected to a single bus. DRAM was chosen over SRAM for the AEB to have a more energy-efficient solution and larger capacity. The AEB has to be at least a couple of hundred of megabytes, as shown in Section 3.3.2, for some workloads.

As Figure 4 shows, our initial AEB design has two storage areas: a *page cache* and a *mapping table*. The page cache acts as a large data cache, used to improve performance and endurance; it holds application data at page granularity. The tags for CPU addresses in the page cache are kept by the Memory Manager (see below) rather than in the AEB, to expedite lookups.

The AEB also stores metadata – the mapping table – for endurance management of the PCM components. It maps CPU physical page addresses to PCM physical page addresses. This table is a duplicate of the one stored on the PCM devices. The duplicate is kept in the AEB (DRAM) to permit fast lookups. The number of requests to the mapping table in the AEB is limited since a mapping table access is needed only when a request is a miss at the AEB. The performance impact to the majority of the requests is minimal but a large latency reduction is achieved for the misses, since a slower request to the mapping table at the PCM is not required. Note that the page size can have a dramatic impact on the spare table size: a very small page size (e.g., the same as a L2/L3 cache block size) would result in an exceedingly large table. In turn, for a given total AEB capacity, less storage space would be allocated to the pages area with a small page size. Instead, a large page size allows a more modest mapping table size, leading to more storage (for a fixed capacity) for the pages area. A smaller page size for PCM than the one used for the AEB will require multiple requests to the mapping table, since each PCM page size can be remapped to a different location, increasing latency and impacting AEB performance.

3.1.3 MM: Memory Manager

The Memory Manager has several components as shown in Figure 4. The primary components are the request controller, a request buffer, an In-Flight Buffer (IFB), a PCM controller and a DRAM controller (AEB interface). The request controller is responsible for receiving requests from the CPU interface, allocating resources necessary for the requests (such as IFB buffers) and executing the transactions on behalf of the CPU. The request buffer stores information about pending requests that are required during the execution of the requests. It stores the current state of a request, including its CPU/DRAM/PCM addresses, size of the transaction and resources reserved (e.g., buffers and tag array entries). The In-Flight Buffer is used as temporary data storage by the request controller. The PCM and AEB controllers have DMA engines to read and write data from the devices at a higher granularity than a single bus transaction.

In PMMA, all CPU transactions are made to the AEB, rather than the PCM. Any miss on the AEB will read the page from PCM and store it in the AEB. Writes are done to the PCM devices only when a dirty/modified page is evicted from the AEB (i.e., a writeback), which reduces the total number of writes. Because writes in PCM are expensive in time and energy, a reduction in the number of writes leads to better energy-efficiency and higher performance. Furthermore, a reduction in writes means that PCM cells will potentially "live" longer, thereby improving endurance.

During writeback of a dirty page, the memory manager checks for PCM device wear-out. A write-read-verify process is done by our PCM controller: an evicted page is written to the PCM device, it is read back and the read data is compared to the evicted page. The evicted page is kept at the IFB until the write-read-verify process is done. If page wear-out is detected, that is, the write operation is not successful, a spare is used to replace the "broken" page and the spare tables in the AEB and the PCM are updated. Both tables are updated to ensure fault tolerance (i.e., the mapping is not lost, due to PCM non-volatility) and good performance (i.e., the table is accessed from the AEB).

Because the AEB is used as a data cache, tags are needed. A SRAM tag array is incorporated in the Memory Manager (separately from data in the AEB) to permit a fast tag check (on the critical path of *all* memory requests). The tag array is an SRAM implemented as an n-way set-associative tag store. The size of the tag array is determined primarily by the number of pages available at

the AEB. Associativity has a smaller impact. A larger associativity tag array requires more logic than a lower associativity one. The associativity of the tag array has three main constraints: power consumption (favors lower associativity), miss rate (favors larger associativity) and size (favors lower associativity). A design constraint originates from the need to reserve area on the AEB to store metadata. In a n-way set-associative array, each way maps a $1/n$ fraction of the data cache. Hence, $1/n$ of the memory will not be used by the cache if one of the ways in a n-way is not implemented. A larger associativity can tailor the $1/n$ to be closer to the fraction of the AEB required by the metadata, a smaller associativity will waste memory by reserving more area than it is needed.

The tag array is the largest structure in the MM; its size is linearly related to the number of pages available in the AEB. A large page size reduces the size of the tag array for a fixed AEB size. The size reduction of the tag array is an important reason to use large page sizes in PMMA, this is evaluated in Section 3.3.4.

The In-Flight Buffer is used to increase parallelism and to decouple the PCM and DRAM controllers. It is a small storage area on the MM that receives pages to be transferred to/from the AEB and PCM. The IFB allows multiple transfers to occur simultaneously on the AEB and PCM busses. The write-read-verify process to detect wear-out also uses the IFB. The data written to a PCM page is kept in the IFB until the correct operation of the write is verified. This arrangement decouples wear-out detection from the operation of the AEB and servicing of other requests but increases the size of the IFB and requests buffer since the request will take longer to finish. The IFB size is based on the number of requests that can be executing simultaneously at the system. Its bandwidth requirements is the sum of the bandwidth of the AEB, PCM and the CPU interface. Design choices here include implementing the IFB with SRAM or eDRAM, and choosing between fast IFB technology or using a larger internal bus size. In PMMA, a SRAM IFB is used since the size of the IFB (32KBytes and 128KBytes), is not large. If the IFB, DRAM controller (for the AEB) and PCM controller (PCM main memory) are located on the same chip, the data bus can be widened so the IFB can achieve the desired performance with almost any technology.

The IFB is managed by the request controller. A specific CPU memory request can use none, 1, or 2 buffers. The primary function of the IFB is to allow the decoupling of AEB and PCM transactions for the same request. Without the IFB a larger latency would be imposed in the CPU

requests. A request does not use the IFB when a needed page is in the AEB and the transaction will occur between the AEB and the CPU interface since either the data is available already, in the case of writes, or we want to forward it to the destination as soon as possible, in case of reads. The use of the IFB in this situation would only increase latency without any additional benefit. The request controller uses 1 buffer when a miss occurs (in the AEB) and 2 buffers when a miss with an eviction is needed (see Section 3.1.4). In a miss, one buffer in the IFB is utilized to store the page that is read from the PCM. The page comes in multiple bus transactions allowing the IFB to fulfill other requests until the page read is completed. In a miss with eviction, the IFB is more important, since it allows the PCM read to occur even though the allocated page in the AEB still is being evicted. Without the IFB, the PCM read would have to be postponed until the AEB would be able to receive the new data (after the page eviction or one PCM write). The request controller uses a busy bitmap to manage the IFB, since only an information of busy or free is necessary for an allocation, the specific buffers allocated to each request are stored in the request buffer. This avoids the need to search the request buffer for an unallocated buffer.

Figure 4 shows that PMMA has a finite-state machine (labeled FSM) that controls the operation of memory actions. The FSM uses the current request state and events coming from the DRAM and PCM controllers to determine the next action to execute.

The FSM uses the request buffer to track memory requests. The request buffer stores, for each CPU memory request, (1) information about the original CPU transaction, namely read or write, CPU address, size of the transaction, CPU interface tag, write data in case of an write; and (2) internal MM information, namely AEB physical page address, IFB buffers allocated and PCM physical page address (see Section 3.1.4 below).

PCM and DRAM controllers are responsible for the interface with the actual memory devices. These controllers are also responsible for scheduling requests to their respective devices and implementing acceleration techniques like critical word first (see section 3.1.5 below). Memory ordering hazards, namely RAW (Read After Write), WAR (Write After Read) and WAW (Write After Write), are avoided in this implementation by requiring that all requests to the same page be executed in order but requests destined to different pages can be reordered for better performance. A more sophisticated approach could be used to disambiguate the requests allowing more parallelism, but this requires more logic at the FSM and likely occurs infrequently since the CPU caches

would prevent transactions to the same address in a short distance (number of memory transaction). The sequential ordering is kept for correctness and the request controller is responsible for checking the request buffer to enforce it.

3.1.4 PMMA Operation

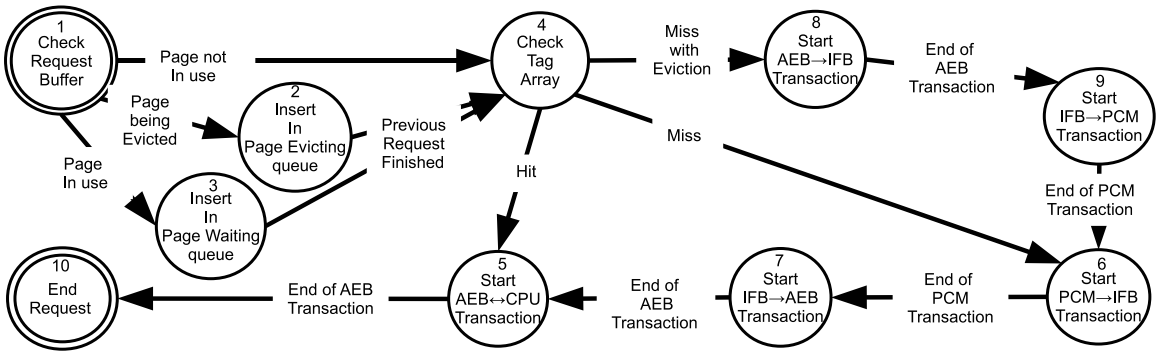

Figure 5: Simplified FSM state diagram for a single request

PMMA uses the CPU physical address to identify the page that will be used to fulfill the request. The CPU physical address does not correspond to a specific physical location in the AEB or PCM, since both rely on translation tables to correlate the CPU physical address to an AEB or PCM physical address. The FSM in Figure 5 describes a simplified PMMA operation. The full state machine has a very large number of states that are used to increase parallelism or are used for the endurance algorithms. The necessary modifications of the state machine are described later (see Page 28). In the figure, the action in a circle represents the operation executed to enter a state. The arrows are the possible results of an operation.

During operation, the PCM page address of a new CPU request is checked against the request buffer (state 1), to identify if the page used by this request is cached at the AEB or is being used (processed) by another request, as shown in the Figure 6. A request can be blocked by a previous request that uses the same page or if a previous request evicted the page that this request needs and eviction process is still in execution (see Figure 9). In both situations (states 2 and 3), the new request will be restarted when the previous requests finishes. The states on Figure 6 are designed to maintain FIFO order for the requests that are directed to the same page, but allow reordering of requests that go to different pages (enabling them to run in parallel). The shaded states in Figure 6

correspond to states where the request is not in execution and waiting either for the checking if the page is in use or waiting for previous requests to finish. State 4 in Figure 6 correspond to the request in execution (AEB, IFB or PCM transactions will be executed).

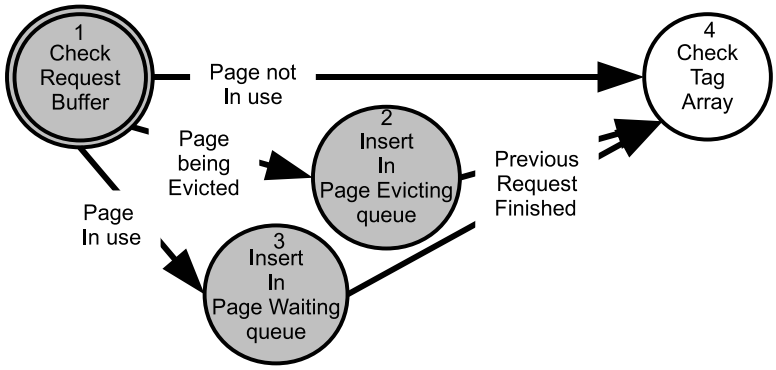

Figure 6: Concurrency control and tag matching states of the FSM state diagram. Shaded states maintain the sequential ordering of the requests.

The tag array is checked next (state 4). A hit implies that the AEB contains a valid copy of the needed page and the CPU transaction can use the cached version of the page (state 5), as shown in the Figure 7. A *miss* requires that the page in PCM be brought to the AEB before the CPU transaction is executed (to be executed in states 6 and 7), as shown in Figure 8. In case of a *miss*, state 4 also updates the tag array by replacing the previous AEB page entry address and status bits with the information of the page being read from the PCM. State 4 is also responsible for allocating an IFB buffer to receive the PCM page.

Figure 9 shows the relevant states whenever a *miss with eviction* occurs, requiring the AEB cached page to be written back to the PCM (to be executed in states 8 and 9) before the AEB cache page can be reused (to be executed in states 6 and 7). In case of *miss with eviction*, state 4 executes the same operation as in a *miss* and also allocates a single buffer in the In-flight Buffer (state 4) because the eviction (AEB read and a PCM write) will be executed before the update with the new page (PCM read and AEB write) can be executed. A single buffer can be used for both operations, eviction and update. In a miss with eviction, a race can happen if a request to the page that is currently being evicted is received by the FSM. To avoid the race, the FSM updates the tag as soon as the request that caused the miss with eviction allocates this entry, prohibiting any new requests to use the evicted page. Any new requests directed to the page that is in the process of

being evicted will wait until the page is evicted and then they will be go to the tag array again creating a miss and bringing it back from the PCM. This case is expected to be very rare since if a page was selected for eviction it was not accessed for a while.

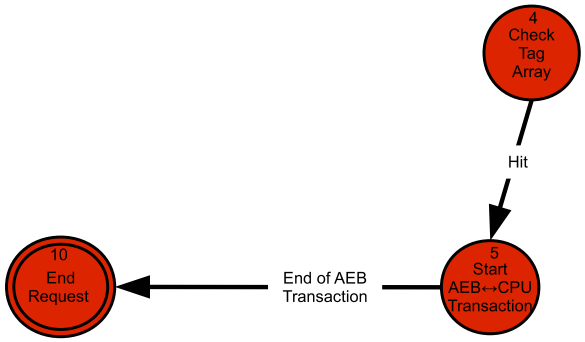

Figure 7: AEB Hit operation of the FSM state diagram.

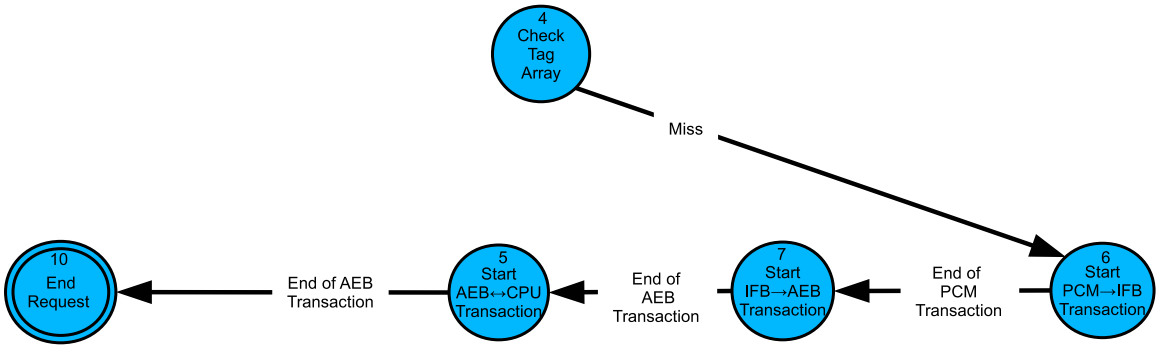

Figure 8: AEB Miss operation of the FSM state diagram.

The diagram in Figure 5 is a simplified view and does not show the following optimizations that actually exist in our design and implementation:

- Operation for state 4 can be done speculatively in parallel with state 1. The result of state 4 is ignored if the FSM moves from state 1 to states 2 or 3.
- For a CPU read, the needed information is available inside the MM (at the IFB) at the end of state 6 and data can be immediately forwarded to the CPU.
- A CPU write can finish even though the full transaction is not completed by using the MM as a write buffer. This allows a response to be sent to the CPU at the end of state 6 or 8 because the transaction has already started.

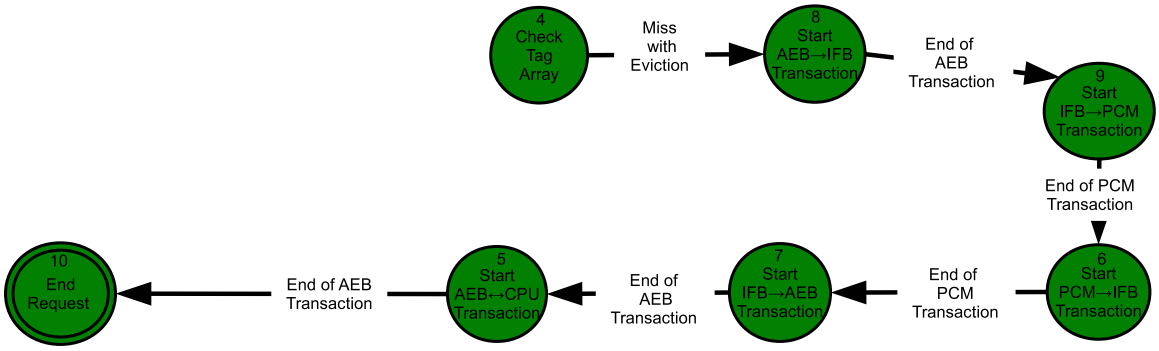

Figure 9: AEB Miss with eviction operation of the FSM state diagram.

- For a miss with eviction, states 6 and 7 can be done in parallel with states 8, 9 by using different in-flight buffers to store the data and with the constraint that state 8 is finished before state 7 is started. This requires adding intermediate states to the FSM to account for the fact that any of the two asynchronous transactions (AEB$\longrightarrow$IFB and PCM$\longrightarrow$IFB) can finish before the other. It also requires two IFB buffers be allocated in state 4 instead of one and the IFB busy bitmap to be updated.

The use of a FSM that allows parallel execution of transactions, e.g. states 6 and 7 in parallel with 8 and 9, reduces the latency of memory operations and therefore increases the number of requests pending at the MM, assuming the CPUs will be able to send more requests to the PMMA. Even after a request sends a response to the CPU, it may stay in execution at the PMMA until all the internal operations finish (e.g, writeback). The buffers must be sized properly, as lack of resources would reduce PMMA's throughput by reducing the number of requests it can support simultaneously. The number of buffers can be estimated by computing the number of simultaneous requests that the PCM subsystem of PMMA can support.

The wear-leveling mechanism, described in Section 4.1, requires modification to the state machine. Three additional operations are required to implement wear-leveling: mapping table access, write failure recovery and page swapping.

1. Mapping Table Access

   Before any read or write of a page from PCM, the mapping table in the AEB has to be accessed to translate the physical address sent by the CPU (logical address in the context of the PCM)

to the current physical PCM page address that contains that logical pages (See Section 4.1). Figure 10 shows that an AEB read is necessary and has to occur before any PCM transaction (states 6 and 9) in the FSM described in Figure 5.

2. Write Failure Recovery

   Write failures require a simplified version of the swapping operation. In this simplified swap, the new physical page is allocated from the excess capacity pool and no data is copied from the new physical page. It is assumed that the page had no logical page data stored in it. The failed write is reexecuted using the new physical page as the destination, and the old physical page is discarded by simply not reinserting it in the mapping table, effectively removing this physical page from the pool of available physical pages (see Section 4.1 and Figure 23). The additional states are inserted after state 9 of Figure 5.

3. Page Swapping

   Page swapping is the operation that implements the wear-leveling. A page swap exchanges PCM physical pages between two CPU physical pages (See Section 4.1). In a page write to PCM, a swapping operation can be triggered. In this case, a new physical page is selected, the original value that exists in the new physical page is copied to the previous physical page and the mapping table entries are exchanged. Figure 11 shows the transactions required to execute a swapping operation. The original page is written at the new physical location, so effectively this operation performs the swap of the physical pages between two logical pages. The FSM in Figure 11 is inserted before state 9 of Figure 5.

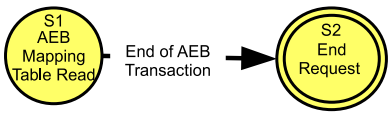

Figure 10: Additional states for consulting the AEB Mapping table.

3.1.5 Performance Enhancements

A **page** is the unit of logically managed data at the PMMA. A **block** is defined as the amount of data that is transferred to/from PCM in a single transaction. A block is transferred with multiple bus transfers, each of which is called a **bus transfer unit**. Each bus transfer unit is composed

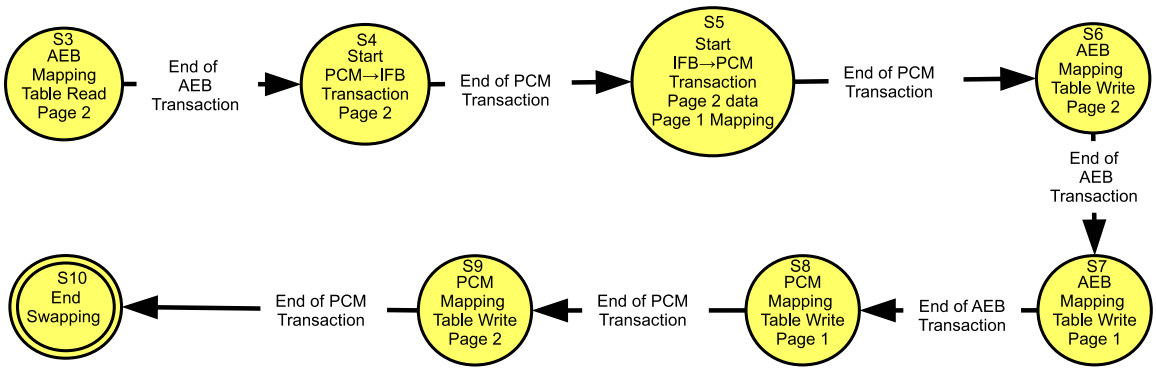

Figure 11: Additional states required to execute a swap.

of sequential bus transactions each transferring a physical bus width worth of data. The physical bus width, block and page sizes are defined by PMMA. The bus transfer unit is a property of the specific bus protocol used (DDR2/3 or a variation). Some constraints affect the choice of sizes: a page is always larger than a block, which is always larger than a bus transfer unit.

PMMA performance is improved by a number of techniques. These techniques are designed to address the major limitations of PCM, such as large read/write latency and limited PCM bandwidth.

Critical word first. The use of critical word first in the requests that read data from PCM can lead to a large reduction in latency. The PCM controller has to be modified to read data from the PCM starting at the proper offset so the requested word is the first to be read. Typical controllers start reading from the initial page address (0 offset). The PCM controller also has to inform the request controller when the necessary data is available at the IFB, since it is only a subset of the whole transfer.

AEB bypass. The IFB can be used to provide the data for read requests directly to the CPU without waiting for all the data to be transferred from the PCM or to be sent first to the AEB. This is clearly true for critical word first, when the CPU request that is waiting for the data was the one responsible for the PCM request. It is also true when servicing a subsequent request to the same page and the page is still being transferred. In this case, if the data corresponding to the requested address is already available at the IFB, it is sent to the CPU to finish the request. If the data is not available yet, it will be sent to the CPU as soon it is copied to the IFB. The implementation of this improvement requires that the request controller dynamically query the PCM controller to

determine whether a requested address is available at the IFB or scheduled to be transferred. If the specific address is scheduled, the PCM controller will signal the request controller when the address becomes available otherwise the request controller can already transfer the data from the IFB.

Read-Write-Read (RWR) Read-write will be used to avoid sending writes to PCM that do not modify the existing information already stored. In this case, each block of data is first read from PCM, and only the bus transfer unit that contain modified information is sent as writes to PCM. For each bus transfer unit written, an additional read is made, and the information is compared to validate if the write was successful. In case of failure, a replacement page (write failure operation) is allocated, and the failed one discarded. Most devices already implement a similar mechanism (see Section 2.1.1) and which in this case it is redundant.

Page Partitioning. As previously discussed, a large page size reduces the amount of metadata; however, it has the negative effect of increasing the transfer time, bandwidth and energy consumption at the PCM and AEB interfaces. Page partitioning is a technique to lower metadata size and offset the costs of a large page size. Each page is divided into subpages. A single entry in the tag array is maintained for each page and valid and dirty bits are kept for each subpage. Because PCM reads are much less expensive (in terms of energy and time) than writes, we propose a mechanism to distinguish between the two operations and use large subpages for reads and small ones for writes. A small write subpage reduces the number of bytes written to PCM. In turn, this reduction lowers the time to write and consumes less energy. This optimization requires each entry of the tag array to have a valid bit per read subpage and a dirty bit per write subpage. This is complementary to the RWR mechanism mentioned above because it avoids initiating a write operation to PCM, where RWR would have to read each block of data from PCM to identify that the block is unchanged.

Cache replacement policies. Due to PCM's asymmetry on reads and writes, eviction of a dirty block from the AEB should have a lower priority over a clean block. We designed the *clean-preferred LRU* algorithm that modifies a traditional least-recently used (LRU) algorithm to evict the next-to-least-recently used page instead of the least-recently used page when the least-recently used page is dirty and the next-to-least-recently used page is clean. This algorithm tries to strike a balance between keeping dirty pages and keeping younger clean pages by only looking at the last

two LRU pages. This algorithm is useful in PCM, because it trades an expensive operation (a write to the PCM) for an increase in the probability of requiring a lower cost operation, namely a read to the recently evicted clean page.

We note that Critical word first and AEB bypass are very useful and have no negative impact since they always reduce latency with a little modification of the FSM and the PCM controller. Clean-preferred LRU could have a theoretical negative impact (i.e, larger number of read misses), but it is expected to have a positive net effect in all non-pathological benchmarks. Thus, they are implemented in PMMA. The effect of page partitioning is examined in Section 3.3.3, since the impact is less predictable.

3.2 ARCHITECTURE EVALUATION

A PMMA simulator was designed and implemented to evaluate the performance and energy impact of the memory architecture; the input to the simulator is a memory trace, which is obtained by collecting memory references with Simics [29]. We use Simics to simulate the CPU, L1 and L2 caches with a zero-latency main memory. In other words, in the resulting memory traces, delays are caused only by the CPU, computation (multithreaded applications), and caches (L1/L2 latency). The memory trace contains, for each memory request by the CPU, a time stamp (assuming zero memory latency), type of operation (read or write) and the CPU virtual and physical address used in the operation.

3.2.1 Simulator

Figure 12 shows a high-level view of the simulation architecture and the internal structure of the PMMA simulator. The simulator is able to model a conventional DRAM architecture and PMMA, allowing for a direct comparison between the two architectures in the same experimental framework. The input to the simulator is a memory trace and a configuration file that contains the description of the parameters used in the simulation. The configuration file can alter many characteristics of the memories or enable or disable all optimizations.

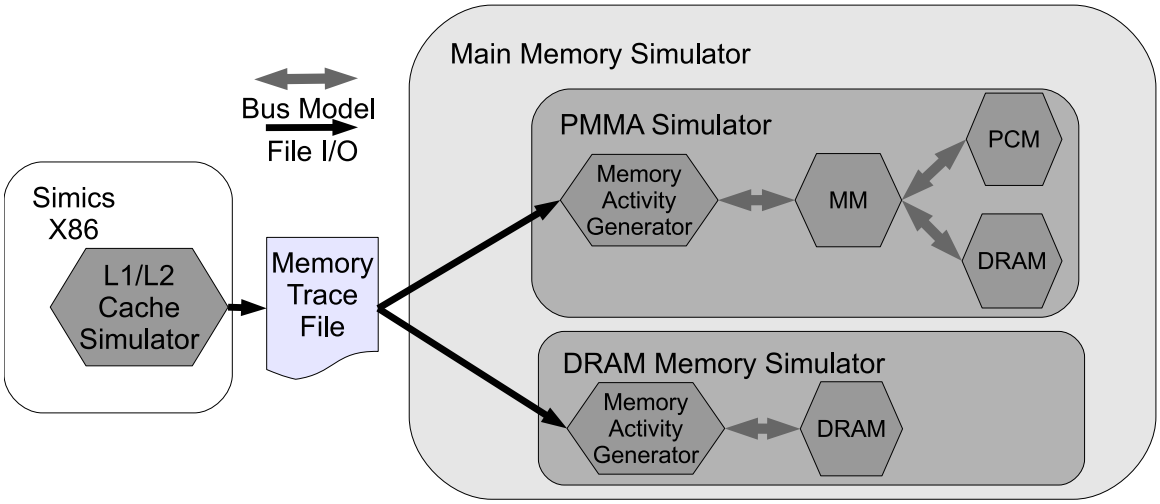

Figure 12: Simulator Architecture

The PMMA simulator has a memory activity generator module that processes the memory trace and initiates memory requests to other simulation modules. The simulator also models the memory manager, the PCM controller and devices, and the DRAM controller and devices. The DRAM, PCM and MM simulators export the same API that implements a split transaction model with callback, allowing the simulation of other architectures by reconnecting the modules. A split-transaction model allows asynchronous operation by decoupling the initiation of a request and its response. For example, the simulator uses FIFO with priority scheduling for all shared resources and accounts for latency due to resource contention, like request buffers, IFB buffers, internal busses, PCM controller queue, DRAM controller queue, PCM bus and DRAM bus. PCM and DRAM controllers have internal queues of finite size (parameterizable). PCM and DRAM controllers implement flow-control by using a queue-full signal that back-propagates and halts the sender until the queue can accept new requests. The parameters used to obtain the results of Section 3.3 are shown in Table 2.

Timing and power models for DRAM were extracted from DDR2-DRAM devices from Micron [34]. Two DRAM devices were modeled, one is a better fit for a small and fast DRAM memory and the other is more suitable to a large but slower DRAM memory. The latter represents the largest capacity chip available today, while the former is a fast and wide device that can be used for AEB (which requires a small number of DRAM devices, privileging speed over size).

Parameter	PCM	333MHz DRAM	533MHz DRAM
Bus Size (Bits)	8	8	16
# Banks	8	8	8
# Rows	32768	16384	16384
# Columns (Bytes)	1024	1024	1024
Bus Cycle Time (ns)	16.7	3	1.87
Read Latency (ns)	66.8	15	15
Write Latency (ns)	334	15	15
Read Bus Speed (MHz)	66	333	533
Write Bus Speed (MHz)	33	333	533
Idle Current (mA)	1	7	7
Read Current (mA)	10	160	170
Write Current (mA)	70	160	170
Vdd (V)	1.8	1.8	1.8
Max requests in queue	16	16	8

Table 2: Parameters used in the simulation

The implemented model supports memory DPM (Dynamic Power Management) [34], where the memory enters a lower power state when idle. DRAM refresh was not modeled because it is a small percentage of energy consumed (around 1% to 5% [34]) and has a small impact on performance (also around 1% to 5% [34]). By not including DRAM refresh, the energy and latency of the DRAM-only system with be underestimated giving it an advantage when compared to PMMA; PMMA is able to show performance and energy improvements even without accounting for the refresh impact.

The primary function of the memory activity generator is to simulate sending the requests to the off-chip memory and accumulate statistics for each request. Given the zero-delay memory traces, the memory activity generator determines the request issue time assuming that the CPU sustains a single memory request at a time and the difference between the time stamps is due to computing time and L1/L2 latency, since the traces were created with a zero-latency memory the timing has to be adjusted. A new memory request is only issued to PMMA after the previous requests finish and the computing time has elapsed with computing time being estimated by the interarrival time of the requests. For example, assume that in the trace file request N_1 had a timestamp TS_1 and request N_2 had timestamp TS_2, since in the trace file the memory latency was zero, $TS_2 - TS_1$ correspond to the computation and cache latency. Assume that the memory activity generator sent request N_1 at timestamp TE_1 to the PMMA simulator, and the response to the request N_1 came back at timestamp TER_1. Request N_2 will be sent to the PMMA simulator at a timestamp $TE_2 = TER_1 + (TS_2 - TS_1)$. This models a CPU that can sustain only a single pending memory request and no write-buffer at the lowest level cache (L2 in our case).

The MM simulator models the operation of the Request Buffer, the Tag Array and request controller. The Request Buffer model assumes that it will process a single event at a time and all events consume a fixed amount of time with a FIFO queue at the input to order the events in case of contention. The tag array can only process a single query at a time (also FIFO), allowing a more accurate model of a single-ported cache. The FSM (request controller) is modeled as a very fast state machine, compared to DRAM and PCM access time, with a fixed latency for all operations, such as initiating a PCM or AEB memory request.

Each of the DRAM and PCM simulators has a memory controller with a finite size input queue. Requests can be reordered. The memory devices (chips) can be aggregated in multiple

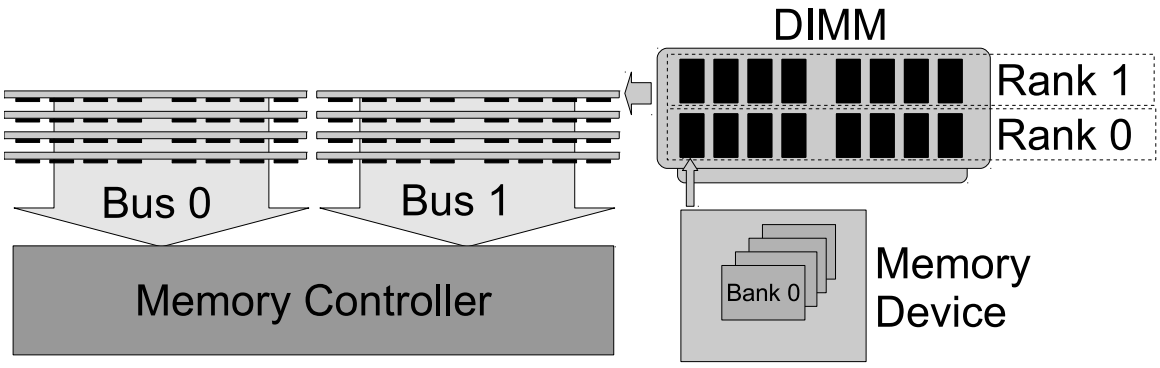

Figure 13: Physical Memory Architecture

ranks, DIMMs and busses. A rank is a set of devices that are active to respond to a single request, a DIMM can be composed by one or more independent ranks. Each DIMM is connected to a single bus and the simulator allow the use of multiple independent busses. Figure 13 shows how the memory is modeled in the simulators. Each chip can have multiple independent banks and the DIMM bus width is configurable in multiples of device bus width. Busses, devices and banks uses a modified FIFO scheduling with reordering, which can only occur when a later request can be scheduled without changing the schedule of previous requests or a higher priority request arrives, priority is defined as a request that is in a critical path for a CPU request. This reordering allows low latency requests for banks that are free to proceed when a long latency operation (PCM write) happens in another bank on the same bus.

A PCM or a DRAM memory request has to be mapped to the devices that will serve this request and the mapping function uses the trace-generated physical address provided to determine which bus/DIMM/rank/bank will be used for this access. The mapping used assumes that each bus transaction (a bus transfer unit) is contained in a single bank, so a transaction will involve only a single bank.

3.3 EXPERIMENTS AND RESULTS

3.3.1 Experimental Setup

The Simics configuration used to generate the traces has four 1.6GHz x86 cores, each with dedicated L1I and L1D caches (4-way, 32KBytes, cache line size of 64 bytes and a hit latency of 1 cycle) and each core pair share a L2 cache (16-way, 4 MBytes, cache line size of 64 bytes and a hit latency of 6 cycles). The traces were collected until 200Million main memory accesses were seen, which is enough to validate the architecture because a large page size makes the AEB warm-up with a few million main memory requests (around a few hundred million instructions). The warm-up with a low number of requests happen because the transfer unit is a sub-page that is larger than a typical cache line, otherwise the warm-up period would be at least one or two orders of magnitude larger.

The baseline DRAM main memory system was configured as having a total of 16GBytes and using a 333MHz 64bit DDR2 bus with 16 DIMMs. Each DIMM is a 1GByte DRAM composed of 8 memory devices each with 1Gbit. The PMMA was configured with an AEB with a physical size 512MBytes of DRAM memory. The AEB interconnection to the MM is a single 64bit DDR2 533MHz bus. The DRAM memory at the AEB is composed of 4 DRAM memory devices each with 1Gbit of memory, bus of 16bits and 533MHz bus speed. The logical size of the AEB was varied to identify the impact of the AEB size on the applications, but the physical configuration was kept constant to keep the energy and timing information comparable. The PCM memory is a 16GByte using a similar configuration as the DRAM-only system also with 16 DIMMs with each DIMM with 1GByte. The PCM interconnection to the MM is a single 64/128bit 66/133MHz DDR2 bus.

The benchmarks used to validate the system were chosen based on memory footprint and main memory utilization. Multithreaded applications and a mix of applications were used to validate the design with multicore systems and increased memory pressure. PARSEC [6], SPECcpu2006 [8] and SPECjbb [8] were the sources of benchmarks used. From PARSEC, Canneal and Facesim were used since they are multithreaded, have a large memory footprint (2GBytes) and have a high miss rate ($\geq 10\%$) at the AEB. From SPECcpu2006, GCC, mcf and bwaves were chosen because they

have the largest memory footprint and are main memory intensive (highest miss rate) among all the applications, according to [17]. A mix of SPECcpu2006 applications, bwaves, GCC, mcf and bzip2 were used as a multiapplication, multitasking benchmark (seen as SPECmix in the results). SPECjbb2005 was also used because: (a) it has a large memory footprint, around 2GBytes; (b) it is main memory intensive; (c) it is multithreaded; and (d) it represents a large Java application, which have specific memory access patterns due to garbage collection.

In the simulations, we record how long it took to serve all memory requests in the trace and the energy used for each element (MM, PCM and AEB). These metrics allow comparison between PMMA and a conventional DRAM architecture with energy-delay product as the primary metric (a small performance loss is acceptable if that generates large energy gains).

3.3.2 AEB and Page Size

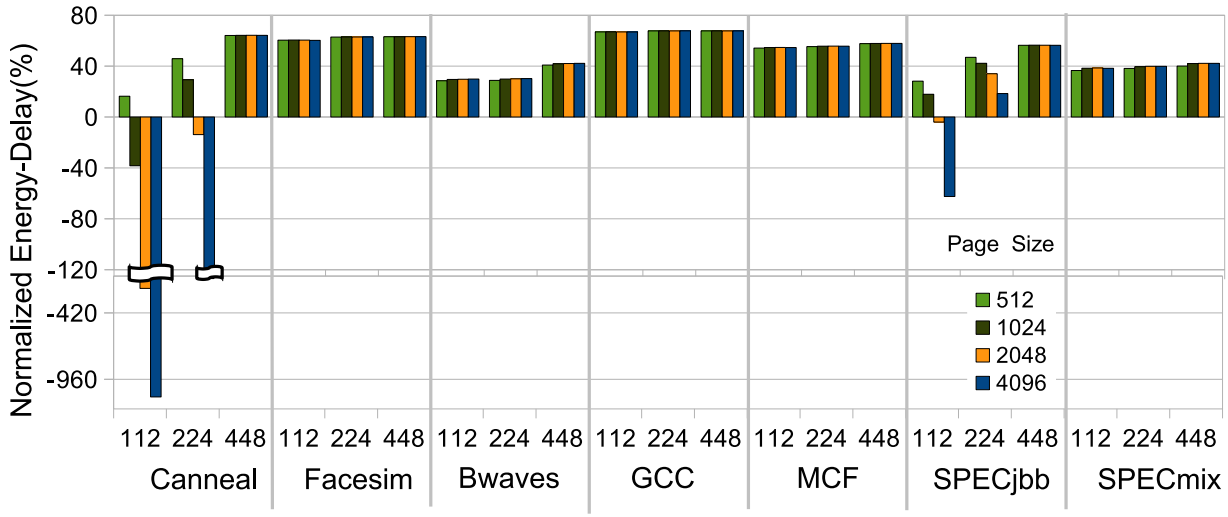

Figure 14: Impact of AEB size (112, 224 and 448MBytes) and AEB page size (512, 1KByte, 2KBytes and 4KBytes) on energy-delay normalized to a DRAM-only memory (gain is positive).

Figure 14 compares the impact of different AEB sizes and AEB page sizes on energy-delay. All values presented are normalized against a baseline, namely a DRAM-only main memory subsystem. Positive results indicate improvements compared to the baseline. PMMA includes the use of critical word first and AEB bypass enhancements. The main result is that PMMA in most configurations (AEB cache size, page size) have large gains in energy-delay, up to 65%. Larger

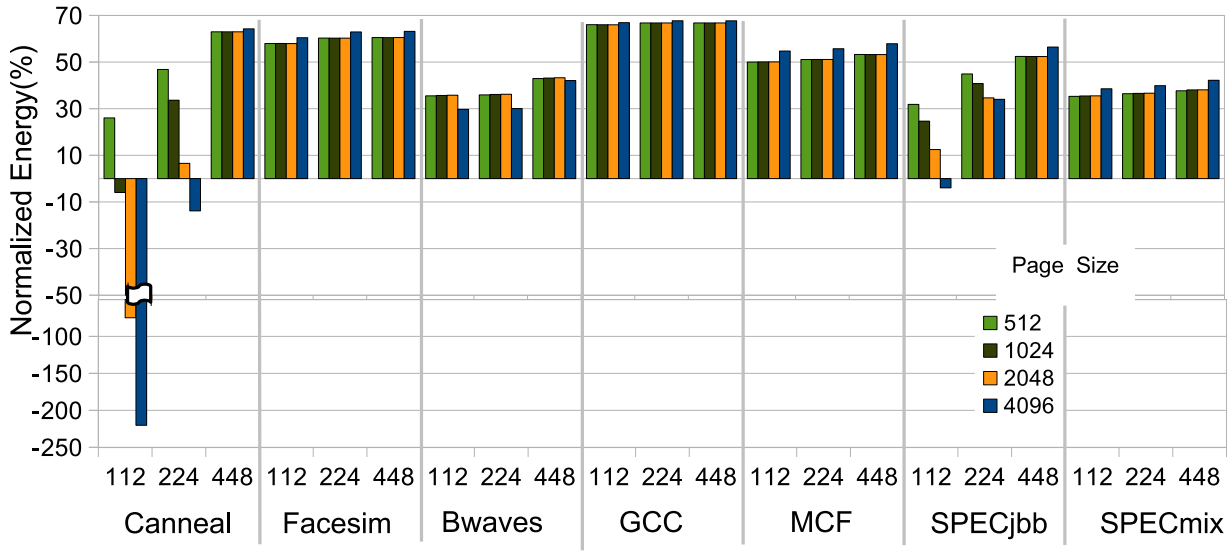

Figure 15: Impact of AEB size (112, 224 and 448MBytes) and AEB page size (512, 1KByte, 2KBytes and 4KBytes) on energy normalized to a DRAM-only memory (gain is positive).

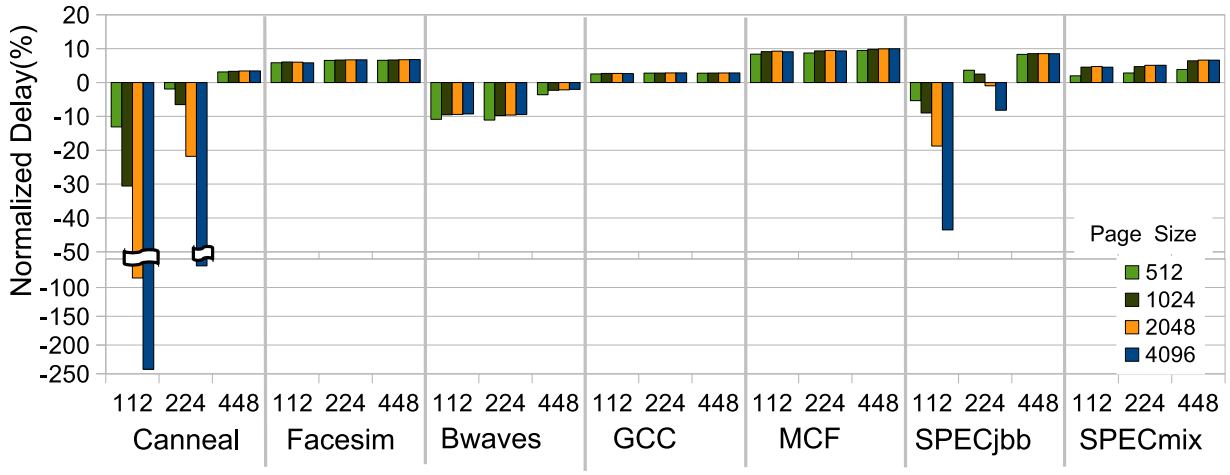

Figure 16: Impact of AEB size (112, 224 and 448MBytes) and AEB page size (512, 1KByte, 2KBytes and 4KBytes) on delay normalized to a DRAM-only memory (gain is positive).

AEBs are always beneficial for the applications but have a negative impact on energy, performance and system cost. Some applications have a higher profit from a larger AEB than others, namely Canneal, Bwaves and SPECjbb.

In contrast, page size has a mixed effect, although three different patterns can be seen in the figure when page sizes are increased: marginal increase in energy-delay (SPECmix, Bwaves and MCF), much worse energy-delay (Canneal and SPECjbb), and no variation. In other words, increasing page size yields either small gains or almost exponential negative effects. This clearly shows that small pages should be used for performance. Unfortunately, small pages increase the size of the Tag Array, which becomes a limiting factor since the size of the Tag Array is dictated by how much area of the chip can be allocated to it. The higher latency and smaller bandwidth available on an off-chip Tag Array would negate the performance gain achieved with smaller page sizes.

Figure 14 can be better analyzed by looking at each metric separately. Figure 15 shows energy gains only. PMMA shows positive energy gains of up to 60% on all configurations for all applications, with the exception of Canneal and SPECjbb. Canneal and SPECjbb show energy gains for smaller page sizes when the AEB is smaller and for all page sizes when a larger AEB is used. As with energy-delay, energy gains also always improve with a larger AEB size but page size does present a more complex relationship with energy where some applications benefit from a larger page size and others prefer a smaller page size. The applications that benefits from larger pages, presents higher intrapage locality, profiting from the prefetching mechanism.

Figure 16 shows delay (application runtime) gains and losses, positive and negative, respectively. Only modest delay gains on most benchmarks can be achieved since AEB has a similar latency compared to the DRAM architecture. The gain comes from the IFB and the prefetching that is a natural consequence of transferring pages from the PCM to the AEB. Applications that present temporal locality at a page level will benefit from this reduction in latency, but the gain is not expressive, as seen in Figure 16, since the prefetching is intra-page only and do not improve inter-page accesses. Canneal, SPECjbb and Bwaves are the only applications that show performance loss; Canneal and SPECjbb, as mentioned before, show a very strong negative relationship with page size. This is because these applications have poor spatial locality at the page level as shown in Figure 17. Otherwise, Bwaves does show a lower negative impact with larger pages than

the other two applications that originates from a higher intra-page locality that offsets some of the negative impact. Poor spatial locality and high miss rate is very detrimental to PMMA because every miss transfers a whole read subpage that only a subset is useful and the high number of miss requires a large number of transfers. The PCM bus becomes the bottleneck and limits the performance of PMMA. Canneal has the largest measured (AEB) miss rate and is the only application that shows increase in (AEB) miss rate when a larger page is used. This shows that Canneal has a large memory footprint but also has little short-distance spatial locality (4K page). Bwaves is also an interesting case: even with a large cache, the miss rate is large. However, the miss rate is reduced by 75% with a large page size, see Figure 17. This indicates strong short-distance locality with poor global locality. The beneficial effect of using a larger page size (essentially prefetching) is only realized if the miss rate is substantially reduced when a large page size is used. Otherwise, the PCM bus bandwidth will be consumed transferring whole read sub-pages when only a fraction of it is useful, wasting bandwidth and limiting the performance. Only a few of the benchmarks tested have sufficient short distance spatial locality to benefit much from large pages.

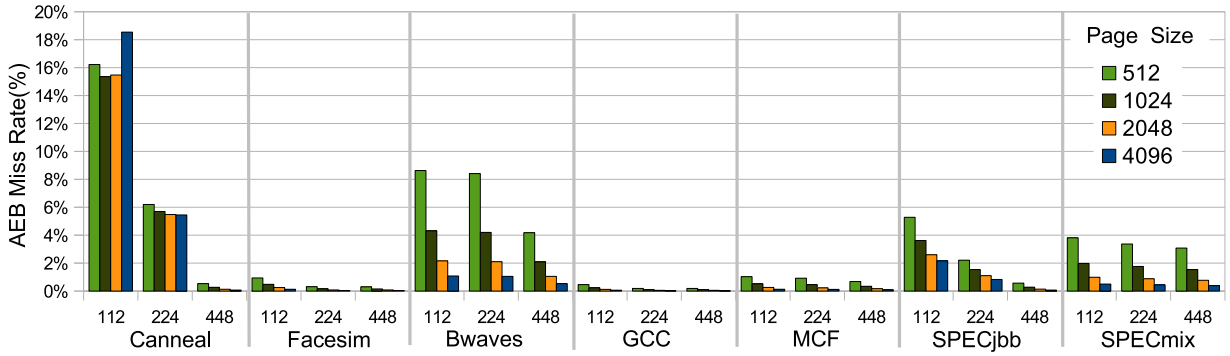

Figure 17: AEB Miss rate per application, AEB size (112, 224 and 448MBytes) and AEB page size (512, 1KByte, 2KBytes and 4KBytes).

From a design perspective, a smaller AEB size is preferable since it allows the use of faster DRAM memory. The densest DRAM devices tend not to be the fastest and also if more than one rank is needed, a faster point-to-point interface cannot be used. From our experiments, an AEB size of 224MB satisfies the needs of almost all tested benchmarks. Hence, this AEB size is used as the default value in the remaining experiments. Page size determination requires more analysis since small page sizes are more efficient (smaller energy and delay) but considerably more costly

in terms of metadata. A 2KBytes page will be used since in the next Section page partitioning will be able to offset the performance loss. A 4KBytes page is too large and even the use of page partitioning is not sufficient to achieve an acceptable performance.

3.3.3 Page Partitioning

In Figures 18, 19, 20, and 21, energy-delay improvements (positive) or losses (negative) are shown when Page Partitioning is applied. In these figures each configuration is labeled by page size–read subpage size–write subpage page if partition is applied otherwise only the page size is shown.

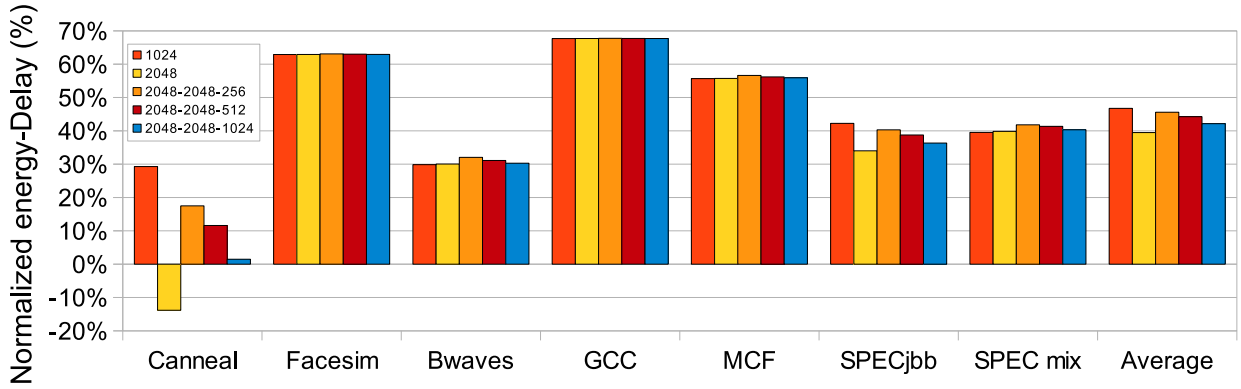

Figure 18: Impact of write page partitioning (256, 512, 1K) with a 2KBytes page on energy-delay normalized to a DRAM-only memory. Each configuration is labeled Page Size[–Read subpage Size–Write subpage Size].

Figures 18 and 19 shows the impact of write partitioning when applied to 2KBytes or 4KBytes pages. The results show gains in all configurations and applications. As expected, smaller write subpages have higher gains, which indicates that pages are frequently being evicted with most of it untouched. It is also apparent by examining Figures 18 and 19, that even though a page size of 2KBytes and 4KBytes can have bad behavior for some applications (Canneal and SPECjbb), write page partitioning can mitigate a loss in energy-delay and even change a loss into a gain. Some applications such as Bwaves, MCF and SPECmix have a better energy-delay using 2KBytes page and 256Bytes write subpages than any other tested configuration. Figure 19 shows that using subpages does improve the energy-delay but not enough to offset the impact of a 4K page for Canneal.

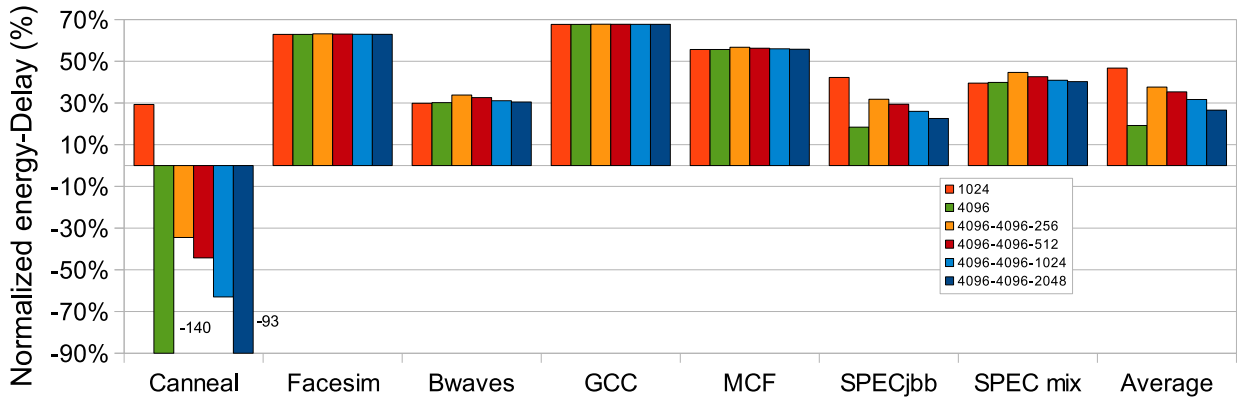

Figure 19: Impact of write page partitioning (256, 512, 1KByte, 2KBytes) with a 4K page on energy-delay normalized to a DRAM-only memory. Each configuration is labeled Page Size[–Read subpage Size–Write subpage Size].

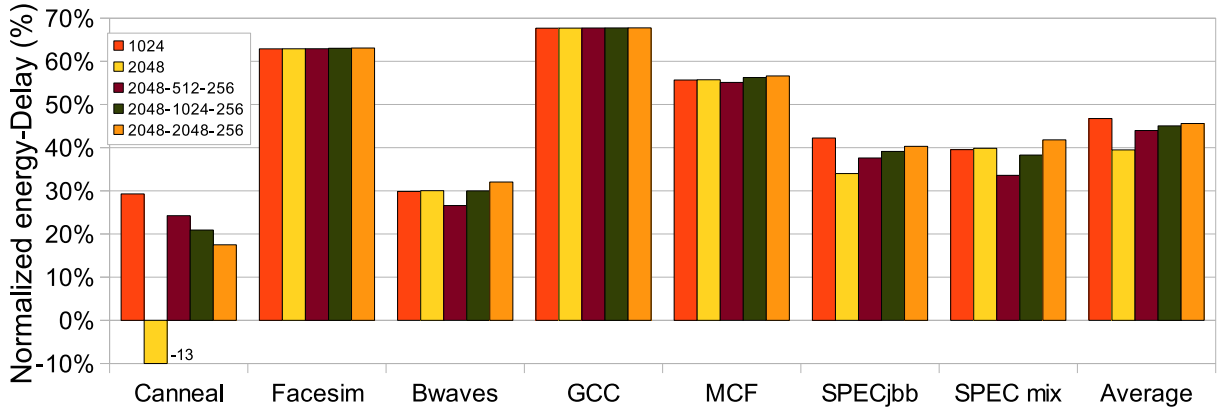

Figure 20: Impact of both read (512,1KByte and 2KBytes) and write page partitioning (256) with a 2KBytes page. Each configuration is labeled Page Size[–Read subpage Size–Write subpage Size].

Energy-delay results for different sizes of subpages for reads and writes (write subpages are smaller than read subpages) are shown in Figure 20. Results are mixed. Applications in which read subpages are beneficial are the ones that have less spatial locality and also have high miss rates, so reducing the congestion at the PCM bus improves performance. The other applications suffer since any eviction will remove a whole page but a miss will only bring one subpage, which loses the prefetching effect of large pages. The use of read subpages allows the designer to use a smaller Tag Array for the same AEB size as analyzed in the next Section.

3.3.4 Technology Constraints

This section, two main constraints of PMMA are analyzed to identify the impact and solutions. The first is the MM size that is determined by the Tag Array and IFB, the largest structures of the MM. The impact of changing PCM technology characteristics, bus speed, latency, bus size is also analyzed to determine what would be impact on PMMA if PCM devices are slower than predicted A larger AEB (448MBytes) and smaller pages (1KByte) would achieve the highest performance and significant energy savings mitigating a slow PCM memory, but also would be difficult to implement by requiring a large Tag Array. Smaller configurations of AEB with larger page sizes can reduce the size of the Tag Array, reducing the area occupied and the PMMA.

3.3.4.1 MM size PMMA physical size is dictated by the size of the tag array and the IFB. The tag array can be seen as the tag portion of a CPU cache and the IFB is a RAM block. Note that large pages sizes lead to a small tag array, with a large IFB. The IFB must have an integer multiple of pages, accounting for at least one buffer per possible pending request at the request buffer. Page partitioning changes the IFB size by transferring at most one read subpage at a time (not the whole page) reducing the IFB size proportionally. At the Tag Array there exists an entry per available page at the AEB, so it depends on the AEB size and page size. Page partitioning also increases Tag Array size by requiring more bits per entry. Figure 21 shows the impact of page size and subpage sizes (read and write) on the MM size (obtained from Cacti 5.3) compared to a 2KBytes page (no read or write subpages) for a 224MB AEB size when accounting for the area consumed by the IFB

and Tag Array. Note the 2K-1K-256 have the same size as 2KBytes, the IFB is much smaller (half of the size) and compensates from the increased size of the Tag for each entry.

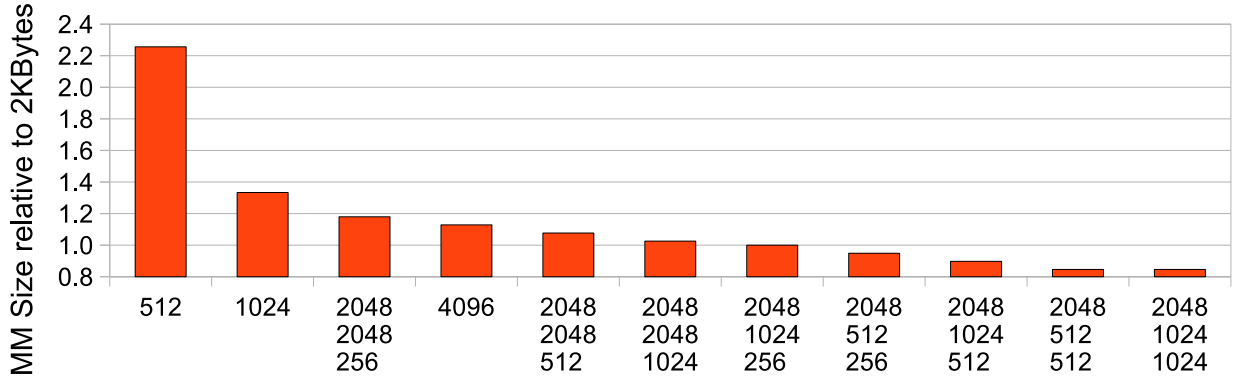

Figure 21: Impact on MM size (Tag Array and IFB) of AEB page size (1KByte and 2KBytes) and read (512, 1KByte and 2KBytes) and write page partitioning (256, 512 and 1KByte).

3.3.4.2 PCM technology PCM devices are available only as prototypes and have interface specifications that are slow compared to DRAM devices in production today. Our base design used a 133MHz device with a 60ns latency, but we carried out a sensitivity analysis to find the impact of PCM technology. Figure 22 shows the impact of reducing the speed of the PCM bus from 133 to 66 MHz and lowering latency of the PCM devices from 60ns to 30ns. The figure shows clearly that a lower speed PCM would hurt some applications that are more sensitive to bandwidth, high miss rate and lower locality (e.g., Canneal). Lower speed PCM would only be useful in a small page configuration (ruling out large pages). This result also shows that reducing PCM latency has minimal effect on energy-delay.

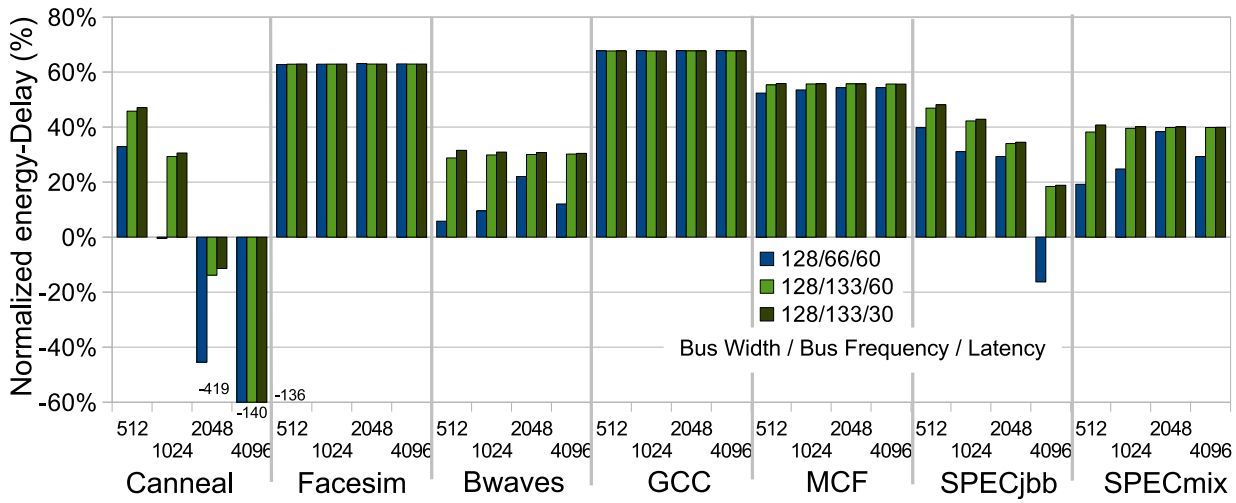

Figure 22: Impact of PCM bus speed (66 and 133MHz), PCM bus latency (30,60ns) and AEB page size (512, 1KByte, 2KBytes and 4KBytes) on energy-delay.

4.0 PCM RELIABILITY AND ENDURANCE IMPROVEMENTS

PCM endurance is one of the major hurdles that limits the use of PCM in main memory. We propose a wear-leveling algorithm that uses swapping of pages, exchanging heavily written physical pages for lightly written ones. The technique avoids expensive operations, such as searching and accounting, by selecting the pages using a random algorithm. Even though the selection is random, the algorithms achieve a lifetime that is close to the theoretical maximum for large number of pages. The memory system has to be protected from bad program behavior, defined as intentional or unintentional behavior that leads to lifetime reduction. The applications try to undo the wear-leveling by wearing some pages with a faster rate. Our algorithm is naturally resistant since the random selection avoids any pattern that can be explored to cause more skewed wear (instead of an uniform wear).

4.1 SWAPPING-BASED WEAR-LEVELING

Although a reduction in the number of writes will increase the lifetime of a PCM main memory as shown in Chapter 3, the distribution of writes across the PCM devices can still affect lifetime. Our experiments show that applications often have a highly skewed distribution of writes (i.e., AEB writebacks in PMMA). The benchmarks used had 70% of the writes directed to only 1% of pages and 90% of the writes only used 20% of the pages (skewed behavior). The other pages are primarily read only. A PCM memory system that does not counter this behavior quickly wears out the heavily written pages, leading to a lifetime for the overall PCM main memory of a mere months. Since the behavior of the applications is detrimental to the lifetime of the memory, a mechanism is needed to prevent this behavior to affect from affecting the lifetime.

As mentioned in Section 2.1.2, a wear-leveling mechanism can be used to distribute the wear evenly over the pages, with the goal of ensuring that all pages have a similar amount of wear at any time. PMMA is designed to replace a DRAM-only main memory implying that wear-leveling should be achieved without support of the applications, or requiring modifications in the OS or CPU architecture. The wear-leveling mechanism should also be impervious to adversarial behavior of applications (e.g. malware). A *swapping-based wear-leveling* technique with very low overhead algorithm is proposed in this dissertation to support the above two requirements of being self-contained and resistant to attacks.

The skewed behavior of applications determines that a small number of addresses will be the destination of a large percentage of all writes. Assuming the memory subsystem is self-contained, this behavior implies that the association between a CPU physical address and the corresponding PCM physical address needs to be alterable to distribute the writes to the same region of CPU physical addresses over a larger number of PCM physical addresses. The presence of this capability, or alterable translation between CPU and PCM physical addresses, creates the appearance to the CPU that it is still writing to the same address even though different PCM physical locations are used to store the information. A configurable mapping function or a mapping table can be used as a translation mechanism.

A swapping-based wear-leveling exchanges data stored in two PCM physical locations and the corresponding CPU physical to PCM physical mapping. The CPU view of the memory is kept consistent by ensuring that a CPU physical address points to the same data even though it is stored in a different PCM physical address during the system lifetime. A naive implementation of a swap operation requires two reads and two writes to the PCM memory, reading the data in the original location and storing it back in the new location. A swap operation will decrease the memory lifetime by increasing the total number of writes. Hence, reducing the number of swap operations is essential to maximize the memory lifetime.

Although the swap mechanism defined above enables wear-leveling, specific polices can be defined by electing among the few choices of the following components.

- Granularity

  Granularity defines the amount of data that is used as origin or destination for a single swap operation. A larger granularity requires a smaller mapping table but increases the cost of each

swap operation by requiring the exchange of a larger quantity of data. It also reduces the effectiveness of the wear-leveling because the number of possible PCM physical addresses can be allocated to a CPU physical address is smaller, assuming a fixed memory size.

- Frequency

  Frequency defines how frequently swap operations are executed by defining the specific condition that will trigger a swap. Swapping more frequently makes the wear-leveling more effective by reducing the number of writes that can be directed to a single physical location before it is swapped. A high frequency increases overhead and decreases lifetime, since each swap operation adds extraneous reads and writes to the PCM memory. It is desirable to reduce the frequency so memory performance is not significantly affected and lifetime is maximized. In the implementation, a swap operation is executed every 256 to 512 writes to PCM.

- Victim Selection

  Victim selection defines which pages will be subjected to a swap operation. The selection of the pages of the swap operation has a large impact in determining the effectiveness of the wear-leveling. Poor choices of pages to swap, such as two highly written pages or two lightly written pages, do not improve the wear-leveling and can be damaging by just adding writes to the physical locations without improving the wear-leveling.

- Mapping Mechanism

  A mapping mechanism defines how a CPU physical address is translated to a PCM physical address. A number of mapping mechanisms have been proposed, low overhead ones, such as start-gap [40], which uses a configurable function that computes the address using a few parameters, or a more flexible but more expensive mechanism that uses a mapping table (storing the PCM physical address that corresponds to each CPU physical address).

A number of swapping-based wear-leveling algorithms have been proposed (see Section 2.1.2), each with different policies and algorithms for components, such as start-gap [40] that propose a new mapping mechanism or segment swapping [48] that uses a granularity of 1MByte with a victim selection based on a LFW. Our approach is based on a novel mapping algorithm to be described in the next section.

4.2 OUR APPROACH

Our approach is based on the skewed behavior of the applications where a small subset of the CPU physical addresses is the destination of the majority of the writes. Hence, the PCM pages that correspond to these CPU addresses are the pages that need to swapped to avoid over stressing them. This behavior privileges a flexible association between a CPU page to any PCM page, since the highly written CPU pages are not necessarily clustered. A mapping table is used to map a CPU physical page address to the corresponding PCM physical page address allowing any to any mapping. Our scheme, using the components described in the previous Section, is defined by:

- **Victim Selection Mechanism** This mechanism is at the center of our algorithm. Our victim selection mechanism is a very low overhead scheme that avoids high cost operations, such as searching and the need to keep statistics. It is based on the previous observation that the highly written pages are the ones that need swapping, so a PCM physical page that is currently being written is a good candidate and the other PCM physical page should come from the read-only or lightly written pool of pages. Assuming a that the number of highly written CPU pages is small, a random selection of a PCM physical page has very low probability of choosing a page that is a highly written one. Using a PCM page that is the subject of a write to PCM that is in execution allows savings in the number of additional operations necessary for the swap: instead of the 4 operations (two reads and two writes), only two operations are needed, a read an a write.
- **Granularity** Our In our algorithm we use page level (1KByte to 4KBytes) granularity. In Section 4.2.1, it is shown that 4KBytes is enough to achieve the expected lifetime (8 years), but much larger pages have a large negative impact on lifetime. A swap operation will exchange a page worth of data between the two PCM physical pages. A smaller page size would reduce the swap overhead and potentially increase lifetime, by distributing the wear over a larger number of PCM memory cells, on the other hand it increases the mapping overhead by requiring more entries in the mapping table.
- **Mapping Mechanism** A mapping table is used as the mapping mechanism. The table is indexed by the CPU physical page address and each entry contains the corresponding PCM physical page address. The mapping table contains one entry for each PCM physical page, where

only a subset is accessible by the CPU since it is assumed that excess capacity is available at the PCM.

- **Frequency** Our algorithm decreases the overhead of the wear-leveling by swapping infrequently. A swap is performed only when a write to PCM is being executed, our trigger to start a swap operation is based on the number of PCM writes seen since the last swap executed.

Term	Description
M	Number of PCM physical pages
L	Number of CPU physical pages
N	Number of excess capacity pages
L_1	CPU physical page being written
L_2	CPU physical page begin swapped
P_1	PCM physical page that current allocated to L_1
P_2	PCM physical page that current allocated to L_1

Table 3: Parameters and terminology used in Swapping-based Wear-leveling.

Table 3 present the terminology used in the following analysis. Our swap algorithm is formally described in Algorithm 1. The support architecture for mapping table is describe in the Figure 23.

The swap algorithm is based on the expectation that L_1 will have more writes than L_2, so a swap of the PCM physical pages (P_1 and P_2) that supports the CPU physical pages (L_1 and L_2) will equalize the number of writes. The "swap condition" in Algorithm 1 implements the triggering condition (see frequency component). Since a swap operation adds additional writes (both PCM physical pages have to be updated), frequent page swapping will lower memory lifetime. A natural condition to trigger a swap happens when the number of writes to L_i's physical page crosses a threshold. An implementation that can determine when this condition is met requires a counter per PCM physical page. A less precise but much lower cost alternative is to use a single global counter. A swap is done when the total number of writes to the whole PCM crosses a threshold. The loss of precision means that there can be more uneven wear – some pages receive more writes than others, and the chosen page to swap may not be the one with with the highest number of

Algorithm 1 Swapping-based Wear-Leveling Algorithm

{Writeback of CPU physical page L_1 onto a PCM physical page P_1}

if $SwapCondition() == FALSE$ **then**

write L_1 data on P_1

else

$L_2, P_2 = SelectSwapTargetPage()$

$P_1 \Leftarrow P_2$ {Copy data from P_2 to P_1}

$Map[L_2] \Leftarrow P_1$ {L_2 points to P_1}

$Map[L_1] \Leftarrow P_2$ {L_1 points to P_2}

write L_1 data on P_2

end if

End.

{Swapping Condition}

Function SwapCondition()

$NumberWrites++$

if $NumberWrites \geq Threshold$ **then**

Return True {Swap}

else

Return False {Proceed with write}

end if

End SwapCondition()

{Select the target CPU physical page L_2 and PCM physical page P_2}

Function SelectSwapTargetPage()

$P_2 \Leftarrow Random(0..$***Number of PCM pages***$)$

$L_2 \Leftarrow$ CPU Physical page mapped to P_2

Return L_2 and P_2

End SelectSwapTargetPage()

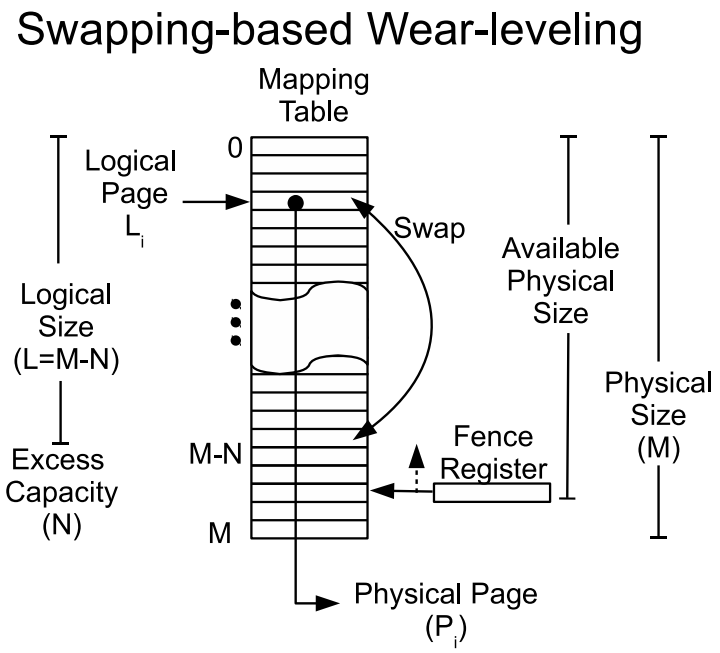

Figure 23: Architectural support for Swapping-based Wear-leveling.

writes. However, it is expected that the difference in wear will "even out", as the number of writes increases. The global counter has the desirable property that highly written CPU physical pages will have higher probability of being swapped, which happens in most applications due to their skewed write distributions.

Our swap algorithm implements victim selection by selecting the PCM physical page being written as one of the pages to be swapped since this reduces the swap operation overhead by half (two operations instead of four). The use of the current PCM physical page being written also has the benefit that the corresponding CPU physical page has a high probability of being a highly written page, and it is one that it is desirable to swap. An adequate selection of a target PCM physical page (P_2) as the destination of the swap is crucial to achieve uniform wear. A bad choice would trade two physical pages that have a similar number of writes. The best choice intuitively is the least-frequently written (LFW) PCM physical page. However, finding the LFW page is extremely expensive due to the large number of physical pages that have to be searched to find the minimum count. Furthermore, LFW requires a counter per page and that the counter be updated at each write to the physical page. A much simpler solution picks a random physical page as the

target. The probability of choosing a highly written target page is low due to the large number of physical pages and the small number of highly written pages. The random algorithm should ultimately approximate LFW for a large number of writes, since the probability of choosing a highly written page is small and all pages should be chosen in one point of time to be allocated to a highly written page.

Figure 23 presents the architecture that implements our swap wear-leveling. The swap-based wear-leveling algorithm uses a mapping table to translate between a CPU physical page address and PCM physical page address (See Section 3.1.2 for PMMA implementation). It is indexed by CPU physical page number; each table entry contains the PCM physical page address for a CPU physical page. In a swap, two entries are chosen according to the victim selection mechanism and both the entries in the mapping table and the data in the PCM physical pages are exchanged, executing the swap. The architecture in Figures 23 supports the existence of excess capacity on the PCM memory and allows graceful degradation by retiring pages. The fence register is used to implement retirement by restricting the choice of pages to that can be used. If a write failure happens, the failed PCM physical page is discarded and the PCM physical page that is currently allocated to the entry that is pointed by the fence register is selected to be used by the CPU physical page, with the fence register being decremented. Note that no information is lost because the page pointed by the fence register is not currently used by a CPU physical page. Decrementing the fence register retires the failed page by not allowing it to be used in a swap or write failure operation.

The mapping table present on the architecture, could be portrayed as creating excessive size and time overhead but its size is small relative to PCM size. For example, a 4GBytes PCM with 2KBytes page size needs only 6MBytes (using 3 bytes per entry). In PMMA (see Section 3.1.2), the mapping table is kept in both PCM and DRAM, reducing the time overhead. The existence of a copy in the the PCM allows it to be persistent but creates the problem of endurance of the cells that support the mapping table. A write-failure with page replacement can be used to guarantee the reliability of the mapping table. The entries are individually updated, since both memories support byte-addressing, and an update will only be required when a swap is executed for this specific CPU physical page. This is a fraction of the number of writes to the PCM physical pages.

The swap-based wear-leveling algorithm and the supporting architecture allows the use of small page sizes (smaller than 4KBytes). The primary reason is the use of a victim selection mech-

anism that has constant overhead, independent on the number of pages. A small page size reduces overhead since less data needs to be copied and creates a larger pool of possible destinations which to distribute writes. A small page size also has a higher probability of avoiding specific application behaviors that would impact lifetime. Larger pages would map a smaller number of CPU physical addresses to each PCM memory cell, increasing the variation on the number of writes per address. The disadvantage of a small size is the need to map a larger number of pages, since each CPU physical page can map to any physical page. The need for small page sizes can be eliminated if the applications does have a lightly skewed intra-page distribution of writes. The benchmarks tested have a highly skewed inter-page distribution but a very small intra-page variation on the number of writes. In PMMA, the wear-leveling granularity should be equal or larger than the largest sub-page used, this guarantees that a single access to the mapping table is sufficient to access all the data, otherwise multiple access to the mapping table would be necessary. Figure 24 shows that the variation on the number of writes directed to each of the cache lines (64 Bytes) in all pages (2KBytes page size) is small, differing by less than 3%. Figure 24 is constructed by summing all writes directed to a single cache line independent of the specific PCM physical page the write as directed to. The small difference removes the need for a smaller swapping page size or a specific intra-page wear-leveling algorithm.

Our implementation of the swap algorithm uses a global write counter for the swap condition and randomly selects the target page. The global counter is a decrement counter initially loaded by a random value and a swap is executed when the counter underflows. A fixed threshold would create a pattern that could be explored by an adversarial application to prevent wear-leveling to occur. We use random page selection due to its low cost – it avoids the need for page usage counters.

4.2.1 Simulation Set-up

To understand the effectiveness of our techniques, we evaluated their impact on PCM lifetime. We used the same setup described in Section 3.2 with PMMA as the PCM main memory architecture. We configured PMMA with a 4GBytes address space, a 2KBytes page, and a 256Bytes write sub-page because this page/sub-page configuration had the best performance average for

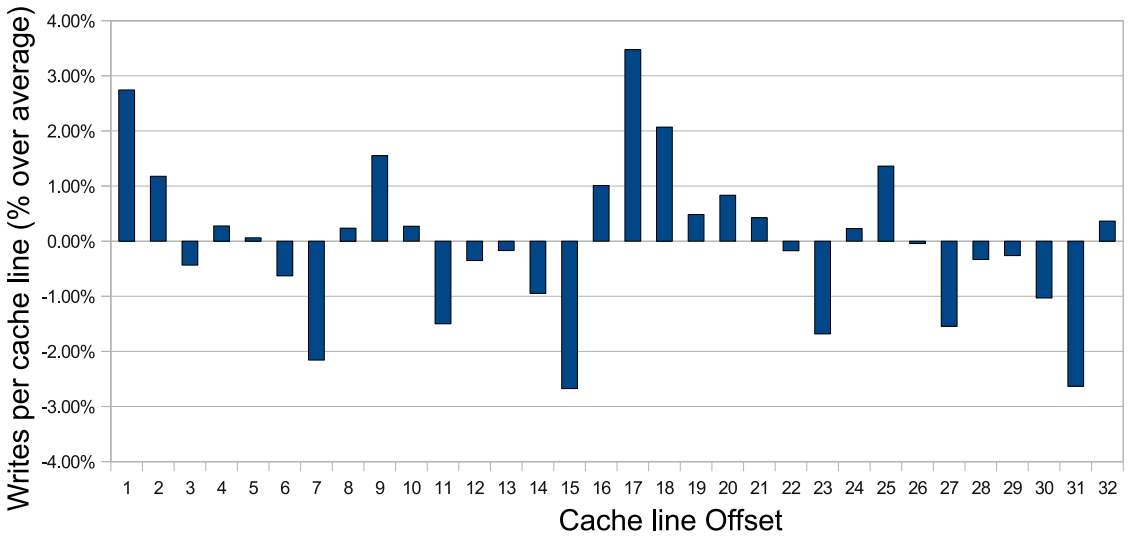

Figure 24: Variation on the number of writes directed to each cache line aggregated for all writes .

our benchmarks. The DRAM page cache (AEB) uses 224MBytes. The remaining 32MBytes of DRAM holds the mapping table. The AEB is mapped as a 14-way set associative with each set having 16MBytes.

The simulator used in Section 3.2 is extremely accurate, modeling the system all the way down to the bus and device event level. This simulator was used to obtain performance and energy results for the wear-leveling algorithms. To measure lifetime, since a very large number of writes is necessary to damage a page when our techniques are modeled (i.e., a 10^7 write limitation, when multiplied by a few million pages would need almost 10^{13} memory requests), we used a separate fast behavioral simulator. Even with a real system it would take 8 years (expected lifetime) to damage enough pages. A trace of writes to PCM from the PMMA simulator was used as input to the endurance management simulator. The trace can be repeated or mixed with other traces to get the required number of writes to damage a page. A single trace obtained from a PMMA run of each benchmark is orders of magnitude smaller than necessary: we used 2 billion PMMA memory access for each benchmark (some benchmarks ended before the that number could be achieved), but that number is still 3 to 5 orders of magnitude smaller than the necessary to simulate a server lifetime.

The same benchmarks used in Section 3.2 were selected since they stress endurance by having a large number of AEB misses. The benchmarks are: Canneal and Facesim from PARSEC; MCF, GCC, Bwaves and bzip2 from SPECcpu2006; and SPECjbb2005. A mix of SPECcpu2006 applications, composed of MCF, GCC, Bwaves and bzip2, were executed together to obtain a large memory footprint and utilization. Each benchmark was run in Simics for 2Billion requests to main memory. The collected memory traces were used as input to the PMMA simulator that produced a PCM write trace that was applied to the endurance management simulator until a page was damaged (10^7 writes to any bit on a page).

A baseline is needed to identify the impact of our swap-based algorithm when applied in non-ideal environments (real traces). The chosen baseline is a swap-based algorithm that has a high overhead but has a guarantee bound on the variation on the number of writes per page. Three conditions make this algorithm impractical, the need for one counter per page, an expensive search algorithm (for the LFW) and the susceptibility to adversarial behavior since the swap operation can be predicted. This algorithm uses a counter per page to accumulate the number of writes and implements a swapping condition based on each counter passing a threshold. The victim selection selects the LFW page to be the page to be swapped. This algorithm is called CT-LFW (counter per page swapping condition, LFW victim selection). We compare the reference algorithm with relaxed versions that use less overhead modifications up to our global counter (GC) and random replacement selection algorithms. The global counter version removes a counter per page to trigger a swap, using instead a global counter. The random victim selection randomly chooses the page to swap, instead of searching for the LFW.

4.2.2 Wear-Leveling Experimental Results

Figure 25 shows the lifetime achieved with different swap algorithms. The reference algorithm, CT256-LFW, implements a counter-per-page with a threshold of 256 (a swap every 256 writes to a page) and LFW as the victim selection mechanism. In the graph, the frequency is changed from 1 swap every 256 writes to a page to 1 swap every 512 writes to a page. The lifetime was computed by running each benchmark 500 times and recording the number of writes per physical page. The number of runs necessary to damage each page (10^7 writes) is computed and the minimum overall

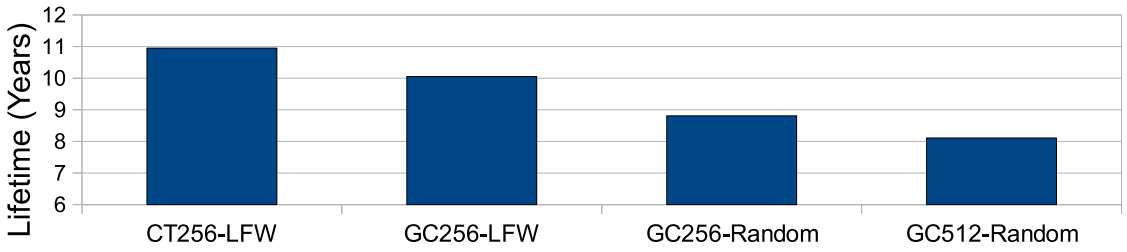

Figure 25: Impact of wear-leveling on lifetime

is the predicted lifetime. The measured lifetime is a conservative estimate since it assumes that the distribution of writes per page will be the same for all runs. The system does not change the mapping table between runs so the same set of highly written pages will continue to be directed to the same set of PCM physical pages in the last run. This will overestimate the number of writes per page by removing the randomization that an OS would impose when reloading a process (in the data pages).

The global counter and random victim selection mechanisms are expected to have better behavior for a larger number of writes, which will tend to make the distribution of number of writes per page more uniform as more runs are executed. The algorithms count each write to a write sub-page as one write to PCM. CT256-LFW, i.e. counter-per-page, 1 swap per 256 writes to a page, least-frequently written victim selection, is an expensive algorithm that uses one counter per page and requires an expensive search to find the LFW page at each swap operation. The search for LFW is expensive because of the large number of pages involved, a more complex data structure like binary or red-black trees can be used to reduce the cost of search with the trade-off of increasing the cost of each write. Other algorithms have lower overhead.

Figure 25 shows the reduction in lifetime that is caused by the change in the frequency and victim selection mechanism. A change from a local per-page trigger (CT256-LFW) to a global trigger (GC256-LFW) in the swap operation reduces lifetime by 8%. This reduction in lifetime is a result of the larger difference between the number of writes in pages causing the maximum number of writes to be achieved faster. The global counter does not guarantee a swap when the number of writes to a *particular* physical page reaches a threshold. Next, a random replacement mechanism is used instead of a LFW mechanism for victim selection. GC256-Random uses a

global counter with random replacement. It reduces lifetime by 12.4% compared to GC256-LFW. The difference in the number of writes per pages is accentuated by the random replacement; the difference now is unbounded. Even though the random replacement mechanism makes no effort to correct imbalances, only 12.4% of lifetime is lost for complete removal of the need of keep counters in the pages. A total of 20% reduction in lifetime when GC256-LFW is compared to CT256-LFW. The swap overhead of GC256-Random is only an additional 3% more writes due to swap operation. When the overhead is reduced to 1.5% by setting the frequency to a swap every 512 writes to PCM, the decrease of lifetime is 25.4% when compared to the reference (GC256-LFW). In Figure 25 all algorithms have at least eight years of lifetime, but GC256-Random and GC512-Random have the lowest implementation cost and overhead.

The use of large 1MByte swapping pages, as in [48], has a negative impact on lifetime. In our experiments, CT256-LFW with 1MByte pages has a lifetime of 7.5 years. This value is 31% smaller than the lifetime obtained with 2KByte pages (CT256-LFW). The overhead is caused by each swap requiring a transfer of 1Mbyte and not 2KBytes, so each swap is equivalent to 512 swaps of 2KBytes pages and the optimization of removing the additional operations (one write and one read) is not possible, so a total of 2MBytes have to be read and written in each swap. Our GC512-Random algorithm gets 6.3% better lifetime with significant lower cost than swapping on 1MByte pages.

4.2.3 Comparison with other techniques

The segment swapping algorithm is proposed in [48]. It uses a large 1MByte segment to reduce overhead since it uses an expensive search for the least-frequently written segment to swap. The large size of the segment requires additional intra-segment wear-leveling and the use of counters to store the number of writes directed to each segment. Our swap-based algorithm can afford to use a much smaller page by taking advantage of the low overhead of the victim-selection algorithm used.

The start-gap algorithm presented in [40] (see 13) requires less architectural support than our swap-based algorithm, as it does not require a mapping table and only uses two counters but demands higher energy and achieves a lower lifetime. Start-gap requires excess capacity in the PCM

memory but does not allow graceful degradation to occur. Any page that fails has to be replaced transparently by an additional algorithm since the pool cannot be reduced without modifications to the algorithm. Start-gap translates address using a function with two parameters, a start counter and a gap counter. All the address computations are done wrapping around the PCM physical size (modulus operation). The start counter marks the PCM physical page that is currently holding the date from the CPU physical page 0 and gap counter points to the beginning of gap (region that contains the excess capacity). The computation of the PCM physical page address depends on the CPU physical page address being below [gap - start] or above it. If below, CPU physical pages between [0 ... gap - start - 1], the start counter is added to the CPU physical page address to get the PCM physical pages, i.e. [start ... gap-1]. If above, values between [gap - start ... Max CPU page address - 1], the start counter and the size of the gap (excess capacity) is added to the CPU physical page address to get the corresponding PCM physical page address. The wear-leveling operation is executed by copying the data at the PCM physical page pointed by gap counter + gap size to the PCM physical page pointed by the gap counter and incrementing the gap counter. The start counter is moved whenever the gap counter passes by it. Looking at a specific CPU physical page, it will only use a single PCM physical page until the gap passes by it and it is copied to a different location. The consequence is that any writes to this particular CPU physical page will be directed to the same PCM physical page until the next time the gap counter pass by this CPU physical page. Assuming a skewed write distribution in the application, e.g., 20% of the pages is the destination of 70% of the writes (as measured in our benchmarks, see Section 4.1), a single pass of the start-gap algorithm will copy the whole PCM memory to a different location but only 20% of it had large amount of writes. To achieve wear-leveling, all PCM physical pages have the same amount of writes, the highly written CPU physical pages have to be mapped to all PCM physical pages. Assuming an even distribution of the highly written CPU physical pages over the address space (one at each 5 pages), 5 passes have to be executed requiring the memory to be copied 5 times (a pass is represented by a gap passing by a single PCM physical location). Our algorithm swaps only the highly written pages, read-only pages are never the origin of a swap and only can be a destination. Copying the 20% once over the whole space would achieve a level of wear-leveling similar to a 5 passes by allowing our algorithm to use less operations to achieve the same lifetime, reducing energy and increasing performance. Our algorithm will be more efficient than start-gap

as more skewed the write distribution of the application is. A smaller page size would also benefit our algorithm assuming that the write distribution is the same per page (only a subset of the page is written), since the skewness of the application would increase.

5.0 MODELING USAGE OF EXCESS CAPACITY

Wear-prone memories, such as PCM or NAND Flash, normally have more capacity than is visible to the higher layers (CPU). This excess capacity is used to increase reliability and extend lifetime of the memory and without excess capacity, any portion of the memory with low endurance would dictate the whole memory lifetime. Process variation in wear-prone memories affects the endurance by changing characteristics that are responsible for the failure mechanisms. A constant model for Endurance is now a too simple model and a more comprehensive model assumes that endurance will follow some statistical model. The analysis of wear-leveling techniques in Section 2.1.2 and in the previous Chapter 4 leave one question unanswered: how to use the excess capacity provided by the wear-prone memories in the presence of process variation. The following sections answer that question assuming that we have some knowledge of the endurance statistical distribution.

5.1 ANALYSIS OF ENDURANCE ALGORITHMS WITH PROCESS VARIATION

The wear-leveling algorithms shown on Section 2.1.2 and on previous Chapter 4), partitions the memory into fixed size pages and uses the excess capacity to extend the memory lifetime. The memory system is considered failed whenever the visible area cannot be stored in the physical memory, in other words, the excess capacity was exhausted.

The excess capacity can be used in two different schemes, Physical Capacity Degradation (PCD) and Physical Sparing (PS). Figure 26 shows the two schemes. Physical Capacity Degradation assumes that a wear-leveling mechanism will use the whole physical memory (non-failed pages), removing a page whenever it fails. Physical Sparing, otherwise, assumes that the wear-

leveling mechanism uses only a subset of the physical memory that correspond to the visible memory (visible to the CPU) and whenever a page fails a replacement is used from the excess capacity. As PCD, PS will fail whenever the excess capacity is exhausted.

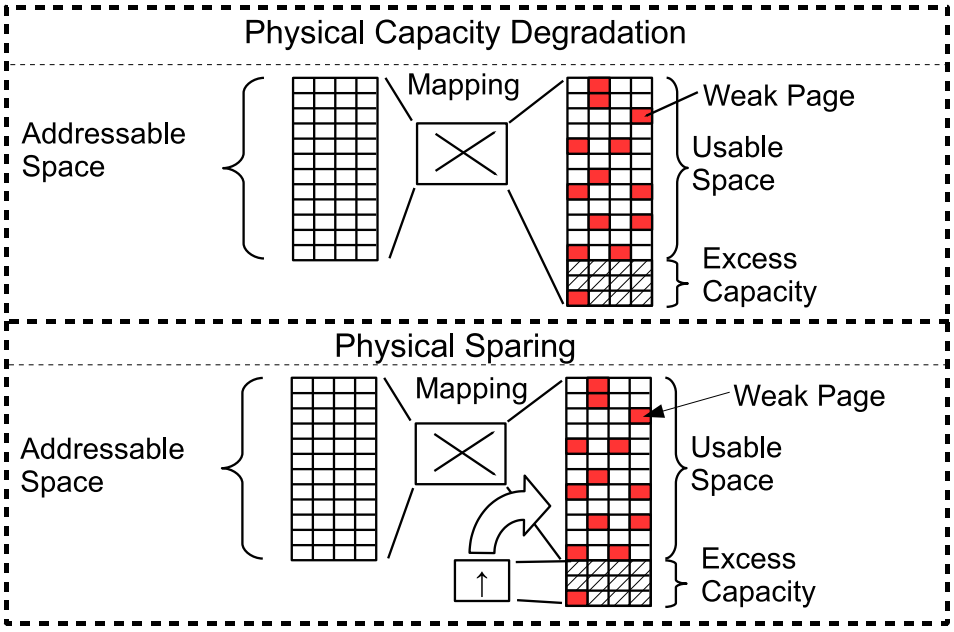

Figure 26: Model of the endurance algorithms

In the model, the memory subsystem has M physical pages, each of fixed size. Akin to many storage systems, the set of pages is logically partitioned in two areas: a visible *addressable space* of L pages and a reserved *excess capacity area* of N pages, as seen in Figure 27. In other words, $L = M - N$ and we assume the memory subsystem fails when there are less than L undamaged physical pages. Note that the addressable user space has the same physical size L independent of the endurance algorithm (PCD or PS) and therefore the performance of user processes is the same for both endurance algorithms. A hybrid PS + PCD algorithm, which uses PS for until a subset of the excess capacity is used and PCD until the rest of the excess capacity is exhausted, can be modeled as two separate process. One is a PS with parameters M,N',$L' = M - N'$ and the other is a PCD with $M'' = M - N'$,$N'' = N - N'$, $L = L$.

We use an idealized wear-leveling scheme, which distributes the writes among all pages used (the wear-leveling algorithm is ideal in the sense that all pages have had the same number of writes at any instant of time), to be able to analyze behavior of PCD and PS. To distribute writes among the

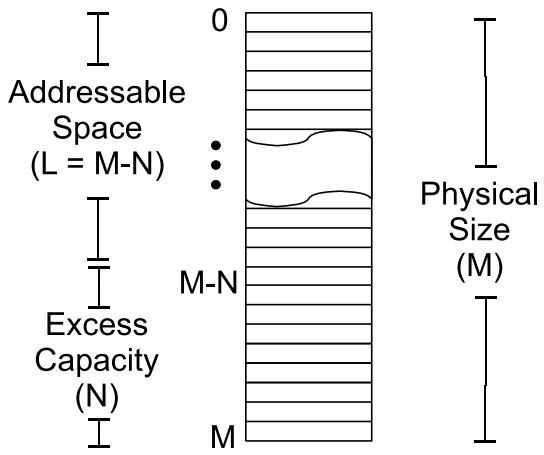

Figure 27: Memory Model with excess capacity.

pages, a mapping is necessary which translates from a CPU physical page address to the memory physical page address, as shown in Figure 26. This mapping depends on the implementation of the wear-leveling algorithm [25, 47, 48], as described in the Section 4.1. The *lifetime* of a set of M pages will be measured as the number of writes that the memory supports until its capacity is reduced below L.

Term	Description
M	Number of physical pages
L	Number of addressable pages
N	Number of excess capacity pages
$LPCD(M,N)$	Number writes before failure for PCD
$LPS(M,N)$	Number writes before failure for PS

Table 4: Parameters and terminology used in the analysis of endurance distributions.

Table 4 summarizes the key parameters and terminology used in the analysis of the different endurance techniques and distributions.

In each subsection below, we compute the lifetime of a memory for four different models of process variation, namely *constant, bimodal, linear* and *normal*, for each of the two endurance algorithms (PCD and PS). We present the models by order of complexity.

5.1.1 The Constant Model

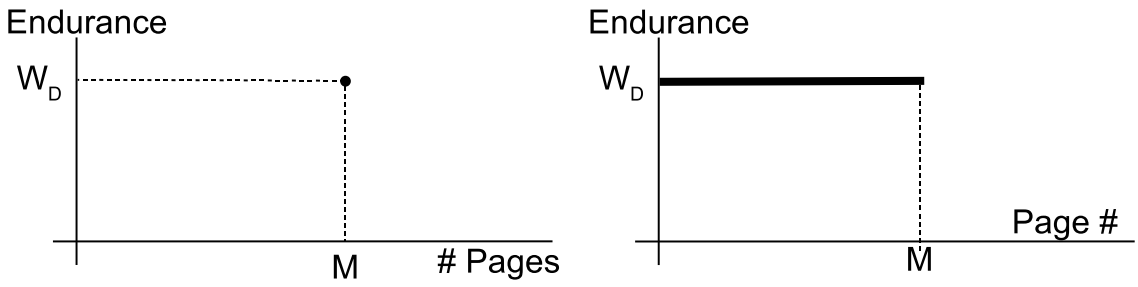

Figure 28: Page Endurance distribution for a constant model.

In the constant model, we assume that all pages have the same endurance, W_D, as shown in Figure 28. In the PCD case, all M pages receive the same number of writes due to the underlying wear-leveling algorithm. This implies that the lifetime for a memory with M pages is: $LPCD(M,N)=W_D \cdot M$. In the PS case, a page in the addressable space will be damaged after W_D writes. Because there are only $M-N$ pages to distribute the writes, $M-N$ pages will be damaged at the same time. Since $M>2N$ implies $M-N>N$, there will not be enough spares to replace all the $M-N$ damaged pages. Hence, the lifetime of PS in this case is: $LPS(M,N)=W_D \cdot (M-N)$. Comparing the lifetimes, $LPS(M,N)<LPCD(M,N)$, and thus, it is clear that PCD leads to a longer lifetime than PS for any number of spares under the constant model.

5.1.2 The Bimodal Endurance Model

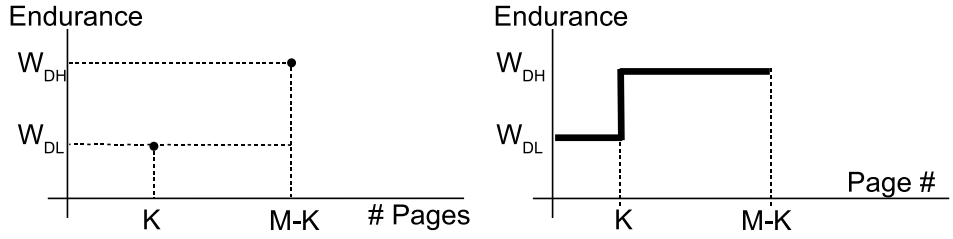

Figure 29: Page Endurance distribution for a bimodal model.

The bimodal model, as shown in Figure 29, assumes that pages are divided into two sets: a low endurance set with K pages that has an endurance of W_{DL} per page and a high endurance set with $M-K$ pages that has an endurance of W_{DH} per page. It is assumed that $W_{DL} \ll W_{DH}$.

In the PDC case, the K weak pages are damaged and retired after $W_{DL}{\cdot}M$ writes (since the memory starts with M pages). There are two cases to consider:

- $K{\leq}N$: For N pages to be damaged, some strong pages have to be damaged since only K weak pages exist. This allows an additional $(W_{DH}{-}W_{DL}){\cdot}(M{-}K)$ writes to be applied to the memory. Thus,

$$LPCD(M,N){=}W_{DL}{\cdot}K{+}W_{DH}{\cdot}(M{-}K) \text{ if } K{\leq}N \tag{5.1}$$

- $K{>}N$: After $W_{DL}{\cdot}M$ writes, the K weak pages will be damaged and the number of available pages will be less than $M{-}N$, thus leading to system failure. Hence,

$$LPCD(M,N){=}W_{DL}{\cdot}M \quad \text{if } K{>}N \tag{5.2}$$

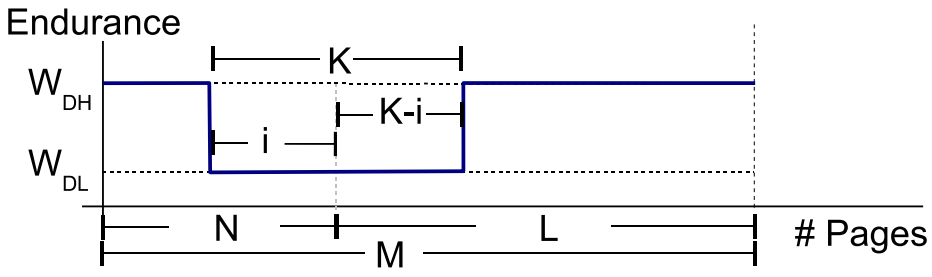

Figure 30: Spare and addressable page endurance distribution in a bimodal model.

In the PS case, there are three cases to be analyzed:

- $K{\leq}N$: When all the K weak pages are in the addressable $M{-}N$ pages, they will be replaced when they reach their endurance limit (W_{DL}). The lifetime will be determined by the endurance of the strong pages, W_{DH}, and the size of the addressable space ($M{-}N$). Hence,

$$LPS(M,N){=}W_{DH}{\cdot}(M{-}N) \text{ if } K{\leq}N \tag{5.3}$$

- $K{>}2N$: When there are at least $N{+}1$ weak pages in the addressable space, these weak pages will be damaged after $W_{DL}{\cdot}(M{-}N)$ writes and the number of spare pages available, N, will not be enough to replace them. Thus,

$$LPS(M,N){=}W_{DL}{\cdot}(M{-}N) \quad \text{if } K{>}2N \tag{5.4}$$

- $N<K\leq 2N$: Among the K weak pages, let i be the number of *weak spare pages* and $K-i$ be the number of *weak used pages*, as shown in Figure 30. The $K-i$ weak used pages located in the addressable space will be damaged first since they have the lowest endurance and are being constantly used. Two cases may occur:
  - $K-i>N$: If $i<K-N$, the number of weak used pages is larger than the number of spare pages. This implies that when the weak used pages are damaged, the memory will fail since there will not be enough spares to replace all the damaged pages. The lifetime is then $LPS(M,N)=W_{DL}\cdot(M-N)$.

    $K-i\leq N$: In this case, there are at most N weak used pages. These weak used pages will be damaged after $W_{DL}\cdot(M-N)$ writes to the addressable space and will be replaced by spares, extending the lifetime of the addressable space by an additional $W_{DL}\cdot(M-N)$ writes. Given that $K>N$, then $i>0$, and some of the newly commissioned spares will be weak and will be damaged after an additional $W_{DL}\cdot(M-N)$ writes. If $K-i=N$, then no more spares will be available and the memory will fail at this point (here we assume that $2\cdot W_{DL}\ll W_{DH}$, that is, the weak spare pages will be damaged before the strong pages). However, if $K-i<N$, some spares will still be available after the first replacement round and the lifetime will be extended by an additional $W_{DL}\cdot(M-N)$ writes with any new replacement round for which spares will still be available. The lifetime is then $LPS(M,N)\geq 2W_{DL}\cdot(M-N)$, with the equality achieved when only the first replacement round is possible.

Summarizing, the lifetime in the region delimited by $N<K\leq 2N$ is:

$$LPS(M,N)\begin{cases} =W_{DL}\cdot(M-N) \text{ if } i<K-N \\ \geq 2W_{DL}\cdot(M-N) \text{ if } i\geq K-N \end{cases} \tag{5.5}$$

It is important to note that the lifetimes for PCD and PS depend on $\frac{K}{N}$, the ratio of weak pages to spare pages.

Comparing the lifetime of PCD and PS:

PCD has a higher lifetime than PS when (a) the number of weak cells is larger than the number of spares (i.e., when $\frac{K}{N}<1$); compare Equations (5.1) for PCD and (5.3) for PS, or (b) the number of weak cells is much smaller than the number of spares (i.e., $\frac{K}{N}>2$); compare Equations (5.2) for PCD and (5.4) for PS.

In the case of $N<K\leq 2N$, the result depends on Equation (5.5) since the lifetime of PCD, as clear from Equation (5.2), is constant in this region.

The lifetime of the PS algorithm depends on the number of weak pages in the spare area. The probability of having x weak pages among the spares, $Pr(i = x)$, follows a hypergeometric distribution. Equation (5.2), PS will have a higher lifetime than PCD when $i\geq K-N$, which occurs with probability:

$$Pr\begin{pmatrix} LPS(M,N)> \\ LPCD(M,N) \end{pmatrix} = \sum_{x=K-N}^{N} Pr(i=x) \tag{5.6}$$

The average, $E[i]=\sum_{x=0}^{N} x\cdot Pr(x)$, of a hypergeometric distribution is $E[i]=\frac{NK}{M}$.

Since $LPS(M,N)>LPCD(M,N)$ only if $i>K-N$, the probability can be estimated by:

$$\sum_{x=K-N}^{N} Pr(i=x)\geq 0.5 \quad \text{if } \frac{NK}{M}\geq K-N \tag{5.7}$$

By changing the condition in Equation (5.7) to $\frac{N}{M}=1-1/\frac{K}{N}$, Equation (5.6) can be rewritten as a function of the ratios of K, M, and N, showing that $LPS>LPCD$ more than 50% of the time:

$$Pr\begin{pmatrix} LPS(M,N)> \\ LPCD(M,N) \end{pmatrix} \begin{cases} \geq 0.5 \text{ if } \frac{N}{M}\geq 1-\frac{N}{K} \\ <0.5 \text{ if } \frac{N}{M}<1-\frac{N}{K} \end{cases} \tag{5.8}$$

Note that Equation (5.8) depends on the ratios $\frac{N}{M}$ and $\frac{K}{N}$ and not on the specific value of M. graceful degradation depends only on the percentage of spares and fraction of weak over spares pages. Figure 31 shows the curve $\frac{N}{M}=1-1/\frac{K}{N}$ in the region $1<\frac{K}{N}\leq 2$, that is, when $N<K\leq 2N$. The curve $\frac{N}{M}=1-1/\frac{K}{N}$ creates two regions. In the region above the curve, it is more probable that the lifetime of PS will be higher than the lifetime of PCD (points 1, 2, 3, 5 and 8 in Figure 31). The region below the curve will have the opposite behavior, with higher lifetime when using PCD (points 4, 6 and 7 in Figure 31).

Note that for the hypergeometric distribution the standard deviation, σ, is at most the square root of the average, that is, $\sigma\leq\sqrt{\frac{NK}{M}}$. The hypergeometric distribution can be approximated by a normal distribution for which the probability that $i<E[i]-2\sigma$ or $i>E[i]+2\sigma$ is less than 2.5%. This allows Equation (5.8) to be rewritten as:

$$Pr\begin{pmatrix} LPS(M,N)> \\ LPCD(M,N) \end{pmatrix} \begin{cases} \geq 0.975 \text{ if } \frac{N}{M}\geq 1-\frac{N}{K}+\frac{2\sigma}{K} \\ \leq 0.025 \text{ if } \frac{N}{M}<1-\frac{N}{K}-\frac{2\sigma}{K} \end{cases} \tag{5.9}$$

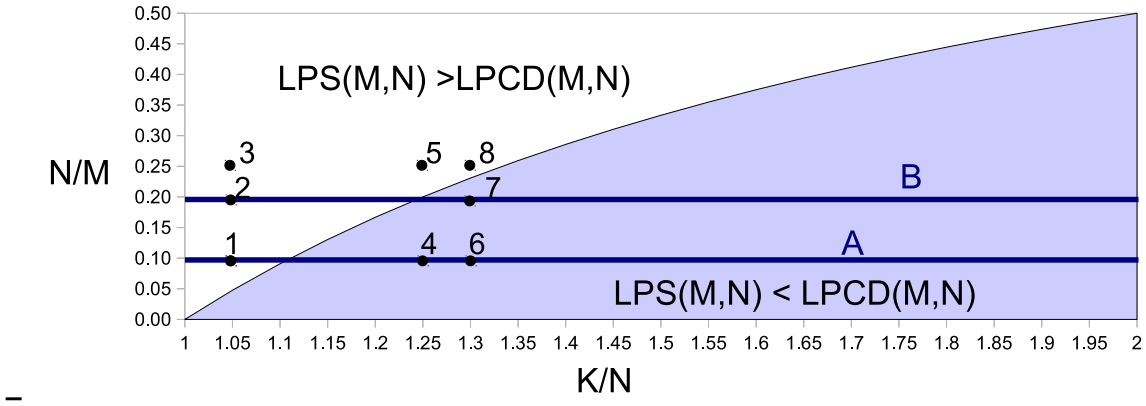

Figure 31: Regions where PS increases lifetime.

Given that $\sigma=\sqrt{\frac{NK}{M}}$, we can conclude that $\frac{2\sigma}{K}=2\sqrt{\frac{N}{MK}}$ implies that $\frac{2\sigma}{K}\leq\frac{2}{\sqrt{M}}$ since $\frac{N}{K}\leq 1$.

Equation (5.9) indicates that there is a very narrow band (of width proportional to $\frac{2}{\sqrt{M}}$) around the curve $\frac{N}{M}=1-1/\frac{K}{N}$ of Figure 31 within which $Pr(LPS(M,N)>LPCD(M,N))$ is between 0.025 and 0.975. Above this band the lifetime of PS is longer than the lifetime of PCD (with very high probability) and below this band, the lifetime of PS is shorter than the lifetime of PCD (with a very high probability).

To examine the validity of our analysis, we plot in Figure 32 the exact probability given by Equation (5.6), for $M = 2000$ and $M = 5000$ at $\frac{N}{M}=0.1$ and $\frac{N}{M}=0.2$. This corresponds to the cuts A and B in Figure 31. Clearly the probability of $LPS(M,N)>LPCD(M,N)$ goes from 100% to 0% very sharply near $\frac{N}{M}=1-1/\frac{K}{N}$, which shows the approximation given by Equation (5.9) is true for large values of M, which is what happens in real life.

5.1.3 The Linear Endurance Model

The linear model, as shown in Figure 33, is a tractable approximation of the normal distribution assuming that cells have lifetime linearly distributed between W_{DL} and W_{DH}. We model it as the lifetime of page i as $W_i=W_{DL} + R\cdot i$ with $R=\frac{(W_{DH}-W_{DL})}{M}$.

The lifetime under PCD is determined by the first $N + 1$ pages that wear out. We approximate this by N. Let $W_{DN}=W_{DL}+R\cdot N$ be the point on the endurance curve where N pages have failed.

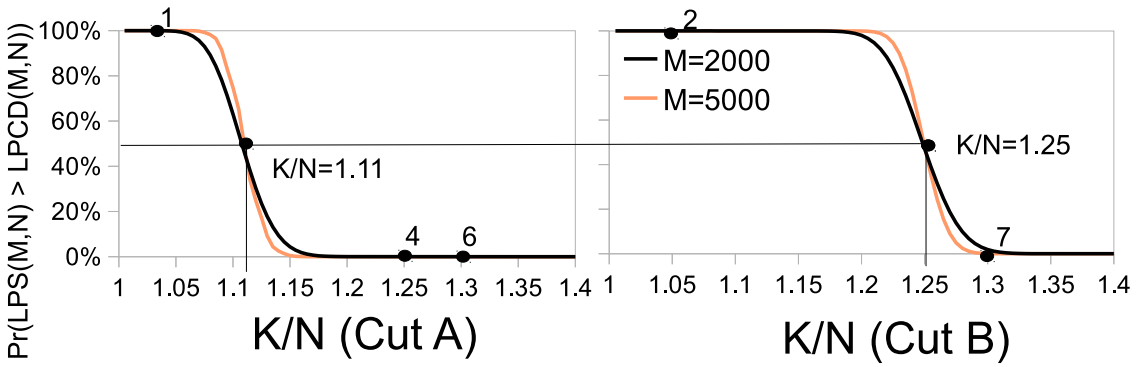

Figure 32: Probability of sparing having a higher lifetime

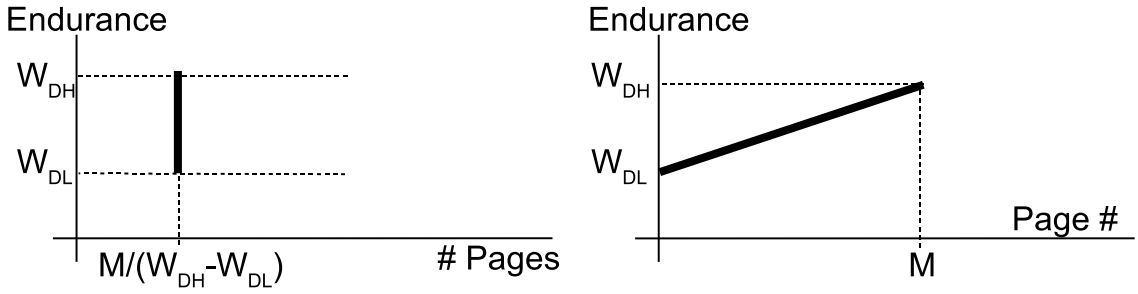

Figure 33: Page Endurance distribution for a linear model.

We can compute the area under the endurance curve up to N ($N{\cdot}W_{DN}/2$) and between N and M ($W_{DN}{\cdot}(M{-}N)$). The sum of these two areas is $LPCD(M,N){=}W_{DL}{\cdot}M{+}R{\cdot}N(M{-}N/2)$.

PS lifetime is determined by the endurance (W_{Dj}) of the last page to die (jth page), that is $LPS(M,N){=}W_{Dj}{\cdot}(M{-}N)$. It is apparent that $j{\geq}N$, since the worst case is when all the spares have endurance higher than W_{DN}. The maximum value of j is $N{+}\frac{N^2}{M-N}$. $\frac{N^2}{M-N}$ is the expected number of pages with $W_D{\leq}W_{DN}$. The impact of larger values of N on lifetime is minimal because it increases J but also decreases the number of pages in the addressable space. Large values of R would in theory benefit PS but the probability that a page needs to be replaced more than once before the memory subsystem dies increases, reducing the number of addressable pages that can be replaced. The end result is a lower lifetime than predicted by the maximum j. Using Monte Carlo simulations, we determined that the lifetime of PCD and PS for the linear endurance model are very similar, with a maximum of 3% difference.

5.1.4 The Normal Endurance Model

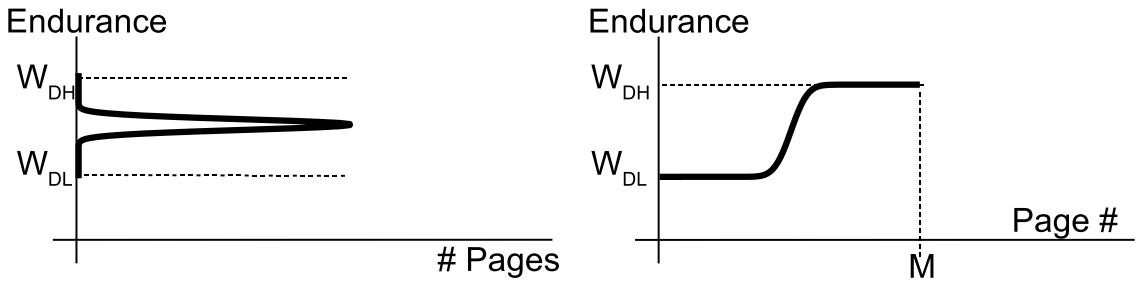

Figure 34: Page Endurance distribution for a normal model.

The normal model can be approximated by a constant model if the standard deviation is small compared to the average. Figure 33 show the endurance distribution for this model. The linear model is a good approximation if the normal model has a large standard deviation and the number of spares is small (we only are interested in the pages with a low lifetime). In cases that the approximations are not applicable, numerical simulations show that PCD and PS present a very similar lifetime under the normal endurance model with PCD always winning by less than 5%.

5.1.5 Generalization for Other Distributions

The models presented can be used as an approximation of a more complex distribution if the distribution of the weakest $2N + 1$ pages is similar to one of models.

Theorem 1. *As defined in section 5.1, the model is based on a memory with M pages with N reserved as spares. Let the pages be ordered by endurance so the page 0 is the weakest and page M is the strongest.*

Lemma 2. *The lifetime of a Physical Graceful Degradation algorithm is independent of specific distribution of the endurance of the strongest $M - N$ pages.*

Proof. Assuming ideal wear-leveling, all the pages will have the same number of writes at any instant of time. The first page to be damaged will be page 0 since it is the weakest page. The subsequent one being the page 1 and so forth until N pages are damaged. At this point the memory has no more additional pages to retire and will be considered damaged. The specific value of the endurance of the pages $M - N$ is not important as long as they are larger than the endurance of the page N. □

Lemma 3. *The lifetime of a Physical Sparing algorithm that follows a uniform model for the weakest $2N$ pages and any specific distribution of endurance for the stronger ones is equal or larger than the lifetime of the sparing algorithm under a uniform model.*

Proof. A generalized uniform model has the weakest $2N$ cells with the same endurance. Even if N of the weakest cells are reserved as spares, there will be $N + 1$ weak pages which will be damaged at the same time causing the memory to fail. □

Lemma 4. *The lifetime of a Physical Sparing algorithm that follows a bimodal model for the weakest $2N$ pages and any specific distribution of endurance for the stronger ones is equal or larger than the lifetime of the sparing algorithm under a bimodal model.*

Proof. The section 5.1.2 analysis can be reused since it depends only on the number of weak pages and their endurance. □

Lemma 5. *The lifetime of a Physical Sparing algorithm that follows a linear model for the weakest $2N$ pages and any specific distribution of endurance for the stronger ones is equal or larger than the lifetime of the sparing algorithm under a linear model.*

Proof. The probability of a page requiring more than one replacement is not affected, since the lifetime of the replaced page is at least $W_{D2N} > W_{Dj}$. The section 5.1.3 analysis can be reused by using only the linear portion of the distribution. □

Lemma 6. *The lifetime of a Physical Sparing algorithm that follows a normal model for the weakest $2N$ pages and any specific distribution of endurance for the stronger ones is equal or larger than the lifetime of the Physical Sparing algorithm under a normal model.*

Proof. The probability of a page requiring more than one replacement is not affected, since the lifetime of the replaced page is at least $W_{D2N} > W_{Dj}$. The section 5.1.4 analysis can be reused by using only the normal portion of the distribution. □

5.2 USES OF THE LIFETIME MODELS

In Section 5.1, we showed that system lifetime under a specific endurance model can vary depending on the algorithm, the percentage of spares and percentage of weak pages. The analysis and results above can be used in design tool to obtain the longest lifetime of the memory.

The decision to use PS or PCD depends primarily on the lifetime distribution of the pages. For the *constant* model, PCD will result in the highest lifetime. In the *linear* and *normal* models the difference is small allowing the decision to be taken based on other design constraints.

The *bimodal* model is less straightforward and two cases should be examined. The first case is when a manufacturer produces a device with size M and wants to sell it with a size L. If $\frac{K}{N} \leq 1$ or $\frac{K}{N} > 2$ then PCD is the recommended algorithm according to Equations (5.1)-, (5.3), (5.2) and (5.4). Equations (5.1)-(5.4). In the more interesting case, $1 < \frac{K}{N} \leq 2$, the algorithm selection depends on the relative values of $\frac{N}{M}$ and $\frac{K}{N}$. Specifically, if $\frac{N}{M} \geq 1 - 1/\frac{K}{N}$ then PS should be used, otherwise PCD is recommended.

In a second case, a manufacturer produces a device of capacity M with K bad cells and by choosing N and the endurance algorithm, can market it as a device of size $L{=}M{-}N$. Using the objective of highest lifetime with the largest addressable space, the selection of the endurance algorithm and of N are coupled. Figure 35 shows how the lifetime changes when N is varied in a system with a fixed M and K. The maximum lifetime is achieved when N is bigger than K, and we can choose $N{=}K$ to minimize resources, because the lifetime does not change for all values of N that satisfies this relation. It is possible that, for marketing reasons, restrictions will prohibit the use of $N{\geq}K$ (e.g., when an already advertised device size has to be sold but the devices were produced with too many weak pages). In this case, if the restriction on N allows $N{\geq}\frac{KM}{K+M}$, then PS should be used because $N{\geq}\frac{KM}{K+M}$ implies $\frac{N}{M}{\geq}1{-}1/\frac{K}{N}$ and Equation (5.9) indicates that the lifetime of PS is longer than PCD (see middle section of the figure). Finally, if it is not possible to use $N{\geq}\frac{KM}{K+M}$, then the use of spares is not recommended since (in this case) the lifetime is constant for the PCD algorithm for any value of spares, as shown on the left section of Figure 35.

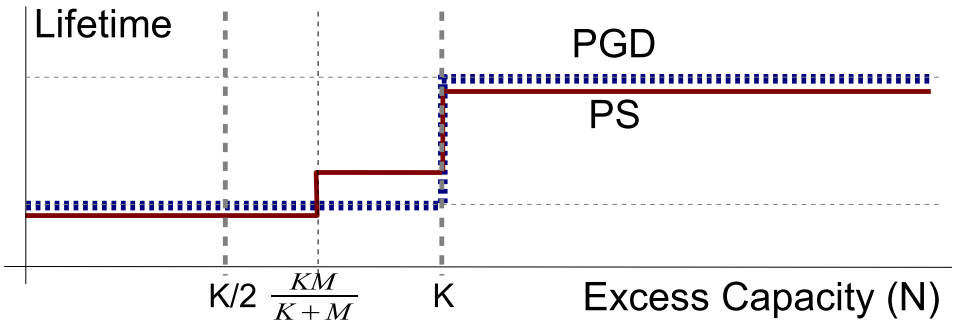

Figure 35: Lifetime impact of varying N for a fixed K and M.

At times, the manufacturer knows M, N, and L but K is variable. This can happen, for example, due to wafer to wafer process variation or fabrication process improvements. The selection of the algorithm to operate the memory will be based on the region the memory subsystem falls in, as described above.

All the previous results assume that the weak pages are distributed randomly. If it is possible to identify the weak pages, then those pages should be used as spares and a PS algorithm should be used. This result is valid for the region $1{<}\frac{K}{N}{\leq}2$. The use of weak pages as spares guarantees that the constraint of Equation (5.5) is valid, increasing the lifetime of the PS algorithm.

In a constant model, PCD lifetime is independent of the number of spares but PS lifetime actually decreases with a larger number of spares. In this model, reserving space as excess capacity is unnecessary and all memory should be exported to the system. A linear model with a low value of R behaves in a similar fashion to the constant model and the same recommendations apply. A linear model with larger R will have a similar lifetime with either PCD or PS so either can be used. The amount of excess capacity reserved determines the expected memory lifetime, since a larger excess capacity will also increase lifetime while reducing the addressable space.

6.0 CONCLUSIONS

In this dissertation, a new hybrid main memory architecture is proposed that uses multiple memory technologies, namely, Phase-change memory, DRAM and SRAM, to achieve higher energy-efficiency than a conventional DRAM based main memory, while maintaining the same level of performance. The objective of high performance and lower energy required that a number of novel algorithms and mechanisms were developed to overcome limitations of PCM technology. The validation of the new main memory was enabled by a new main memory simulation infrastructure, that allowed many different configurations of PMMA and DRAM-only main memory to be evaluated in terms of energy, performance and lifetime. A novel low-overhead wear-leveling algorithm was proposed that extended the main memory lifetime to be similar to the server's. The impact of process variation of the endurance of the memory cells on the memory lifetime is analyzed and mitigated by the proper selection of algorithms and parameters.

The results on PMMA shows that PCM is a viable alternative but not a direct replacement to DRAM as main memory. PMMA is a high-performance and energy-efficient main memory architecture with savings of up to 65% in energy with a performance loss of less than 5% leading to a very advantageous energy-delay gain of 60%. The use of a relatively small DRAM as a cache for PCM and SRAM as an even smaller auxiliary structure is fundamental to obtain those results reinforcing the proposal of a hybrid approach. Various performance enhancements are utilized to achieve the results, such as DRAM bypass, asymmetric page partitioning and a new page replacement algorithm such as clean-preferred LRU.

A sensitivity analysis was executed to determine the impact of various PMMA configurations that shows that PMMA performance is mainly limited by the PCM bus bandwidth but only lightly affected by PCM latency suggesting that future PCM devices should privilege bandwidth instead of latency. The PMMA architecture was designed to replace an existing DRAM memory controller

and DRAM without changes on the CPU, OS, applications and utilize existing implementations of PCM and DRAM devices. PMMA's goal to be a viable solution was achieved by managing sizes of metadata and caches. Asymmetric page partitioning is the tool used to achieve a reduction in size of the architecture overhead without requiring a very large DRAM cache or giving up performance.

Our swap-based wear-leveling algorithm has a very low overhead, requiring very little architectural support beyond a mapping table. When compared to existing algorithms, the biggest advantage of our algorithm is the very little energy overhead by only requiring an insignificant number of writes to PCM to achieve a lifetime of 8 years. Our algorithm is also naturally resistant to bad application behavior such as malicious or faulty behavior that could reduce the memory lifetime, since it uses randomized selections, avoiding patterns that can be exploited.

Wear-prone memory technologies, such as PCM and Flash, provide excess capacity as a way to increase lifetime. Our methods identify rules that empower designers to achieve the best use the devices. One interesting aspect is even for the same generic distribution, changes on physical size of the memory, number of spares or parameters of the distribution can benefit a different wear-leveling algorithm. We also demonstrate that under some conditions, other endurance distributions can be approximated by one of the analyzed ones, allowing reuse of the rules. The results show that for specific endurance statistical distributions, twice the lifetime can be achieved.

PMMA creates a high performance, energy-efficient, durable main memory but does not explore all properties that PCM provides. PCM is a non-volatile memory and that characteristic is not explored in this dissertation. The non-volatile property can be used to improve system energy-efficiency and reliability. We expect that many areas, mainly in operating system and systems research, will be affected and can explore the use of a non-volatile main memory. PMMA was designed to be a plug-in replacement of a DRAM-only main memory, but a clean design that explores specific application characteristics, such as very large data intensive applications may lead to a different architecture and it is an important new venue to be explored.

BIBLIOGRAPHY

[1] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem. Near-memory caching for improved energy consumption. In *Proc of the International Conference on Computer Design (ICCD)*, San Jose, CA, Oct 2005.

[2] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem. Energy conservation using Power Aware Cached-DRAM. In *IEEE Transactions on Computers*, volume 56-11, pages 1441–1455, 2007.

[3] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem. Power management in external memory using PA-CDRAM. In *The International Journal for Embedded Systems (IJES)*, volume 3-1, pages 65–72, 2007.

[4] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, pages 3–14, Washington, DC, USA, 1998. IEEE Computer Society.

[5] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. In Madison Mark D. Hill, University of Wisconsin, editor, *Synthesis Lectures on Computer Architecture*. Morgan and Claypool, 2009.

[6] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[7] Sangyeun Cho and Hyunjin Lee. Flip-n-write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 347 –357, December 2009.

[8] Standard Performance Evaluation Corporation. Spec. http://www.spec.org.

[9] Dell. Dell PowerEdge R910 4U Rack Server. http://www.dell.com/us/en/enterprise/servers/poweredge-r910/pd.aspx?refid=poweredge-r910&s=biz&cs=555.

[10] Alexander Driskill-Smith. Latest advances and future prospects of STT-RAM. Non-Volatile Memories Workshop, April 1113 2010. University of California, San Diego.

[11] Kang et al. A 0.1 μm 1.8V 256Mb 66MHz Synchronous Burst PRAM. In *Proc. IEEE International Solid-State Circuits Conference (ISSCC'06)*, 2006.

[12] Everspin. Everspin MRAM products 16 parallel bit memory interfaces, Apr 2010. http://www.everspin.com/products.php?hjk=16&a1f3=16Mb.

[13] International Technology Roadmap for Semiconductors. Process integration, devices and structures. In *Int'l. Technology Roadmap for Semiconductors*, 2009.

[14] International Technology Roadmap for Semiconductors. Process integration, devices and structures (update 2010). In *Int'l. Technology Roadmap for Semiconductors*, 2010.

[15] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling. In *In Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 109–120, February 2007.

[16] B. Gleixner, F. Pellizzer, and R. Bez. Reliability characterization of phase change memory. In *Non-Volatile Memory Technology Symposium (NVMTS), 2009 10th Annual*, pages 7 –11, October 2009.

[17] Darryl Gove. CPU2006 working set size. *SIGARCH Comput. Archit. News*, 35(1):90–96, 2007.

[18] Jon Haas and Pete Vogt. Fully-buffered DIMM technology moves enterprise platforms to the next level, March 2005. http://www.intel.com/technology/magazine/computing/fully-buffered-dimm-0305.htm.

[19] IBM. IBM power™systems family quick reference guide, Dec 2009. http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=SA&subtype=ST&attachment=POY03001USEN.PDF&appname=STGE_PO_PO_USEN&htmlfid=POY03001USEN.

[20] Bruce Jacob, Spencer W. Ng, and David T. Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann Publishers, 2007. Pg 349.

[21] Kinam Kim and Su Jin Ahn. Reliability investigations for manufacturable high density PRAM. In *Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International*, pages 157 – 162, 2005.

[22] Lai and T. Lowrey. OUM - A 180nm Nonvolatile Memory Cell Element Technology for Stand Alone and Embedded Applications. In *IEEE IEDM*, 2001.

[23] C. Lam. Phase-change Memory. In *Device Research Conference, 2007 65th Annual*, 2007.

[24] B.C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-change technology and the future of main memory. *Micro, IEEE*, 30(1):143, Jan-Feb 2010.

[25] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM.

[26] Lee, K. J. et al. A 90 nm 1.8 V 512 Mb Diode-Switch PRAM With 266 MB/s Read Throughput. *Solid-State Circuits, IEEE Journal of*, 43(1):150–162, Jan. 2008.

[27] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller. Energy management for commercial servers. *Computer*, 36(12):39 – 48, dec. 2003.

[28] Jiang Lin, Hongzhong Zheng, Zhichun Zhu, Howard David, and Zhao Zhang. Thermal modeling and management of DRAM memory systems. In *In Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 312–322, June 2007.

[29] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50 –58, February 2002.

[30] J. Maimon, K. Hunt, J. Rodgers, L. Burcin, and K. Knowles. Radiation Hardened Phase Change Chalcogenide Memory: Progress and Plans. In *Proc. 4th Annual Non-Volatile Memory Technology Symposium '03)*, 2003.

[31] R. Melhem, D. Mossé, and E. Elnozahi. The interplay of power management and fault recovery in real-time systems. In *IEEE Transactions on Computers*, volume 53-2, pages 217–231, 2004.

[32] Micron. Datasheet for MT36HTS1G72F 8GB FB DRAM memory. http://download.micron.com/pdf/datasheets/modules/ddr2/HTS36C1Gx72F.pdf.

[33] Micron. Datasheet for MT41J128M8BY-187 1Gb DDR3 DRAM memory. http://download.micron.com/pdf/datasheets/dram/ddr3/1GbDDR3SDRAM.pdf.

[34] Micron. DDR2 memory power calculator. http://download.micron.com/downloads/misc/ddr2_power_calc_web.xls.

[35] Micron. Numonyx omneo P8P PCM 128-mbit parallel phase change memory. http://www.micron.com/get-document/?documentId=5829&file=316144_P8P_DS.pdf.

[36] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 146 –160, dec. 2007.

[37] Numonyx. Phase change memory. http://www.pdl.cs.cmu.edu/SDI/2009/092309.html, 2010. Presentation at Carnegie Mellon, Sept 23rd, 2009.

[38] E. Prinz. The Zen of Nonvolatile Memories. In *Design Automation Conference*, 2006.

[39] M.K. Qureshi, M.M. Franceschini, and L.A. Lastras-Montano. Improving read performance of phase change memories via write cancellation and write pausing. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –11, January 2010.

[40] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, New York, NY, USA, 2009. ACM.

[41] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 24–33, New York, NY, USA, 2009. ACM.

[42] Rambus. Rambus XDR. http://www.rambus.com/in/downloads/document_abstracts/products/xdr_ts_v1_0.html.

[43] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465 –479, July 2008.

[44] Stuart Strechner, Gabriel H. Loh, Karin Strauss, and Doug Burger. Use ECP, not ECC, for hard failures in resistive memories. In *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, New York, NY, USA, 2010. ACM.

[45] C. Villa, D. Mills, G. Barkley, H. Giduturi, S. Schippers, and D. Vimercati. A 45nm 1Gb 1.8VPC phase-change memory. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 270 –271, February 2010.

[46] Wangyuan Zhang and Tao Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 2–13, New York, NY, USA, 2009. ACM.

[47] Wangyuan Zhang and Tao Li. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 101–112, Washington, DC, USA, 2009. IEEE Computer Society.

[48] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 14–23, New York, NY, USA, 2009. ACM.

[49] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. Fine-grained QoS scheduling for PCM-based main memory systems. In *IPDPS 2010: Proceedings of the 36th annual international symposium on Computer architecture*, pages 14–23, New York, NY, USA, 2010. ACM.