

PDRAM: A Hybrid PRAM and DRAM Main Memory System

Gaurav Dhiman
gdhiman@cs.ucsd.edu

Raid Ayoub
rayoub@cs.ucsd.edu

Tajana Rosing
tajana@ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

ABSTRACT

In this paper, we propose PDRAM, a novel energy efficient main memory architecture based on phase change random access memory (PRAM) and DRAM. The paper explores the challenges involved in incorporating PRAM into the main memory hierarchy of computing systems, and proposes a low overhead hybrid hardware-software solution for managing it. Our experimental results indicate that our solution is able to achieve average energy savings of 30% at negligible overhead over conventional memory architectures.

Categories and Subject Descriptors

B.7.1 [Integrated Circuits]: Memory Technologies

General Terms

Design, Experimentation, Performance

Keywords

Phase Change Memory, Energy Efficiency, Memory management

1. INTRODUCTION

Power consumption is a major concern in the design of modern systems. Most of the recent research in power management has focused more on dynamic management of CPU power consumption, assuming it to be the most dominant contributor to system power consumption. However, recent studies [1] have shown that in modern systems CPU is no longer the primary energy consumer. Main memory has become a significant energy consumer, contributing to as much as 30-40% of total consumption on modern server systems.

In this paper, we propose a new approach for tackling the high levels of energy dissipation in main memory while minimizing the impact on performance. We introduce a new heterogeneous organization for main memory that is composed of DRAM and PRAM memories. The properties of PRAM that we leverage are its lower read access and standby power compared to DRAM while having a comparable throughput. However, the primary challenges in using PRAM include its lower write endurance (typical mean time to

failure in the range of $10^9 - 10^{12}$ cycles), and the higher power cost of write accesses compared to DRAM. These properties motivate the use of a heterogeneous memory architecture consisting of both DRAM and PRAM, which we refer to as PDRAM, enabling exploitation of positive aspects of the respective memories.

To manage the PDRAM memory organization, we propose a hybrid hardware/software solution. In order to maintain reliability for PRAM (because of write endurance problem), we introduce a cost efficient book keeping hardware technique that stores the frequency of writes to PRAM at a page level granularity. On the software side, we propose an efficient operating system (OS) level page manager that utilizes the write frequency information provided by the hardware to perform uniform wear leveling across all the PRAM pages. Wear leveling refers to the process of prolonging the lifetime of erasable storage devices with endurance problems (like Flash, PRAM etc) by ensuring uniform usage/utilization of all the storage blocks/pages of the device. Furthermore, the page manager intelligently allocates/migrates pages across DRAM/PRAM in order to minimize the impact of wear leveling on performance. The benefits of using such a hybrid approach is that the hardware can maintain and track page level accesses at a very low cost, while the software (OS) can leverage the high level observability and policies for management of free pages across DRAM/PRAM. We experiment across benchmarks with varying memory access characteristics, and show that using the PDRAM organization, we can achieve as high as 37% energy savings at negligible performance overhead over comparable DRAM organizations. Furthermore, we show that it achieves better energy and performance efficiency compared to homogeneous PRAM based memory systems as well.

Overall, the *primary contributions* of our work are: 1) We outline and evaluate the challenges in incorporating PRAM as an alternative main memory technology. 2) We propose an architecture and system policies for managing a PRAM/DRAM based main memory system. 3) We present a thorough evaluation and discussion of the system's performance, benefits and overhead.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the related work. In Section 3, we explain our memory organization and the solution to manage it for higher energy efficiency, reliability and performance. In Section 4, we describe the experimental evaluation and results before concluding in Section 5.

2. RELATED WORK

Existing memory power management research has primarily focused on DRAM based systems. In [12], the authors propose power aware page allocation algorithms for DRAM power management. They assume support in memory controller for fine grained bank level power control, and show that their allocation algorithms give

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC'09, July 26-31, 2009, San Francisco, California, USA
Copyright 2009 ACM 978-1-60558-497-3/09/07....10.00

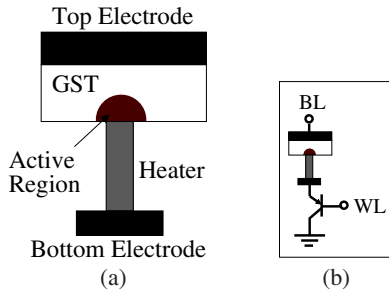


Figure 1: PRAM cell (a) and transistor (b)

greater opportunities for placing memory in low power modes. In [5, 9], the authors propose an OS level approach, where the OS maintains tables that map processes onto the memory banks they have their memory allocated in. This allows the OS to dynamically move unutilized DRAM banks into low power modes.

The possible use of alternative memory technology for improving energy efficiency has also been explored before. The authors in [10] propose NAND flash based page cache, which reduces the amount of DRAM required for system memory. This results in energy efficiency due to lower power consumption and higher density of NAND flash compared to DRAM. However, this approach is beneficial for more disk intensive applications, since flash is used only for page cache. In addition, flash also has endurance problems in terms of number of write cycles, which is tackled using wear leveling in the flash translation layer [14].

PRAM is an attractive alternative to flash, since: 1) Its endurance is higher by several orders of magnitude, 2) It is a RAM, and hence does not require an overhead of an erase before a write. In [13], the authors evaluate the challenges involved in architecting PRAM as a DRAM alternative. The authors in [15] propose a hybrid cache, that is composed of PRAM and SRAM for power savings. To solve the endurance problem of PRAM, they set a threshold on number of writes to the PRAM cache lines, beyond which they do not use those lines. However, the concern is that the typical level of conflicts and activity in cache could create reliability problems very soon on such a configuration. Extensive research has been done in modeling and understanding the basic characteristics of this technology [8, 11, 17–19].

3. DESIGN

3.1 PRAM/DRAM Background

The DRAM memory is organized as a grid of rows and columns, where each bit is stored in the form of charge in a small capacitor. As the charge can get exhausted due to leakage and frequent accesses, DRAM requires a consistent refresh operation to sustain its data. This results in a constant power consumption referred to as the *refresh* power. DRAM further consumes power for setting up its row and column for a physical address accessed, and also for closing a row if some other row needs to be accessed. This is referred to as the *activation/precharge* power. Additionally, it consumes power for the actual read/write accesses, and consistent standby power due to leakage and clock supply.

Unlike DRAM, PRAM is designed to retain its data even when the power is turned off. A PRAM cell stores information permanently in the form of the cell material state, which can be amorphous (low electrical conductivity) or crystalline (high electrical conductivity). A PRAM cell typically comprises of a chalcogenide alloy material (eg. $Ge_2Sb_2Te_5$ (GST)) and a small heater as shown

in Figure 1a. The cell can be addressed using a selection transistor (MOS or BJT) that is connected to the word-lines (WL) and bit-lines (BL) as illustrated in Figure 1b. To write to a PRAM cell, the GST state needs to be altered by injecting a large but fast current pulse (few 100ns) to heat up the GST active region. Consequently, PRAM write power is high compared to DRAM. For reading a PRAM cell, the power consumption is much lower, since no heating is involved. It is also lower than DRAM cell read power based on the measurements shown in [17]. Besides, PRAM consumes no refresh power, as it retains its information permanently, and [17] shows that it consumes much lower standby power due to its negligible leakage. However, PRAM has limited write endurance ($10^9 - 10^{12}$ cycles), which poses a reliability problem. Regarding access times, the access latency of random reads/writes on PRAM is slower compared to DRAM, although their read throughput is comparable. Thus considering all of these factors, PRAM is a promising candidate for energy savings because of its low read and standby power compared to DRAM.

3.2 Architecture

Based on the PRAM/DRAM characteristics described above, we observe that both the technologies have their respective pros and cons. This motivates us to propose a hybrid memory architecture, which consists of both DRAM and PRAM (PDRAM), for achieving higher energy efficiency. While PRAM provides low read and standby power, DRAM provides higher write endurance and lower write power.

The primary design challenge in managing a PDRAM system is to manage efficient wear leveling of PRAM pages to ensure its longer lifetime. For this purpose, we provide a hybrid hardware-software based solution. The hardware portion is based in the memory controller and manages the access information to different PRAM pages. The software portion is part of the operating system (OS) memory manager (referred to as the page manager), which performs the wear leveling by page swapping/migration. The components of the solution are described in detail below.

3.2.1 Memory controller

Figure 2 illustrates the various components and interactions of the PDRAM memory controller. The memory controller is aware of the partitioning of system memory between DRAM and PRAM. Based on the address being accessed, it is able to route requests to the required memory. To help wear leveling the PRAM, it maintains a map (access map in Figure 2) of the number of write accesses to it. This information is kept at a page level granularity, which is a function of the processor being used. For instance, the page size is 4KB for x86, 8KB for Alpha etc. We use page level granularity, since it is the unit of memory management, i.e. allocation and deallocation in the OS. If the number of writes to any PRAM page exceed a given threshold, then the controller generates a ‘page swap’ interrupt to the processor, and provides the page address. The OS then assumes the responsibility of handling this interrupt and performing page swapping as described in the next sub-section. The controller stores the map in the PRAM, for which it reserves space during bootup. The access map is maintained for the lifetime of the system, and after the first page swap interrupt, future interrupts are generated whenever the write access count becomes a multiple of the threshold. To maintain the map across reboots, it is stored on disk before the shutdown, and copied back into PRAM during the startup. To protect this data against crashes, it is synced with the disk periodically. If the write count for a page reaches the endurance limit (10^9 in the worst case), the controller generates a ‘bad-page’ interrupt for that page. This interrupt is also

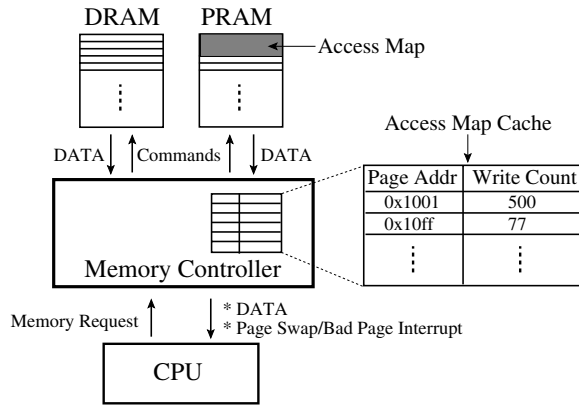


Figure 2: PDRAM Memory controller

handled by the page manager as described in the next section.

The enhancement of the controller incurs energy and memory overhead: 1) time and energy for updating PRAM access map; this involves extra accesses to PRAM, which causes extra power consumption. To avoid this overhead we introduce a small SRAM based cache in the controller (see Figure 2), which caches the updates to the map, hence reducing the consequent number of PRAM accesses. 2) memory overhead for storing the access map; the amount of memory required is proportional to the size of the PRAM used, and the size of the entry stored for each physical page in the access map. For instance, if the PRAM used is 4GB in size and the page size is 8K, the memory required for the access map would be around 4MB (each entry = 8 bytes). This is fairly small for modern systems, which have multiple GBs of memory.

3.2.2 Page Manager

The page manager is the OS level entity responsible for managing memory pages across PRAM/DRAM. It consists of two key subsystems, described below in detail, which help it perform its key tasks: Memory Allocator (page allocation/deallocation) and Page Swapper (uniform wear leveling of PRAM).

Memory Allocator. The goal of the OS memory allocator is to serve the memory allocation requests from the OS and the user processes. A list of free pages is maintained by typical allocators, from which the page request is fulfilled. Traditional memory allocators instantly mark the pages released by applications as free, and may immediately allocate them on subsequent requests. Such an approach can create hot spots of pages in memory, where the activity (reads/writes) is significantly higher compared to rest of the pages. This is fine for memories (like DRAM) that do not face endurance problems, but for memories like PRAM, it is a problem, as it may render some pages unusable very soon. To get around it, we make the PRAM memory allocator aware of these issues. The PRAM allocator maintains three lists of free pages: *free*, *used-free* and *threshold-free* list. At the startup time, all PRAM pages are in the *free* list, and the allocation requests from the applications are served from it. When the pages are freed, they go to the *used-free* list rather than the free list. If an allocated page is written to a lot by an application, and if the number of writes crosses a ‘*threshold*’, then as described before, the memory controller generates a page swap interrupt for that page. At this point, the page swapper (described below) handles the interrupt, and releases this page, which goes to the *threshold-free* list.

When the free-list becomes exhausted, or it lacks sufficient pages

to service a request, it is merged with the used-free list, which is then used as the source of page allocation. The user-free list will get exhausted only when all the free PRAM pages have been written to at least a ‘*threshold*’ number of times. When this happens, the *free* and *threshold-free* list pointers are swapped to move all the free pages from the *threshold-free* list to the original *free* list. Such an approach tries to achieve wear leveling across all the PRAM pages by ensuring that all the free pages have been written to at least ‘*threshold*’ times before getting reused. We assume that there is a separate allocator for DRAM memory, which does not need any such changes.

Page Swapper. The page swapper is responsible for managing the page swap and bad-page interrupts generated by the memory controller. It handles the page-swap interrupt by doing the following: 1) Allocates a new page from the memory allocator. The new page could be either from the PRAM or DRAM. 2) Finds the page table entry/entries (PTE/PTEs) of the physical page for which the interrupt is generated. This can be accomplished in modern systems such as Linux using the reverse mapping (RMAP), which maintains a linked list containing pointers to the page table entries (PTEs) of every process currently mapping a given physical page. 3) Copies the contents of the old page to the new one using their virtual addresses. The advantage of using virtual address for the copy as opposed to something like DMA is that it results in coherent copying of data, which ensures that the new page gets the latest data. 4) Updates the PTE/PTEs derived from the RMAP with the new physical page address. 5) Replaces the TLB entries corresponding to the old PTE/PTEs. 6) Releases the old physical page. If it is a PRAM page, it goes to the threshold-free list as described above.

The key decision for the swapper is in selection of the new page for replacing the old PRAM page, for which interrupt is generated. We implement two policies for this decision:

Uniform memory policy: This policy allocates the new page from the PRAM allocator. We introduce this as our baseline policy, which can be used in a PRAM only memory configuration as well, since it allocates only PRAM pages. This policy exploits the wear leveling mechanism of page swapping to extend PRAM endurance, but does not benefit from the memory heterogeneity of a PDRAM system.

Hybrid memory policy: This policy allocates new page from the DRAM allocator. The motivation to do so is based on the fact that there is a high probability of the page, for which page swap interrupt got generated, being very write intensive. As we show later, this has a two fold advantage: a) It reduces number of page swap interrupts, which is good for performance; b) It reduces number of writes in PRAM, which is good from perspective of both reliability and power, since PRAM writes consume higher power than DRAM writes. This policy exploits the heterogeneity of the PDRAM system, hence the name hybrid memory policy.

For the bad-page interrupt, the page swapper does exactly the same things as it does for the page-swap interrupt, except that it moves the page off its free lists and moves it into a bad-page list. The pages in the bad-page list are discarded and not used for future allocations. The list is stored reliably in a known location on PRAM.

3.3 Endurance Analysis

In this section, we analyze PRAM reliability with and without our wear leveling policies. Lets assume a system with pages of size 4K (eg. x86 systems) and 4GB of PRAM, implying there are $N_p=1M$ pages available for allocation. We assume an application, which consistently writes to two different addresses in PRAM, that

map to different PRAM rows in the same memory bank. This ensures that each row is consistently written back to the PRAM cells with alternative writes, because at a time only one row in a bank can be open. Typical latencies for such writes in PRAM is around 150ns (T_w). We assume that this application writes to these two addresses every 150ns or T_w in an alternative fashion to generate the worst case from endurance perspective. Note that this is unrealistic, since there are caches and write buffers between the CPU and memory, which will throttle the rate of writes. However, the analysis will allow us to estimate PRAM reliability under extreme cases. Lets refer to the write endurance of PRAM (10^9 write cycles in worst case) as N_w . If we do not take any wear leveling into account, then such an application can cause row failure in the page containing these addresses in $2N_w$ writes or $T_w \times 2N_w = 300s$. This is a very low time scale, and hence not acceptable for a real system deployment.

Now we analyze the case with our uniform memory wear leveling policy. Let the threshold of writes, at which the memory controller generates a page swap interrupt, to be N_t (where $N_t \ll N_w$), and the % of free PRAM pages in the system be $\alpha\%$. Lets assume, that the PRAM rows containing the two addresses being written to by the application map to the same physical page. Now as soon as the application writes N_t times, the page swapper will swap the physical page mapping these two addresses to a new PRAM page, which will correspond to different rows in the PRAM. The old physical page will then be moved into the *threshold-free* list. From our discussion in the previous section, we know that the application will not be able to write to the freed physical page (and its corresponding PRAM rows) again until both the *free* and *used-free* lists become empty and the *threshold-free* and *free* list pointers are swapped. For this to happen, the application will have to write at least N_t times to every free PRAM page ($\alpha N_p N_t$ writes) before it can access the old physical page again. In other words, to write $2N_t$ times to the old physical page, the application will do $\alpha N_p N_t + 2N_t$ writes, i.e.:

$$\begin{aligned} 2N_t &\rightarrow \alpha N_p N_t + 2N_t \\ 3N_t &\rightarrow 2\alpha N_p N_t + 3N_t \\ \beta N_t &\rightarrow (\beta - 1)\alpha N_p N_t + \beta N_t \end{aligned} \quad (1)$$

This means that for doing N_w writes ($\beta N_t = N_w$ in equation 1):

$$\begin{aligned} N_w &\rightarrow \left(\frac{N_w}{N_t} - 1\right)\alpha N_p N_t + \left(\frac{N_w}{N_t}\right)N_t \\ N_w &\rightarrow \alpha N_w N_p + N_w \approx \alpha N_w N_p \quad (N_w \gg N_t; \alpha N_w N_p \gg N_w) \end{aligned} \quad (2)$$

Based on this analysis, our application will have to do approximately $\alpha N_w N_p = \alpha 10^{15}$ writes in order to perform N_w writes to a given physical page. This means, to write N_w times to a PRAM row, it will have to perform $2N_w$ writes. Using T_w as 150ns (see the paragraph above) and $\alpha = 50\%$, this translates to around 2.4 years. Thus, with our wear leveling scheme the bounds go from order of seconds to years. It must be noted that this analysis has been done assuming an application which bypasses cache and write buffers to perform just writes. If we assume a conservative assumption of 50% cache hit for writes, the PRAM lifetime would increase to about 4.8 years. As the quality of PRAM is expected to increase to 10^{12} cycles and beyond, the bounds will be much higher. For the hybrid memory policy, the expected bounds would be even higher since the write intensive pages are moved to DRAM, where endurance is not an issue. In our experiments, we use $N_t=1000$ as a good trade-off between endurance and swapping cost.

Parameter	DRAM	PRAM
Power Characteristics		
Row read power	210 mW	78 mW
Row write power	195 mW	773 mW
Act Power	75 mW	25 mW
Standby Power	90 mW	45 mW
Refresh Power	4 mW	0 mW
Timing Characteristics		
Initial row read latency	15 ns	28 ns
Row write latency	22 ns	150 ns
Same row read/write latency	15 ns	15 ns

Table 1: DRAM and PRAM Characteristics (1Gb memory chip)

4. EVALUATION

4.1 Methodology

For our experimental evaluation we use the M5 architecture simulator [4]. M5 has a detailed DRAM based memory model, which we significantly enhance to model timing and power of a state of the art modern DDR3 SDRAM based on the data sheet of a Micron x8 1Gb DDR3 SDRAM running at 667MHz [7]. We implement a similar model for PRAM based on timing and power characteristics described in [13, 17–19]. We assume PRAM cells to be arranged in a grid of rows and columns just like DRAM. Such configuration of PRAM has been practically implemented and demonstrated by Intel [17].

The power parameters used for DRAM and PRAM are listed in Table 1. The DRAM parameters are based on 78nm technology [7]. The read-write power values for PRAM are obtained from the results in [2, 3, 13, 17]. For a fair comparison, the values are down scaled for 78nm technology based on the rules described in [16]. We can observe that read power of PRAM is around three times lower compared to DRAM, while write power is around four times more. This suggests that PRAM is not very attractive for write intensive applications both from the perspective of reliability as well as energy efficiency. The third parameter in Table 1 (Act) refers to the activation/precharge power, which is consumed in opening and closing a row in the memory array. It is higher for DRAM, since it has to refresh the row data before closing a row, which can be avoided in PRAM due to its non-volatility. The standby power, which the memory consumes when it is idle, is also lower for PRAM due to its negligible leakage power consumption. Finally, DRAM consumes refresh power for supplying sustained refresh cycles for it to retain its data. This is not required in PRAM, since it is non-volatile.

For timing, we use the Micron data sheet to get the detailed parameters for DRAM. For PRAM, we use the results in [2] to obtain the read-write latency values for 180nm technology. We scale down the read latency for 78nm, but maintain the same value for write as a conservative assumption, since the write latency is a function of the material property. Table 1 shows these values. We can observe, that for an initial read, PRAM requires almost twice the amount of time as compared to DRAM. This happens due to the higher row activation time of PRAM. Similarly, writing back or closing an open row in PRAM is around seven times more expensive than for DRAM. However, reads/writes on an open row have latency values similar to DRAM.

Besides this, we further extend M5 for incorporating the memory controller and page manager as described in section 3. We implement the access map cache in the memory controller (see section 3) as a 32 entry (each entry = 8 bytes) fully associative cache. For the page manager, we implement both the uniform and hybrid memory

Benchmark	rpi (%)	wpi (%)	# of pages
applu	1.94	0.93	24435
bzip2	0.12	0.08	24600
facerec	0.6	0.5	2240
gcc	0.15	0.06	2781
sixtrack	0.01	0.008	7601

Table 2: Benchmark Characteristics

policies as described in section 3.

For our experiments, we assume a baseline system with 4GB of DDR3 SDRAM, which we refer to as DRAM. The 4GB memory consists of four 1GB ranks, where each rank is made up of eight x8 1Gb chips with characteristics described in Table 1. We evaluate it against two experimental systems: 1) **Hybrid system**: It comprises of 1GB DDR3 SDRAM and 3GB of PRAM, and employs *hybrid memory policy* for managing page swap requests. 2) **Uniform system**: It comprises of 4GB of PRAM, and employs *uniform memory policy* for managing page swap requests. The motivation of the comparison is to show how heterogeneity in memory organization can result in better overall performance and energy efficiency.

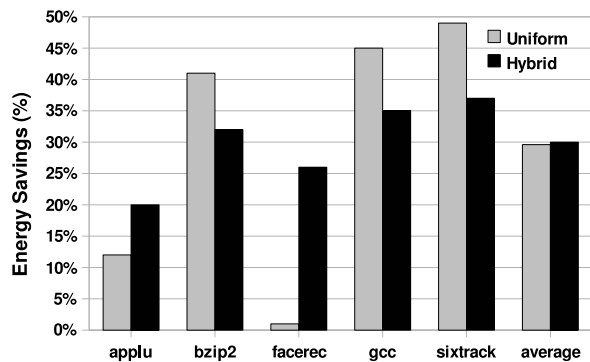
For workloads, we use benchmarks from the SPEC CPU2000 suite, which we execute on M5 using a detailed out-of-order execution ALPHA processor running at 2.66GHz. The simulated processor has two levels of caches: 64KB of data and instruction L1 caches, and 4MB of L2 cache. We use benchmarks described in Table 2, and simulate the first five billion instructions. Table 2 illustrates the memory access characteristics of these benchmarks in terms of rpi (reads per instruction), wpi (writes per instruction), and *number of pages* (total number of pages allocated). We can see that they have varying memory access characteristics. For instance, sixtrack has very low rpi (0.01%) and wpi (0.008%), while for applu it is an order of magnitude higher (1.94% and 0.93%); facerec has high rpi (0.6%) and wpi (0.5%), while gcc and bzip2 have medium rpi and low wpi . Similarly, the working set of these benchmarks in terms of the number of pages allocated also varies from just 2240 (facerec) to as high as 24600 (bzip2).

4.2 Results

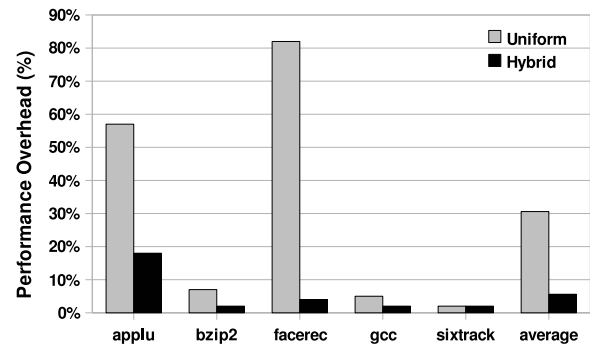
Energy Savings and Performance Overhead

Figure 3 shows the results of the hybrid and uniform memory systems baselined against the DRAM system for all the benchmarks. Figure 3b shows the overhead incurred in terms of execution time by these systems, while Figure 3a shows the reduction in memory energy consumption. Please note that the %overhead and %energy numbers in these figures include the energy and time overhead due to page migrations and accesses to the access map in the memory controller and the PRAM. We describe the details of the overhead in the following sections.

We can see in Figure 3, that on average, the hybrid system achieves around 30% energy savings for just 6% performance overhead across all the benchmarks. In contrast, the uniform system gets 30% energy savings at the cost of 31% overhead. For sixtrack, which has low rpi and wpi , the impact on performance is negligible since there are not enough accesses to expose the slower access times of PRAM. Both systems achieve high energy savings due to the lower standby power consumption of PRAM compared to DRAM (see Table 1). The energy savings for the uniform system is higher (around 49%) than hybrid system (37%) since the uniform system comprises exclusively of PRAM, while the hybrid system contains 1GB of DRAM as described in section 4.1. For gcc and bzip2, the performance overhead is higher for the uniform system (around 6%). This happens due to the relatively higher rpi and wpi of these



(a) Energy Savings



(b) Performance Overhead

Figure 3: Energy Savings and Performance Overhead Results

benchmarks compared to sixtrack (see Table 2), which exposes the slower access times of the PRAM. The overhead is lower for the hybrid system (2%), since it migrates the write intensive pages to DRAM. The energy savings is higher again for the uniform system due to the lower standby power of PRAM.

In contrast, for applu and facerec, the performance overhead of the uniform system is significantly high (57% and 82% respectively). This happens due to the higher wpi and rpi of both these benchmarks (see Table 2). The overhead for facerec is higher than applu (despite its lower rpi and wpi) due to its higher IPC (60% more than applu). This implies, facerec is more sensitive to higher memory access latencies, which results in the poor performance of the uniform system due to slower access times of PRAM. The high overhead nullifies the lower power consumption of PRAM and results in negligible energy savings. In contrast, the performance overhead of the hybrid system is very low (4%). This happens because facerec has a very high locality of read and writes to its pages, and the hybrid system migrates them to DRAM. This results in much higher energy savings as well (26%). For applu, the energy savings for the uniform system is low (around 10%) due to its high performance overhead. For the hybrid system as well, the overhead is high (around 18%) because of low locality of reads and writes. However, the energy savings is still higher (20%) compared to uniform system.

Thus, the results indicate, that in terms of comparison between the systems, the hybrid system is clearly more beneficial. It is able to exploit the read-friendliness of PRAM as well as the write friendliness of DRAM, and hence achieves better overall performance and energy efficiency.

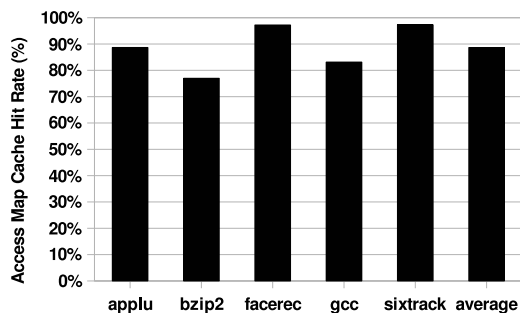


Figure 4: Access Map Cache Hit Rate (%)

Benchmark	Uniform	Hybrid	% Reduction
applu	29000	18000	38
bzip2	2400	635	74
facerec	24600	1500	94
gcc	1350	320	77
sixtrack	0	0	0

Table 3: Page Swap Interrupts

System Overhead

In this discussion, we analyze the sources of system overhead and their impact in terms of performance and energy in detail. We focus on the overhead due to accesses to the access map and its cache, and page swapping.

Access Map

As described in section 3, the access map is used to store write access information to PRAM pages. The updates to the access map and its memory controller cache thus add energy overhead in the system in terms of extra accesses to PRAM and the power consumption of the cache itself.

For understanding the extra accesses to PRAM, Figure 4 shows the hit rate of the access map cache. The hit rate was almost the same for both the hybrid and uniform systems. We can see that the hit rate is fairly high across all the benchmarks (average around 90%). This indicates that the overhead is very low, since extra PRAM accesses are done only for 10% of writes. From Table 2, we know that *wpi* of most of the benchmarks is fairly low, so in the context of overall time frame, the impact is negligible.

For the access map cache, we estimate the power consumption to be around 78mW per access using CACTI 4.1 [6]. Since the cache is accessed only for writes, in the overall time-frame, the extra energy consumption due to it is also very low. It must be noted that the extra energy consumption due to accesses to the access map and its cache is included in the results in Figure 3.

Page Swapping

The second source of overhead is page swapping interrupts and the consequent page swaps. Table 3 shows the statistics related to page swap interrupts for the uniform and hybrid system. We can observe that for most of the benchmarks, the number of interrupts drop significantly for the hybrid system. For instance, for facerec, it drops down by as much as 94%. This happens due to its high *wpi* and significant locality in its reads and writes to a small set of addresses. In the uniform system, when the pages mapping these addresses reach the *threshold*, they get mapped to a new PRAM page. However, sustained writes to these addresses generate further page swap interrupts. In contrast, with the hybrid system, once the pages mapping these addresses reach the threshold, they are mapped to DRAM pages, where they no longer generate any further

page swap interrupts. For applu, the reduction is smaller (37%) due to its bigger working set, and relatively lower lack of locality of reads-writes.

In terms of time overhead of a page swap, it is fairly low since it is a quick software operation of allocating and copying the page, and modifying the page table entries. We assume it to be $5\mu s$, which is based on the estimate of timer interrupt overhead in modern systems that get generated as frequently as 1ms. Hence, overall the page swapping overhead is also minimal in terms of performance and energy. Note, the page swap overhead is also included in the results in Figure 3.

5. CONCLUSION

In this paper we propose PDRAM, a novel, energy efficient hybrid main memory system based on PRAM and DRAM. We highlight the challenges involved in managing such a system, and provide a hardware/software based solution for it. We evaluate the system using benchmarks with varying memory access characteristics and demonstrate that the system can achieve up to 37% energy savings at negligible overhead. Furthermore, we show that it provides better overall energy and performance efficiency compared to homogeneous PRAM based memory systems as well.

6. REFERENCES

- [1] L. A. Barroso et al. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.
- [2] F. Bedeschi et al. An 8Mb demonstrator for high-density 1.8V phase-change memories. In *Symposium on VLSI Circuits*, pages 442–445, 2004.
- [3] F. Bedeschi et al. A multi-level-cell bipolar-selected phase-change memory. *Proc ISSCC'08*, pages 427–429, 2008.
- [4] N. L. Binkert et al. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.
- [5] V. Delaluz et al. Scheduler-based dram energy management. In *Proc DAC'02*, pages 697–702, 2002.
- [6] <http://www.hpl.hp.com/research/cacti/>.
- [7] <http://www.micron.com/products/dram/ddr3/>.
- [8] <http://www.numonyx.com/Documents/WhitePapers>.
- [9] H. Huang et al. Design and implementation of power-aware virtual memory. In *Proc ATEC'03*, 2003.
- [10] T. Kgil et al. Improving nand flash based disk caches. In *Proc ISCA '08*, pages 327–338, 2008.
- [11] A. Lacaita et al. Status and challenges of pcm modeling. *Solid State Device Research Conference, 2007. ESSDERC 2007. 37th European*, pages 214–221, Sept. 2007.
- [12] A. R. Lebeck et al. Power aware page allocation. *SIGOPS Oper. Syst. Rev.*, 34(5):105–116, 2000.
- [13] B. Lee et al. Architecting phase change memory as a scalable dram alternative. *Proc ISCA'09*, 2009.
- [14] H.-L. Li et al. Energy-aware flash memory management in virtual memory system. *IEEE Trans. Very Large Scale Integr. Syst.*, 16(8):952–964, 2008.
- [15] P. Mangalagiri et al. A low-power phase change memory based hybrid cache architecture. In *Proc. GLSVLSI '08*, pages 395–398, 2008.
- [16] A. Pirovano et al. Scaling analysis of phase-change memory technology. *Proc IEDM'08*, pages 29.6.1–29.6.4, 2003.
- [17] A. Pirovano et al. Phase-change memory technology with self-aligned μ trench cell architecture for 90nm node and beyond. *Solid-State Electronics*, 52(9):1467–1472, 2008.
- [18] N. Takaura et al. A GeSbTe phase-change memory cell featuring a tungsten heater electrode for low-power, highly stable, and short-read-cycle operations. *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, pages 37.2.1–37.2.4, Dec. 2003.
- [19] B. Yu et al. Chalcogenide-nanowire-based phase change memory. *IEEE Transactions on Nanotechnology*, 7(4), 2008.