Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase Change Memories

Moinuddin K. Qureshi Michele M. Franceschini Luis A. Lastras-Montaño John P. Karidis

IBM T. J. Watson Research Center, Yorktown Heights NY

{moinqureshi, franceschini, lastrasl, karidis}@us.ibm.com

ABSTRACT

Phase Change Memory (PCM) is emerging as a scalable and power-efficient technology to architect future main memory systems. The scalability of PCM is enhanced by the property that PCM devices can store multiple bits per cell. While such Multi-Level Cell (MLC) devices can offer high density, this benefit comes at the expense of increased read latency, which can cause significant performance degradation. This paper proposes *Morphable Memory System (MMS)*, a robust architecture for efficiently incorporating MLC PCM devices in main memory. MMS is based on observation that memory requirement varies between workloads, and systems are typically over-provisioned in terms of memory capacity. So, during a phase of low memory usage, some of the MLC devices can be operated at fewer bits per cell to obtain lower latency. When the workload requires full memory capacity, these devices can be restored to high density MLC operation to have full main-memory capacity. We provide the runtime monitors, the hardware-OS interface, and the detailed mechanism for implementing MMS. Our evaluations on an 8-core 8GB MLC PCM-based system show that MMS provides, on average, low latency access for 95% of all memory requests, thereby improving overall system performance by 40%.

Categories and Subject Descriptors:
B.3.1 [Semiconductor Memories]: Phase Change Memory

General Terms: Design, Performance

Keywords: Phase Change Memory, Multi-Level Cell, Morphable Memory, Memory Monitor.

1. INTRODUCTION

As DRAM-based memory systems get limited by power wall and scalability challenges, architects are turning their attention towards exploiting emerging memory technologies for building future memory systems. Phase Change Memory (PCM) has emerged as one of the most promising technologies to incorporate into main memories. PCM devices are expected to scale better than DRAM and prototypes with feature size as small as 3nm have been fabricated [17]. Recently, several studies [8][25][14] have advocated for PCM-based memory systems, given the scalability and density advantages of PCM.

PCM devices exploit the property of chalcogenide glass to switch between two states, amorphous and crystalline, with the application of heat using electrical pulses. The phase change material can be switched from one phase to another a large number of times reliably. Data is stored in PCM devices in the form of resistance – the amorphous phase has high electrical resistivity and the crystalline phase has low resistance. The difference in resistance between the two states is typically 3 orders of magnitude. To achieve high density PCM memories are expected to exploit this high resistance range to store multiple bits in a single cell, forming what is known as Multi-Level Cell or MLC devices [1][4]. The density advantage of PCM is, in part, dependent on storing more and more bits in the MLC devices.

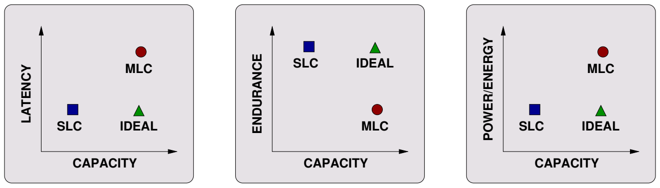


Figure 1: Comparison of SLC vs. MLC (Figure not to scale).

While MLC devices offer more density than devices that store one bit per cell (also called as SLC devices), this advantage comes at a significant price. MLC devices require precise reading of the resistance value. The number of levels in the MLC device increases exponentially with the number of bits stored, which implies that the resistance region assigned to each data value decreases significantly. For example, in a 4-bit per cell device the resistance range is divided so as to encode sixteen levels, and reading the data stored in the cell requires accurately differentiating between the 16 resistance ranges. The read latency of MLC devices, depending with the sensing amplifier technology, can increase linearly or exponentially with the number of bits. Furthermore, when programming (writing) an MLC device, it is likely that attaining a resistance value that is within the desired range will require the use of iterative write algorithms [13][15]. These algorithms contain several steps of read-verify-write operations that increase the write energy and further negatively impact the limited lifetime of PCM memories. Figure 1 captures this trade-off between MLC and SLC. Ideally we would like to have PCM memories with the capacity of MLC devices at the latency, energy, and lifetime of SLC devices.

Architecting a main memory system with MLC devices is thus a challenge as the density advantage of MLC can be useful only if the associated drawbacks are handled carefully. The objective of our paper is design a memory architecture that can obtain the latency, lifetime, and energy of fewer bit per cell devices in the common case while still having the ability to provide the high memory capacity enabled by MLC.

We base our solution on the observation that memory capacity requirements vary widely between different applications as well as within the execution of the same application, and the system is typically provisioned with enough memory capacity to be able to efficiently execute the worst-case workload of interest. In current DRAM-based systems, increasing the main memory capacity does not significantly change the memory latency. However, in case the memory system is based on MLC devices, the capacity boost comes at the expense of increased latency. If the workload does not use all the capacity, then it is beneficial to convert the excess main memory capacity to store fewer bits per cell (from MLC to SLC, for example). Especially given that it is fairly easy to restrict the number of levels used in a MLC device to emulate a fewer bits per cell device. To this end, we propose *Morphable Memory System (MMS)*, that dynamically regulates the number of bits per cell in memory system depending on the workload requirement.

Conceptually, MMS divides the main memory into two regions. First, a high-density high-latency region that contains pages in MLC mode. We denote such a memory region as *High-Density PCM or HDPCM* region. Second, a low-latency low-density region that contains the PCM devices in half the number of bits per cell than in the HDPCM region. We denote such a memory region as *Low-Latency PCM or LLPCM* region. As the percentage of total memory pages that are in LLPCM mode increases, the likelihood of an access being satisfied by the LLPCM region increases, but at the expense of reduction in overall memory capacity. Thus, the key decision in MMS is to determine what fraction of all memory pages must be in LLPCM mode to optimally balance this latency and capacity trade-off.

MMS contains a Memory Monitoring circuit (MMON) that tracks the workload memory requirement at runtime to determine the best partition between LLPCM and HDPCM regions. MMON performs the well-known Stack Distance Histogram (SDH) analysis [12] [21] [16], at runtime for a few sampled pages to estimate the page miss ratio curve [24]. This information along with the estimated benefit from accessing pages in LLPCM region is used to determine the best partition. MMS periodically consults MMON to obtain the estimate for the best partition and accordingly varies the number of pages in LLPCM region.

If a memory access occurs to a page in the HDPCM region, that page can be upgraded to the LLPCM region for reduced latency on subsequent accesses. MMS allows such transfers between the HDPCM and LLPCM regions in order to automatically provide lower latency to frequently accessed pages. Such a transfer between HDPCM and LLPCM region is handled transparently by the MMS hardware, without any involvement of software or the OS. A separate hardware structure, called the *Page Redirection Table (PRT)*, keeps track of the physical location of each page and is consulted on each memory access.

Unlike conventional memory system, the total memory capacity (in terms of number of pages) that are available to the OS can vary at runtime in a system that implements MMS. We provide a hardware-OS interface to facilitate this communication. This allows the OS to evict some of the allocated pages to make them available to the MMS hardware, if the number of pages in the LLPCM region is to be increased. When the demand for memory capacity increases the hardware transfers pages from the LLPCM region to HDPCM region, and the pages thus freed up can be reclaimed by the OS to accommodate another page.

We evaluate the MMS proposal on a 8GB PCM-based system consisting of MLC devices. Our evaluations with 10 workloads show that for workloads with low capacity requirement MMS performs similar to a memory system with fewer bits per cell, whereas for workloads that are capacity constrained MMS offers the full memory capacity. On average, MMS services 95% of all read requests and 90% of writebacks from the LLPCM region. MMS improves overall system performance by 40% on average. Satisfying most of the accesses in LLPCM region also has the associated benefits of saving energy and improving lifetime.

2. BACKGROUND ON MULTI-LEVEL PCM

PCM is emerging as a promising technology to build main memory systems in a power-efficient and scalable manner. One of the important ways in which PCM devices are expected to scale in terms of bits per unit area is by storing multiple bits per cell. As this is one of the first architecture papers on efficiently exploiting such Multi Level Cell (MLC) PCM devices, this section first provides the basic operation of MLC, then the trade-offs with MLC devices, and finally the design of a morphable PCM cell.

2.1 What is MLC and Why MLC?

PCM is a memory technology that stores data by programming and reading the resistance of the memory elements. Although earlier research on PCM was focused on single bit operation, recently, the focus of PCM research has been moving towards MLC operation, mainly to keep up with the increasing demand for low cost memory. Figure 2 shows the concept of MLC devices that can store 2 bits/cell and 4 bits/cell. The resistance range of each level decreases exponentially as the number of bits per cell increases. The maximum number of bits that can be stored in a given MLC device is a function of precision in reading technology, device data integrity, and precision in writing. As technology improves, the number of bits in MLC devices is expected to increase. The ITRS roadmap [1] projects that PCM cells will have 4 bits/cell by 2012.

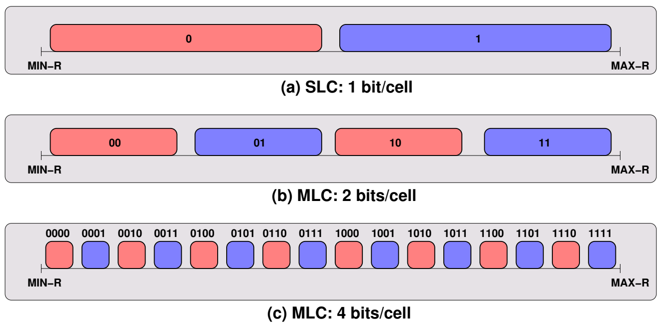


Figure 2: Concept of Multi-Level PCM (Figure not to scale)

2.2 Reading Techniques for MLC

Reading a data value from MLC device requires distinguishing precisely between different resistance levels that are spaced closely. The reading operation is performed by selecting, biasing, and sensing the given PCM device. Although, to the best of our knowledge, there is no published material on sensing devices for MLC PCM, from an high level perspective, possible solutions are limited to the most area efficient ADCs. We will discuss two ADC technologies that are scalable to more than 2 bits per cell.

2.2.1 *Integrating ADC Model*

An integrating ADC (e.g., *charge run-down ADC*) [3] uses a counter to measure the discharge time of a capacitor through the PCM element. An example of this scheme is shown in Figure 3. To represent n bits, the counter must be able to count 2^n different values, thus implying that the number of clock steps is exponential in the number of bits stored in the device. The difference in sensing time between the lowest data value and the highest data value grows exponentially, indicating doubling of sensing time from SLC to 2 bits/cell, and quadrupling from 2 bits/cell to 4 bits/cell.

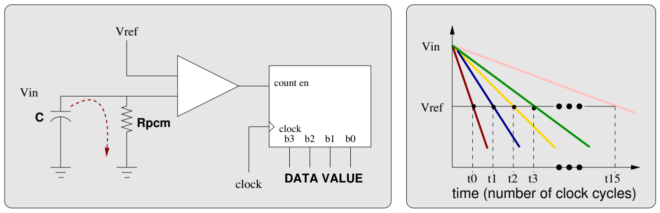


Figure 3: Example of integrating ADC: structure and working.

2.2.2 *Successive Approximation ADC Model*

Figure 4 shows an example of a successive approximation ADC [18, 3]. The conversion is performed in steps, at each step the hardware is reused to obtain an additional bit. The sensing time with this approach increases linearly with the number of bits stored in the MLC devices, thus increasing by a factor of two with each doubling of the number of bits per cell.

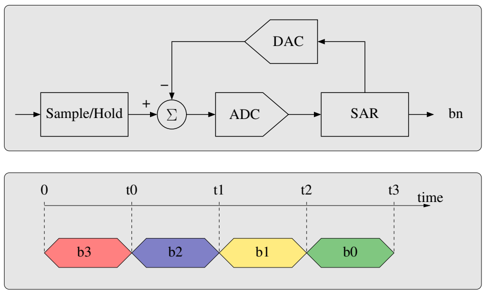


Figure 4: Example of successive approximation ADC.

2.3 Writing Technique for MLC

In MLC devices, each data value is assigned a limited resistance range, which means the writing process must be accurate enough to program that narrow range of resistance. The increased programming precision is obtained by means of iterative write algorithms that contains several steps of write-and-verify operations [15]. The number of iterations required for writing increases (almost exponentially) with the number of bits per cell [13]. Thus, with more bits per cell, these algorithms will consume increasingly more write energy, and exacerbate the limited lifetime of PCM memories.

2.4 A Morphable PCM Device

While MLC devices offer more density, this advantage comes at significant price of increased read latency, increased energy consumption, and reduced lifetime. Ideally, we want a PCM cell that offers the density of MLC but the latency, energy, and lifetime of fewer bits/cell devices. While this may not be practical, it is possible to program the MLC device to store restricted data values and emulate the functionality (and performance) of a lower density device. Examples of such flexible devices can be already found in Flash [5][7], and will conceivably be available in PCM.

PCM is fundamentally an analog medium that stores data in terms of resistance. For changing the number of bits per cell in a PCM cell, one need not change the device itself but just make minor changes to the sensing circuit. For example, a normalized resistance (R) value of 0.6 would indicate a "1" $(R > 0.5)$ in SLC mode and a "10" $(0.5 < R < 0.75)$ in 2 bits per cell MLC mode. Similarly, a cell that stores 4 bits per cell can also be used to store 2 bits per cell by simply regulating the resistance range in which data values get stored and providing a control signal that specifies the bits/cell associated with the device. For example, in integrating ADC model, a subset of the levels can be used and the *Vref* value can be increased to reduce the reading time. Whereas, in case of successive approximation ADC, only the most significant bits can be retrieved, which improves the reading time linearly.

We refer to such a MLC PCM cell that can be programmed to two different cell density as a *Morphable PCM Cell*. The Morphable PCM cell can be operated in two modes: First, high-density high-latency *(HDPCM)*, which stores as many bits per cell as permitted by the technology. Second, low-density low-latency *(LLPCM)*, which stores half the number of bits per cell but with low latency access.[1] Throughout this paper we assume that such a morphable PCM device is feasible and can be used in memory system.

3. MOTIVATION

MLC PCM is expected to be a cost effective means of providing a large main memory. However, architecting a main memory system with MLC devices is a challenge as the density advantage of MLC can be useful only if the associated drawbacks are handled carefully. We base our solution on the observation that memory capacity requirement varies widely across applications, and the system is typically provisioned with enough memory capacity to be able to efficiently execute the worst-case workload of interest. For example, consider the benchmarks in the SPEC CPU 2006 suite. There are 29 benchmarks in the suite, some with multiple (reference) input sets, for a total of 55 unique benchmark-inputset pairs. SPEC specifies that the machine on which these benchmarks are evaluated must have at-least 1GB of main memory [20], and the worst-case workload indeed requires close to 1GB of memory.

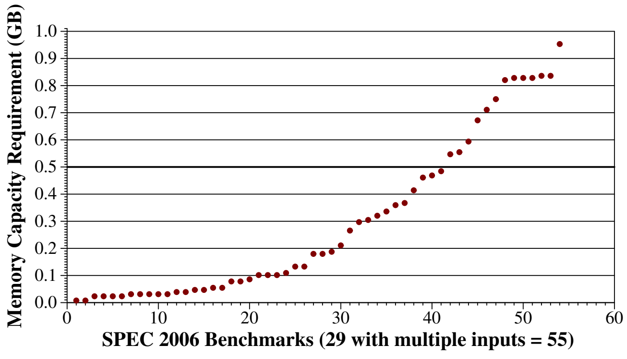


Figure 5: Variation in memory capacity requirement of different applications. Each dot in the graph represents one benchmark-inputset pair from the SPEC CPU 2006 suite.

[1]The density vs. latency trade-off is present not only in MLC and SLC but also within MLC devices of different densities. For example, HDPCM may denote 4 bits/cell MLC and LLPCM can denote 2 bits/cell MLC, and MMS can try to obtain the density of 4 bits/cell at the latency of 2 bits/cell. Therefore, we use the terms HDPCM and LLPCM instead of MLC and SLC.

Figure 5 shows the memory requirement curves for these 55 benchmarks.[2] While the worst-case requirement is close to 1GB, most of the applications require much less than 0.5GB of memory capacity. However, the system must be provisioned with enough capacity to run the worst-case benchmark in the suite. Otherwise, such benchmarks would experience significantly degraded execution time because of memory thrashing, given that page faults require two to four orders of magnitude more latency to service than memory accesses.

It is common to over-provision memory and often applications do not use all the memory throughout the execution. When applications do not use all the memory capacity, it would be appealing to avoid paying the penalties associated with high density MLC when a smaller, less dense memory would have provided better performance. If we can morph the memory on-the-fly, then it is possible to get the benefit of low density PCM in common case, and still retain capacity for applications that need all the memory capacity. To this end, we propose *Morphable Memory System (MMS)*, a memory architecture than can dynamically regulate the bits/cell in MLC devices depending on the workload requirement.

4. MORPHABLE MEMORY SYSTEM

MMS uses a memory consisting of MLC devices that can be morphed between LLPCM and HDPCM modes. For applications that are not capacity constrained, it is beneficial to have most of the memory in LLPCM mode. Whereas, for capacity-intensive workloads it is better to have most (or all) of the memory in the HDPCM mode. MMS achieves these goals. The next subsections describe the architecture and working of MMS.

4.1 Architecture

Figure 6 shows the architecture of MMS. Main memory is arranged in terms of pages[3] (4KB each) and physical memory units (fixed number of PCM cells storing either 2KB or 4KB). Each page can be in one of two states: HDPCM (occupying one 4KB memory unit) or in LLPCM (occupying two 2KB memory units). The OS sees a addressable memory size as if each page occupies one memory unit, i.e., all pages in HDPCM mode. In this way, there is a one to one relation between addressable pages and memory units.

Conceptually, MMS divides the main memory into two regions. First, a high density high latency region (HDPCM region). Second, a low latency, low density region (LLPCM region). A key decision in MMS is to determine what fraction of all memory pages must be in LLPCM mode to optimally balance the latency and capacity trade-off. A memory monitoring (MMON) circuit observes the traffic received by main memory to estimate the capacity requirement of the workload. Periodically, the MMON is consulted to determine the number of pages that must be in LLPCM mode for best overall performance. This number of pages is termed as the *LL-Target*.

When a page is accessed in HDPCM mode, it can be upgraded to LLPCM mode for lower latency. Such an upgraded page occupies

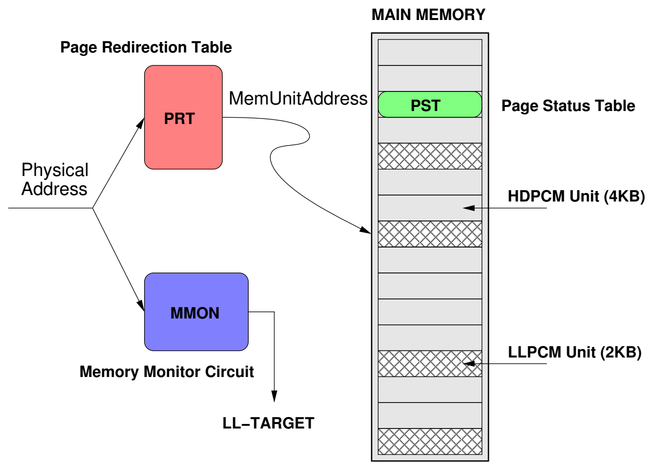


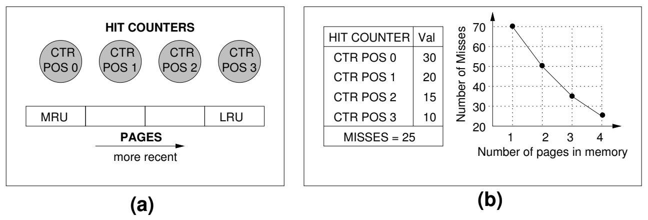
Figure 6: Architecture of Morphable Memory System (Figure not to scale)

two memory units (2KB each). The first half of the page is resident in its corresponding memory unit. A separate hardware structure, *Page Redirection Table (PRT)*, provides the physical location of the second half of the pages that are in LLPCM mode.

Given that some of the pages in memory can be in LLPCM mode, the number of pages usable by the OS reduces. Furthermore, for correctness reasons, the OS must ensure that it does not allocate a memory unit that is storing the second half of another page in LLPCM mode. This hardware-OS interface is accomplished by a memory mapped table, called the *Page Status Table (PST)*. The PST contains information about which units are usable by OS, and which units can be used as placeholders for the second halves of LLPCM pages. In the next subsections we will describe each structure of MMS.

4.2 Memory Monitoring (MMON)

The heart of the MMS is the memory monitoring circuit (MMON), which tracks the memory reference stream. The role of MMON is to estimate the statistics of the memory usage and to set a target for the fraction of LLPCM pages (LL-Target). To estimate the memory requirement of the system, we use the classical stack distance histogram (SDH) analysis [12] that has been widely used in literature for page miss rate curve [24] and cache requirement [21][16] studies. Using the SDH analysis it is possible to estimate the hit-rate for various capacity using a single pass. Figure 7(a) shows an example of SDH analysis for a memory containing four pages. Four counters, one each per recency position, count the number of hits obtained for that recency position. As shown in the example of Figure 7(b) given the counter values, it is possible to compute the miss rate when memory contained only one, two, three or four pages.



HIT COUNTER	Val
CTR POS 0	30
CTR POS 1	20
CTR POS 2	15
CTR POS 3	10
MISSES = 25	

Figure 7: Stack distance analysis for memory with four pages

[2]This data was generated by full run execution of each benchmark-inputset pair using an instrumentation tool. The memory size at which page misses increase by more than 1% compared to unlimited memory size is deemed the memory requirement of the benchmark. We do not use specrand in our studies.

[3]We use the term pages for a region of memory 4KB in size for convenience as it is the commonly used pagesize in OS. If the OS has a pagesize larger than 4KB, then MMS can manage the memory at a sub-page granularity of 4KB. This also allows MMS to automatically handle heterogeneous page sizes, as long as each pagesize is a multiple of 4KB.

A typical memory system contains millions of pages, so it is impractical to measure the exact recency information using a stack algorithm for a fully associative structure of this size. Therefore, we conceptually divide the memory into rows of 64 columns each, and perform a recency analysis using the ordering of pages only within each row. Thus, the SDH information will be discretized to 64 values, much like this information would be present in a 64-way cache. Figure 8(a) shows how such a MMON circuit would work by accruing global counts of hits in each recency position of any row. Note that each row only contains ordering information for all the columns in that row and does not contain any tag (all pages have a corresponding entry) or data values. The number of global counters equals the number of columns (64).

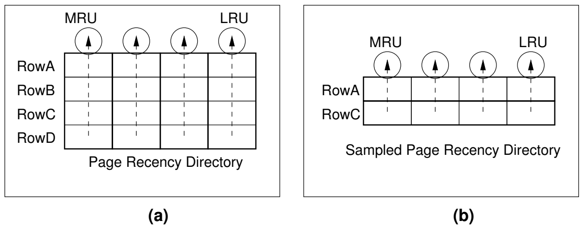


Figure 8: Memory monitor concept and implementation

Obtaining recency ordering for each row requires $64 \cdot \log_2 64 = 384$ bits $= 48$B per row. A typical memory system would contain several hundred thousand rows. We reduce the monitoring hardware by using the well understood concept of set sampling [23][16]. We use only 64 rows in MMON to estimate the memory hit rate. This means only $64 * 64 = 4096$ pages ($< 1\%$ of all pages) participate in monitoring. A sampled MMON has accuracy similar to having all rows participate and incurs a storage overhead of less than 4KB. The proposed MMON is shown in Figure 8(b). The counts of the sampled MMON are scaled by a factor that is equal to `NumTotalRows/NumSampledRows` to estimate the global hit counts.

```
1: BestOffCols = 0, MaxCols = 64, BestCycleSaving =0
2: for OffCols = 1 to MaxCols/2 do
3:     PageFaultIncrease = SDHCounters [MaxCols-1] + ... + SDHCounters[MaxCols-OffCols]
4:     LowLatencyHits = SDHCounters [0] + ... + SDHCounters[OffCols-1]
5:     CyclesSaved = LowLatencyHits * (HDPCM Hit Latency - LLPCM Hit Latency)
6:     CyclesIncrease = PageFaultIncrease * AveragePageFaultLatency
7:     if CyclesSaved-CyclesIncrease > BestCyclesSaving then
8:         BestOffCols = OffCols
9:         BestCycleSaving = CyclesSaved - CyclesIncrease
10:    end if
11: end for
12: Return LL-Target = (BestOffCols/MaxCols)*NumTotalPages (END)
```

Figure 9: Algorithm to estimate LL Target.

Periodically the counters in MMON are read to estimate the increase in page fault and the benefit from LLPCM hits. This is done by using the algorithm described in Figure 9. For a 64 column MMON, there are 33 possible values for LL-Target in terms of columns (0-32). The algorithm estimates for each such point the increase in execution time due to increased page-fault rate and reduction in execution time due to low latency LLPCM hits. The point that minimizes the expected execution time is selected as the LL-Target. We invoke this algorithm once every 250ms. At each invocation, the counters in MMON are multiplied by 0.9 to account for changing phase behavior, similar to as done in [21].

4.3 Changing Effective Memory Size via Ballooning and Page Status Table

The MMON estimates of LL-Target is periodically read by the OS. If the number of LLPCM pages in the system is less than LL-Target, the OS evicts some HDPCM pages so that the memory units associated with those pages can be used for LLPCM operation. To enable this, we use the concept of *Ballooning* which is an effective way of reclaiming idle pages in the system [23]. The concept of Balloon is described in Figure 10. A balloon process is a dummy process which can take away physical memory pages from running processes. When the OS wants to reduce the available pages, it inflates the balloon and vice versa.

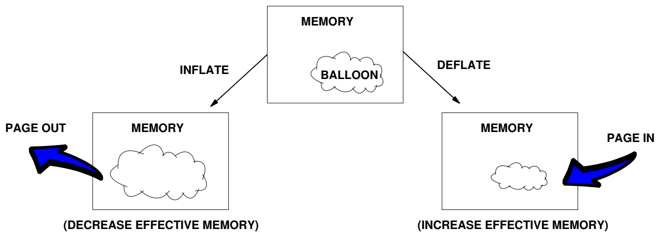


Figure 10: Concept of Balloon process for stealing pages [23]

In MMS, the memory units associated with the pages claimed by the balloon process, are marked by the OS to be free for storing second halves of LLPCM pages. This information is communicated to the hardware using a memory mapped table, called the *Page Status Table (PST)*. Each page can be in one of the four states: a normal OS page (`Normal`), a monitor page used by MMON (`Monitor`), a balloon page that is available to be used in LLPCM mode (`BalloonFree`), and a balloon page that is currently being used in LLPCM mode (`BalloonUsed`). The PST contains a two bit status field for each page (which means PST occupies $<$ 0.01% of main memory). The OS uses it to specify a balloon page, and to reclaim a `BalloonFree` page in case it wants to increase memory capacity. The PST avoids the scenario where OS can victimize a `BalloonUsed` page for allocating another page during page fault. The PST is kept coherent by disallowing any updates to PST from the hardware when the OS is operating on it.

When `NumBalloonPages` $<$ `LLTarget`, the OS converts some `Normal` pages into `BalloonFree` pages using its own replacement algorithm, ensuring that a `Monitor` page is not converted into `BalloonFree`. The hardware can then use these pages for LLPCM operation. When the `NumBalloonPages` $>$ `LLTarget`, the OS converts `BalloonFree` pages into `Normal` pages. If there are not enough `BalloonFree` pages, the hardware is instructed to actively convert some of the `BalloonUsed` pages into `BalloonFree` pages. The LL-Target thus set by MMON is enforced lazily by adjusting the number of balloon pages.

4.4 Redirection, Upgrades, and Downgrades

For pages that are stored in LLPCM mode, the actual physical location of the pages may be different from the physical address specified by the OS. MMS contains a hardware table called the *Page Redirection Table (PRT)*, which is responsible for the translation from a physical address to the PCM memory address. Each page has a corresponding entry in the PRT, which contains a valid bit, a ref bit, and a pointer. For pages in HDPCM mode, the associated PRT entry is invalid. For pages in the LLPCM mode, the associated entry has a pointer which stores the address of the physical memory unit that stores the second half of the page. The first half of such a page is always stored in place, albeit in LLPCM mode. We now describe the process of accessing memory in MMS, and converting pages between HDPCM and LLPCM modes.

4.4.1 Memory Access

The incoming line address is applied to the PRT. If the associated page has an invalid PRT entry then the page is a HDPCM page and the physical location corresponds with the incoming address. That location is accessed in HDPCM mode. If the PRT entry is valid, and the line address corresponds to the first half of the page, then the physical location still corresponds with the incoming address, except that location is accessed in LLPCM mode. Whereas, if the second half of the page is accessed then the pointer in the PRT entry specifies which physical location must be accessed. That location is accessed in LLPCM mode.

4.4.2 Upgrades

When there is a miss in the PRT, the HDPCM page may be converted into LLPCM page for lower latency. We call this process a *Page Upgrade*. This process is performed as follows. First, a victim unit is found either from one of the `BalloonFree` units or from one of the units that is already in LLPCM mode. If the victim is a `BalloonFree`, then it's status is changed to `BalloonUsed` in the PST. The contents of the original page are read out and stored in LLPCM mode for the first half, and the second half is stored in the victim unit, again in the LLPCM mode. The PRT entry for the page is upgraded to be valid, and the pointer field is set to the victim unit. If the victim unit was already in LLPCM mode, then the upgraded page resident in that location must be downgraded first, which we describe next.

4.4.3 Downgrades

We use the term *Page Downgrade* to denote the process of converting a page from LLPCM mode to HDPCM mode. A page downgrade can happen either when the page is being victimized to upgrade another page, or if the LL-Target is much smaller than the number of balloon pages, so the hardware is trying to convert `BalloonUsed` pages to `BalloonFree` pages. The process of downgrade is performed as follows: The data is read out from both memory units occupied by the LLPCM page and is stored in HDPCM format in the original location. The corresponding entry in PRT is invalidated. The PST entry is updated to change the status of the freed unit from `BalloonUsed` to `BalloonFree`.

4.5 OS Support

MMS requires the OS to support the dynamically changing memory capacity. First, the page replacement algorithm is updated to check PST and not victimize Balloon pages. Second, an OS process must periodically check (e.g., every 250ms) to ensure that the target set by MMON is close to number of balloon pages. If the target is higher, then the OS inflates the balloon to create more balloon pages. If the target is lower, then the OS deflates the balloon and converts `BalloonFree` pages into `Normal` pages.

5. CASE STUDY

We now analyze the effectiveness of MMS using a simple kernel *RandStep* as a case study to provide insights. Detailed evaluations with standard benchmarks will be presented in Section 7. RandStep randomly accesses one of the lines in its working set once every 100 instructions. The working set size changes once every 20 Billion instructions, as described in Figure 11. It has three modes: small working set (30% memory), medium working set (60% memory), and large working set (90% memory).

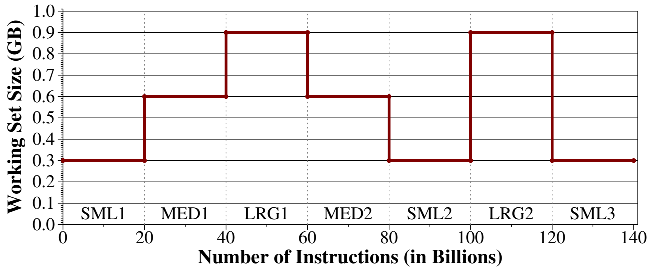


Figure 11: RandStep Kernel: Accesses random memory line in working set once every 100 instructions

Ideally the system should have almost all the memory in LLPCM mode for SML1, SML2, and SML3 phases. For LRG1 and LRG2 almost all the memory should be in HDPCM mode to accommodate the working set. We execute this kernel on a single-core system with 1GB of memory with MMS enabled (other parameters shown in Table 1). We sampled the system once every billion instructions to measure various statistics. Figure 12 shows the effective memory capacity visible to the OS for different phases of the kernel.

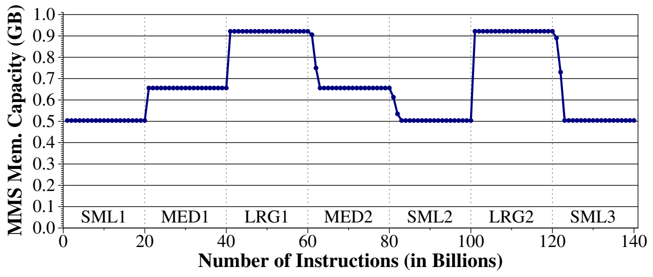


Figure 12: Effective memory capacity usable by OS with MMS

During the SML phases the effective capacity is halved which means almost all pages in memory can be in LLPCM mode. The effective capacity of the system increases as the demand for capacity increases. These results also show the need for dynamic adaptation to different phases as a static partition of memory between LLPCM and HDPCM regions causes thrashing in phases LRG1 and LRG2.

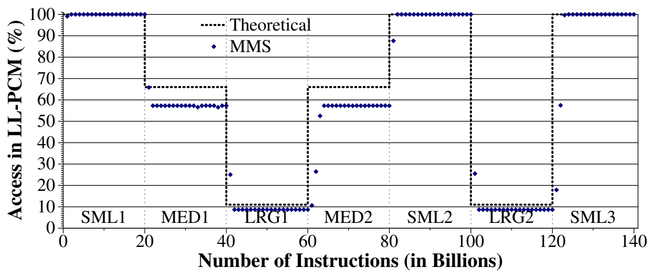


Figure 13: Percentage of memory accesses in LLPCM mode: Expected and with MMS

We also measured the number of accesses that occur in the LLPCM mode. Given that RandStep accesses memory randomly, we can compute the expected percentage of accesses in LLPCM mode for a theoretical system. For SML phases, all the memory accesses can happen in LLPCM mode. For MED phases, 40% of the memory pages are available for optimizing 60% of memory capacity, which means two-thirds of the accesses can be in LLPCM mode. And, for LRG phases, about 10%/0.9=11% of the accesses can happen in LLPCM mode. Figure 13 shows the percentage of all memory accesses that occur in the LLPCM region, both the expected theoretical bounds and the measured statistics with MMS. The two results match well and MMS comes close to the theoretical possible improvement.

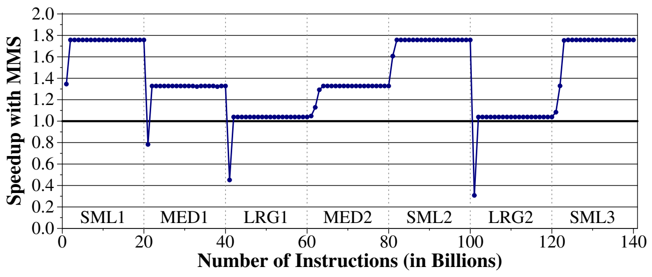


Figure 14: Normalized performance of MMS over HDPCM

Figure 14 shows the performance of MMS normalized with respect to the baseline system, which has all the pages in HDPCM mode, measured over periods of executing one billion instructions each. For SML and MED phases, MMS provides significant performance improvement. However, when the phases changes such that the working set increases instantly MMS takes more time than baseline to accommodate this working set, hence the drop in performance for phase transitions from SML to MED, MED to LRG, and SML to LRG. But once the working set gets accommodated MMS has similar or significantly better performance. MMS reduces overall execution time for this kernel by 22%, providing an effective speedup of 1.28x.

6. EXPERIMENTAL METHODOLOGY

6.1 Configuration

We use an in-house system simulator for our studies. The baseline system is shown in Figure 15 with the parameters given in Table 1. The system contains an eight core CMP. We use a simple in-order processor model so that we can evaluate our proposal for several hundred billion instructions. The baseline also contains a 256MB write-back DRAM cache organized as a 32MB per-core private cache. The DRAM cache uses a linesize of 128B. Write requests arrive at main memory only on DRAM cache evictions.

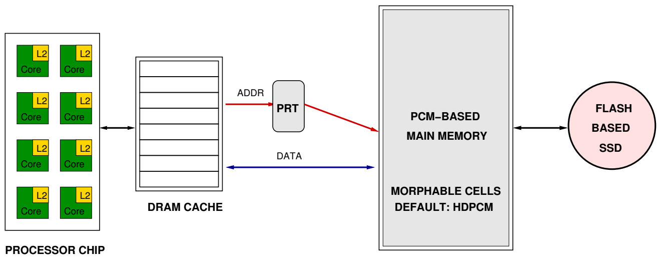


Figure 15: Simulated System (Figure not to scale)

The 8GB PCM-based main memory consists of morphable PCM cells that are defaulted to HDPCM mode in the baseline. We assume that the baseline memory system is sufficiently banked such that the contention between requests is negligible. Memory is assumed to have a read latency of 1000 cycles if the page is in HDPCM mode and 500 cycles if the page is in LLPCM mode.

A page size of 4KB is assumed. Virtual to physical address translation is performed using a page table built in our simulator. A clock style algorithm is used to perform page replacements. Page misses are serviced by a Flash based solid state disk (SSD), which has a latency of 100K cycles.

For the MMS enabled system, we incorporate an EDRAM-based PRT of 6MB (2M pages * 3 byte per entry).[4] To model the latency overhead of PRT-based indirection, we assume that each memory access in MMS incurs extra latency of 5ns (20 cycles). This latency overhead is not present in non-MMS systems.

Table 1: Baseline Configuration

System	8-core CMP, 4GHz single-issue in-order core private-L1
L2 cache (private)	2MB, 4-way, LRU, 128B linesize
DRAM cache (private)	32MB, 8-way, 128B linesize 25 ns (100 cycle) access, writeback policy
Main memory	8GB PCM, consisting of morphable PCM cells that are in HDPCM mode
PCM read latency	HDPCM : 250ns (1000 cycles) LLPCM : 125 ns (500 cycles)
Flash-Based SSD	unlimited size, read latency 25 micro seconds (100K cycles)

6.2 Workloads

We use eight benchmarks from the SPEC CPU 2006 suite: BWaves, CactusADM, GemsFDTD, lbm, mcf, leslie3d, milc, and zeusmp. These benchmarks were chosen because they have at least 1 memory access per 1000 instructions out of the 32MB cache (for workloads that fit in the cache, the proposed scheme neither helps nor hurts performance). We skip the first 50 billion instructions for each benchmark and then execute the next 50 billion instructions. Table 2 shows the main-memory read accesses per 1000 instructions, write backs per 1000 instructions, on average for a 1-core 1GB system. It also shows the page faults at 0.5GB, 0.6GB, and 0.75GB memory normalized to the page fault rate with 1GB memory. The first four benchmarks need less than half of the allocated 1GB capacity, milc needs slightly more than 0.5GB, and the next three benchmarks need more than 0.75GB.

Table 2: Benchmark Characteristics (PF denotes Relative Page Fault Rate with respect to 1GB Memory Size).

Name	Reads PKI	Writes PKI	PF 0.5GB	PF 0.6GB	PF 0.75GB
CactusADM	1.32	0.43	1.00x	1.00x	1.00x
lbm	6.42	3.22	1.00x	1.00x	1.00x
leslie3d	3.15	1.65	1.00x	1.00x	1.00x
zeusmp	1.68	0.69	1.00x	1.00x	1.00x
milc	10.8	4.08	1.35x	1.00x	1.00x
BWaves	6.4	0.86	5.08x	4.15x	2.75x
mcf	3.7	0.04	27.1x	26.1x	13.4x
GemsFDTD	6.6	3.36	70.4x	22.9x	7.00x

[4]The storage of PRT can be reduced by restricting both halves of LLPCM pages to be within 256 pages from each other. This would decrease the storage overhead of PRT to approximately 2MB. It is also possible to have a memory mapped PRT and cache the PRT entries in a smaller structure based on demand. We do not consider such optimizations in this paper.

To form multiprogrammed workloads, we run these benchmarks in a rate mode, where each processor executes the same benchmark in a separate virtual space. We also created two heterogeneous workloads: mix_1 consisting of two copies each of milc, Bwaves, mcf, and GemsFDTD. And mix_2 consisting of four copies each of mcf and GemsFDTD. The workloads used in our study are described in Table 3. We execute one benchmark per core. The simulation continues till each core has executed 50 billion instructions. Execution time of the workload is determined by the time when simulation terminates.

Table 3: Workload Summary.

Name	Description
cactus_r	8 copies of CactusADM
lbm_r	8 copies of lbm
leslie_r	8 copies of leslie3d
zeusmp_r	8 copies of zeusmp
milc_r	8 copies of milc
bwaves_r	8 copies of BWaves
mcf_r	8 copies of mcf
gems_r	8 copies of GemsFDTD
mix_1	2 copies of milc, BWaves, mcf, and GemsFDTD
mix_2	4 copies each of mcf and GemsFDTD

7. RESULTS AND ANALYSIS

7.1 Performance

Figure 16 shows the speedup (normalized to baseline) for four systems: the baseline 8GB system with all memory in HDPCM mode, an iso-area system that has 4GB memory all in LLPCM mode, the proposed MMS system with 8GB morphable memory, and a system that has 8GB all in LLPCM mode. Note that the 8GB LLPCM system requires 2x the area as the other three systems and it serves as an upper bound on the speedup possible with MMS. The bar labeled *Gmean* is the geometric mean over all the workloads.

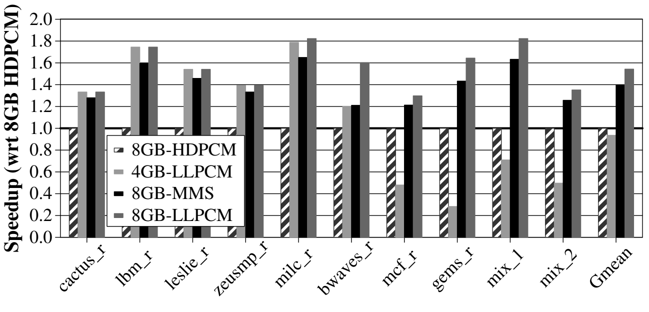


Figure 16: Speedup of MMS with respect to other systems

Neither 8GB HDPCM nor 4GB LLPCM works well across all workloads. For the first five workloads, latency is more important than capacity, hence they show significant improvement with the 4GB LLPCM system. BWaves has phases where sometimes it is latency sensitive and sometimes capacity sensitive. Four workloads, mcf_r, gems_r, mix_1, and mix_2, are capacity intensive and see significant degradation in performance with 4GB LLPCM system. Thus, simply using HDPCM memory instead of LLPCM memory does not provide significant overall improvement. MMS dynamically changes memory mode between LLPCM and HDPCM depending on application demand and provides an average speedup of 1.4x, which is significantly compared to the 1.54x possible with the upper-bound system.[5]

7.2 Memory Accesses from LLPCM

MMS divides memory into LLPCM and HDPCM regions. Ideally we would like to have almost all the accesses satisfied from the LLPCM region, as that mode of operation has lower latency and is expected to have better endurance and energy-efficiency. Figure 17 shows the percentage of hits serviced in the LLPCM mode with MMS. The data is separated in terms of read accesses, write backs from DRAM cache, and overall accesses. The bar labeled *Amean* represents the arithmetic mean measured over all workloads.

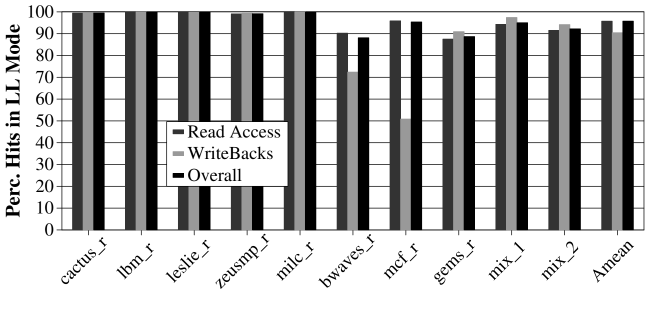


Figure 17: Percentage of accesses serviced from LLPCM

The first five workloads are not capacity sensitive and almost all of the read and write requests are satisfied in LLPCM mode. For capacity sensitive workloads (next five) almost 90% of the accesses are satisfied in LLPCM mode. The writeback stream in mcf_r has poor locality hence only half of the write-backs hit in LLPCM mode. However, mcf has negligible writeback traffic ($\approx$1% of all accesses). Overall, on average, 90% of the writebacks and 95% of read requests are satisfied in LLPCM mode. These benefits come at a small overhead of < 4 transfers between HDPCM and LLPCM, every 1000 memory accesses.

Energy Implications: Satisfying more than 90% of the accesses in LLPCM mode also means that energy consumption of the MMS system would be much closer to LLPCM rather than HDPCM. The exact energy savings between the two modes is a function of bits/cell in each mode and the ADC technology.

Lifetime Implications: Studies in Flash domain have shown MLC devices to have 10x lower endurance than SLC devices [22], indicating significant room for lifetime improvement. MMS enhances lifetime significantly compared to naively using MLC-PCM. For example, if SLC-PCM has endurance of 10^8 writes and MLC-PCM has lifetime of 10^7 writes , then the effective endurance with MMS would be approximately $0.9 \cdot 10^8$ writes, if 90% of the writes were satisfied in SLC-PCM mode.

Write Bandwidth Implications: MLC PCM is expected to suffer from poor write bandwidth. MMS fundamentally solves this problem by satisfying 90% of the writes in LLPCM (SLC) mode (no iterative writes). Since MMS satisfies most of the writes from SLC, the effective write bandwidth is much closer to SLC. For example, if SLC write bandwidth = 5 units, and MLC=1 units, then with MMS the write bandwidth is approximately equal to 4.6 $(0.9 \cdot 5 + 0.1 \cdot 1)$ units.

[5]We also evaluated the overall performance improvement on other metrics as well and got similar results. MMS obtained an average improvement of 38.4% on weighted speedup metric [19] compared to 53.5% possible with upper-bound system. MMS obtained an average improvement of 38% on hmean-fairness metric [11] compared to 53% possible with upper-bound system.

7.3 Dynamic Memory Capacity with MMS

Figure 18 shows the effective memory capacity when MMS is enabled, averaged over the entire execution. The first five workloads have close to 50% capacity, meaning almost all memory is in LLPCM mode. For the other five workloads, memory capacity gets extended to fit the working set each of the workload.

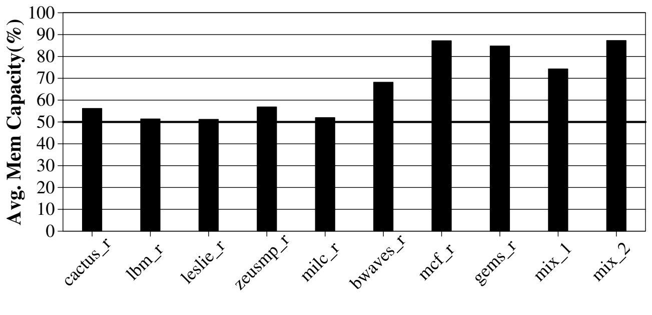


Figure 18: Average memory capacity with MMS

The average capacity statistics do not capture the dynamic variation in memory demand of workloads. Figure 19 shows the dynamic capacity with MMS, for bwaves_r and mcf_r. The data is obtained by sampling the system once every one billion instructions. BWaves has phases of execution where the working set increases close to full capacity, and MMS responds to this demand by increasing the available memory capacity. Other workloads, such as mcf_r, have a regular demand for memory capacity and hence a constant memory capacity with MMS, after the initial period.

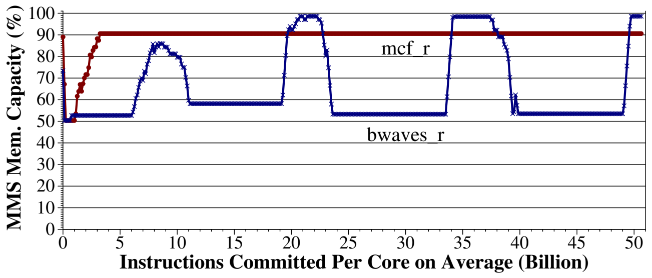


Figure 19: Phase behavior of bwaves_r and mcf_r with MMS

We also evaluated equal cell-capacity Static Memory System (SMS) that statically partitions between LLPCM and HDPCM. However, our baseline already has a relatively large (3.25% of memory-capacity) DRAM cache so a small partition of LLPCM does not help. If LLPCM partition is made large (25%) then it hurts workloads which need more (>75%) memory capacity. As MMS can adapt to variation in memory capacity demand, MMS is much more robust and effective than SMS.

7.4 Page Fault Rate

MMS tries to modulate the capacity of the system in response to the workload. If the workload gets less capacity than required, it would result in increased page fault rate. Figure 20 shows the page faults of 4GB LLPCM system and 8GB MMS system normalized to 8GB HDPCM baseline. For gems_r and mcf_r, the 4GB system has significantly increased page fault at 4GB, whereas MMS has page fault similar to baseline. For bwaves_r, the transitions in demand causes page fault rate to increase with MMS when capacity increases from 50% to 90%. Whereas, the 8GB baseline could accommodate the entire working set of bwaves_r at all times. On average, the number of page faults with MMS (1.3x) is much closer to the 8GB baseline system, as compared to the 4GB system (4.6x).

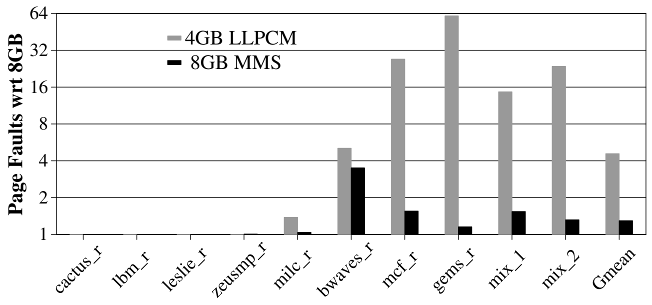


Figure 20: Normalized Page Fault Rate (Y axis in log scale)

8. RELATED WORK

In our work, we have exploited the observation that memory capacity demand varies at runtime. Therefore, a transient excess memory capacity can be used to improve performance by converting the memory from high-density mode to low-latency mode. Lim et al. [6] also showed that memory demand can vary even within the same application for workloads such as TPCC-H and proposed using memory blades to efficiently share capacity between multiple blade systems. Note that the overall capacity of the entire system still remains constant in their solution, and their solution is not geared towards improving hit latency of memory.

Kgil et al. [7] proposed dynamically downgrading MLC pages to SLC pages for Flash-based disk cache when the page nears its lifetime, thereby increasing the overall lifetime of the system. The concept of dynamically choosing between single level cells and multi-level cells for Flash storage was proposed in [9]. Note that managing storage vs. managing main memory are fundamentally different in that data from storage cannot be evicted by controller just to improve performance, whereas in main memory unused resident pages can be evicted to improve overall performance. FlexFS [9] leverages only unused portion (that does not store any data) of Flash storage for improving latency. It typically takes users several months or years to fill up an SSD, so leveraging unused portions for latency optimizations is acceptable in that domain. However, in our case, main memory typically fills up in minutes, so relying only on unused portion is not a viable option. Furthermore, FlexFS does not need to the estimate dynamically varying capacity demand. Whereas, the demand for main memory capacity changes frequently, and hence requires a dynamic mechanism to determine the optimal partition between fast region and high density region.

MMS tries to reduce memory latency by trading off memory capacity. A converse approach is to exploit redundancy in data stored in memory to increase capacity, albeit at higher latency. Both memory compression and MMS require similar support from OS to handle the dynamically changing memory capacity and to reduce or increase balloon pages [2] to control effective memory size. However, these two approaches are fundamentally different in that compression is dependent on data values and not on demand. It cannot be used to reduce the latency for memory hits. These two approaches are orthogonal and can be combined.

We have proposed a hardware managed MMS, where the hardware regulates which pages are in LLPCM mode and which pages are in HDPCM mode. MMS can also be implemented completely as a software module within the OS. In such a case, the OS can divide memory into two regions, and use dynamic page remapping [10] to upgrade frequently accessed page from HDPCM region to LLPCM region. Such a technique obviates the PRT-based indirection of MMS, but requires monitoring usage of each page and performing the upgrades and downgrades in software, which incurs significant performance overhead.

9. SUMMARY

The demand for main memory capacity and the scaling challenges of DRAM call for architects to look towards exploiting other emerging technologies that are expected to be more scalable. Phase Change Memory (PCM) has emerged as a promising technology that has better scalability than DRAM. One of the ways PCM devices are expected to improve density is by packing multiple bits per cell. While such Multi-Level Cells (MLC) have better density they have much higher read latency, reduced lifetime, and higher energy compared to cells that store one bit. Ideally we would like to have the capacity of MLC while having the latency, lifetime, and energy of SLC devices. To this end, our paper exploits the fact that typical applications do not use all the available memory capacity, and makes the following contributions:

1. We propose an adaptive infrastructure, *Morphable Memory System (MMS)* that can dynamically partition the memory into high-density pages and low-latency pages. MMS leverages morphable PCM cells that can be programmed to be in either high-density mode or low-latency mode.
2. We propose a cost-effective runtime mechanism to determine the best partition between high-density region and low-latency region in MMS. This mechanism uses stack distance histogram analysis and sampling to estimate the best partition, while incurring a total storage overhead of less than 4KB.
3. We provide the hardware-software interface for the OS to handle dynamically varying memory capacity. The OS uses the information from the monitoring circuit to regulate memory capacity in order to free up a given number of pages for low latency access.

We evaluate the MMS system using a kernel as well as ten workloads on a system with 8GB of main memory consisting of MLC PCM cells. Our evaluations show that on average MMS satisfies 95% of the read requests and 90% of the writebacks from the low latency region, while keeping the overall page fault rate close to the 8GB system. On average, MMS provides a performance improvement of 40%, bridging three-fourth of the performance difference between SLC and MLC PCM devices.

We believe an architecture such as MMS will be crucial in incorporating technologies that present a trade-off between density and latency, and can be morphed dynamically. In this work, we restricted morphing to two levels, in which one of the level has half the density of the other. A more generalized structure that allows any number of levels without restrictions can also be architected. In our proposal we optimized the partition based on performance, other partitioning algorithms better suited to optimize lifetime or energy can also be investigated. Furthermore, wear leveling algorithms better suited to morphable memory structures can be developed. Exploring such extensions is a part of our future work.

Acknowledgments

We thank Bulent Abali, Basil Smith, and Vijayalakshmi Srinivasan for discussions and feedback during this work.

10. REFERENCES

[1] *International Technology Roadmap for Semiconductors, ITRS 2008 Update*. http://www.itrs.net/Links/2008ITRS/Home2008.htm.

[2] B. Abali et al. Hardware compressed main memory: Operating system support and performance evaluation. *IEEE Trans. Comput.*, 50(11):1219–1233, 2001.

[3] Analog Devices Inc. *Data Conversion Handbook*.

[4] F. Bedeschi et al. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE Journal of Solid-State Circuits*, 44(1):217–227, 2009.

[5] T. Cho et al. A dual-mode NAND flash memory: 1-Gb multilevel and high-performance 512-Mb single-level modes. *IEEE Journal of Solid-State Circuits*, 36(11), 2001.

[6] L. Kevin et al. Disaggregated memory for expansion and sharing in blade servers. In *ISCA-36*, pages 267–278, 2009.

[7] T. Kgil, D. Roberts, and T. Mudge. Improving nand flash based disk caches. In *ISCA-35*, 2008.

[8] B. Lee et al. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA-36*, 2009.

[9] S. Lee et al. FlexFS: A Flexible Flash File System for MLC NAND Flash Memory. In *USENIX '09*, 2009.

[10] J. Lin et al. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *HPCA-14*, 2008.

[11] K. Luo et al. Balancing throughput and fairness in SMT processors. In *ISPASS-2001*.

[12] R. L. Mattson et al. Evaluation techniques in storage hierarchies. *IBM Journal of Research and Development*, 9(2):78–117, 1970.

[13] T. Nirschl et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *Proceedings of the IEEE International Electron Devices Meeting*, 2007.

[14] M. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA-36*, 2009.

[15] M. K. Qureshi et al. Improving read performance of phase change memories via write cancellation and write pausing. In *HPCA-16*, 2010.

[16] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO-39*, 2006.

[17] S. Raoux et al. Phase-change random access memory: A scalable technology. *IBM Journal of R. and D.*, 52(4/5):465–479, 2008.

[18] B. Razavi. *Principles of data conversion system design*. Wiley-IEEE Press., New York, 1995.

[19] A. Snavely, D. M. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *SIGMETRICS-2002*.

[20] The Standard Performance Evaluation Corporation. *Requirements of SPEC CPU 2006*. http://www.spec.org/cpu2006/Docs/system-requirements.html.

[21] G. E. Suh et al. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *HPCA-8*, 2002.

[22] J. Thatcher et al. Nand flash solid state storage for the enterprise: An in-depth look at reliability. Solid State Storage Initiative (SNIA), 2009.

[23] C. A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.

[24] P. Zhou et al. Dynamic tracking of page miss ratio curve for memory management. In *ASPLOS-XI*, 2004.

[25] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA-36*, 2009.