[redacted]

A Study of Page Placement and Migration in Heterogeneous Flat-Addressable Memories

ABSTRACT

The volume of data generated by research, commercial, industrial, communication, entertainment and other fields is growing exponentially. There is a need for faster and very large amounts of main memory for analyzing such volumes of data in reasonable amounts of time. In addition, these systems need to be as energy efficient as possible, since the energy requirements of most high-performance computing systems and data centers are becoming significant portions of their operational budgets. This led to research into different memory technologies such as 3D-stacked DRAM, improvements to DDR, and non-volatile memories like PCM, and Flash memories. There have been some studies on how these memory technologies can be used to address the need for very large amount of main memory at reasonable cost and energy budgets. Flat-addressable memories differ from hierarchical view of the memory system: they are organized as a single (flat) physical address space with two or more types of memory devices, each with its own latencies and bandwidth. In such memories the page placement and migration (or swapping) of pages across these different memory devices requires very careful analysis. Previous studies have explored simple migration policies in a system with 3D-DRAM and DDR4 as main memory. Our analysis shows that for some workloads, static page placement with no page migration outperforms policies that consider only page access counts in deciding which pages to migrate. Additionally, these policies may prove to be inefficient for memory systems when PCM is included. In this paper we present and evaluate several intelligent and efficient policies for migration of physical pages across the memory technologies. We study our policies for two level (3D-DRAM + DDR4; and 3D-DRAM + PCM) and three level (3D-DRAM + DDR4 + PCM) memory systems. We present both performance improvements and memory energy savings for our page migration policies. Compared to previous studies, we observe average speedups of 2.6% and energy savings of 65.9% for 3D-DRAM + DDR4, average speedups of 10% and energy savings of 76.8% for 3D-DRAM + PCM, and average speedups of 8% and energy savings of 68.5% for 3D-DRAM + DDR4 + PCM.

Key Words: Heterogeneous memories, 3D-DRAM, DDR4, Phase Change Memory, Flat physical address space, Page migration

1. INTRODUCTION

The volume of data generated by research, commercial, industrial, communication, entertainment and other fields is growing exponentially. There is a need for faster and very large amounts of main memory for analyzing such volumes of data in reasonable amounts of time. In addition, these memory systems need to be as energy efficient as possible, since the energy requirements of high performance and data centers are becoming significant portions of their operational budgets.

DDR DRAM is the primary technology used as the main memory in today's computing systems. However, capacity, power, latency and bandwidth requirements of emerging applications cannot be met by DDR memories. It has been reported that today's conventional DRAM main memory systems consume up to 40% of the total system power [1, 2]. The limited scalability and high cost-per-bit make the design of large-scale DRAM memory systems infeasible [3, 4]. In fact, no single memory technology is likely to meet these needs [5]. Thus, we must search for new memory technologies and architectural solutions that combine different technologies into a single main memory to meet these needs.

On the performance end, 3D-stacked DRAM (3D-DRAM) provides high bandwidth and low latency at higher cost and limited capacity [6, 7]. On the other hand, non-volatile memories (NVM) are promising with lower cost-per-bit and higher capacity but at reduced performance. The most promising non-volatile memories, in the foreseeable future, are NAND flash memories and phase change memories (PCM) [8]. The flash based memories are nearly 1000 times slower than DDR [9] and may not be suitable as main memories, although some studies proposed using NAND flash as main memories with DRAM caches of various sort [10]. PCM is denser than DRAM, and even though it is slower than DDR DRAM, it is still significantly faster than NAND flash memories. This led to studies using PCM as the main memory, either to augment DDR or to replace DDR memories [1, 11, 12, 13, 4, 14, 15, 16, 17, 3]. The recently announced 3D XPoint non-volatile memory is already in the limelight with the claim of being 1000 times faster than standard NAND flash memory [18]. Although details are not readily available, the community assumes that 3D XPoint is based on PCM technology.

1.1 Motivation

Given these various technologies, it is appealing to consider a memory system built using more than one type of memory technology. There are many choices for constructing such a heterogeneous main memory system. A flat addressable heterogeneous memory differs from a hierarchical memory system. A hierarchy implies that the faster memory is closer to the CPU and the slower memories are farther from the CPU, and data migrates on demand across these memories, often keeping replicas. A flat-addressable heterogeneous memory is composed of two or more types of memory devices, each with its own latency and bandwidth characteristics; and the entire memory is viewed as a single contiguous physical address space, albeit with non-uniform access times. It might be beneficial to migrate (or swap) pages across the memory devices in order to decrease the memory access time for frequently used data and improve the applications' execution time. The page migration policies can be enforced by the runtime system with simple hardware support such that application transparency is ensured [19].

However, the page placement and migration (swapping) of pages in flat heterogeneous memory systems require very careful analyses. Previous study have explored simple migration policies at a page granularity in a system with 3D-DRAM + DDR4 as the main memory [19]. *Our analyses show that for some workloads, static page placement with no page migration outperforms policies that consider only page access counts in deciding which pages to migrate.* Additionally, such simple migration policies may prove to be inefficient for memory systems when PCM is included. In this paper we present and evaluate several policies for migration of physical pages across multiple memory technologies within a single flat address space.

We study migration policies for two level (3D-DRAM + DDR4; and 3D-DRAM + PCM) and three level (3D-DRAM + DDR4 + PCM) memory systems. We present both performance improvements and energy savings for our policies. We use both single and multi-programmed workloads drawn from SPEC 2006. More specifically:

- We propose and evaluate intelligent page migration policies which take into account the post-migration usage of pages (i.e., will a page remain hot after migration or becomes cold soon after migration) as well as the relative differences in the latencies and bandwidths of the memory devices (i.e., how hot a PCM or DDR page should be relative to a page in 3D-DRM before swapping the pages) in determining which pages should be migrated. We show that these policies result in performance gains and reduce energy consumed by applications.

- We catalog the key sources of overhead associated with page migration. We propose and evaluate policies which place a limit on the total number of pages transferred at each epoch in order to balance the page migration overheads against performance gains.

- We also explore locking last-level cache (LLC) lines for heavily accessed data items as an additional optimization and as an alternative to page migration.

- We compare our policies with the Hot Page Policy published in [19], as well as to a heterogeneous memory system that does not migrate pages. In addition, we compare our results to systems with sufficiently large 3D-DRAM only and DDR4 only memories.

In a nutshell, compared to previous studies, we observe average speedups of 2.6% and energy savings of 65.9% for 3D-DRAM + DDR4, average speedups of 10% and energy savings of 76.8% for 3D-DRAM + PCM, and average speedups of 8% and energy savings of 68.5% for 3D-DRAM + DDR4 + PCM.

Our intention is not to show the benefits of PCM (when used as part of main memory) in reducing the number of hard page-faults, but rather to study the performance improvement achieved when PCMs are used as part of a multilevel main memory system. Therefore, we assume that the total capacity of a multilevel system is sufficient to contain the memory footprints of the benchmarks studied.

2. BACKGROUND AND RELATED STUDY

2.1 Introduction to Memory Technologies and Baseline Architecture of Heterogeneous Memory Systems

3D-stacked DRAM (3D-DRAM) is a new memory technology that uses die-stacking technology. 3D-DRAM has lower access latency, higher bandwidth and lower access energy per-bit as compared to today's DDR4 [20, 6]. Mircron's Hybrid Memory Cube (HMC) comes with 4GB capacity and bandwidth of 320GB/sec and HMCs can be chained together using high speed links [7]. Also, JEDEC standard stacked High Bandwidth Memory (HBM) [21] can provide a bandwidth of 128GB/s per stack.

Phase change memory (PCM) is a non-volatile memory (NVM) that relies on the state or phase of material that changes its phases from amorphous to solid. Each PCM cell may hold one bit (single-level cell, SLC) or represent multiple bits (multi-level cell, MLC). This choice leads to differences in latencies, energy and bandwidths of PCMs [3]. PCM may exhibit higher write latency (4x-32x) and higher read (1.2x-2x) and write energies (4x-140x) than DDR4 [22, 3]. PCM has limited write endurance of 10^6 to 10^8 cycles. On the other hand, PCM can be 4 times denser than DDR4 depending on the implementation [13] and hence will provide larger capacity and lower cost-per-bit [3]. Addressing limitations in terms of write latencies, endurance and energy is an ongoing research area (see for example [23, 24, 13, 17, 25]).

Recently, Intel and Micron jointly announced the **3D XPoint**, which is a non-volatile memory, and claims to be 1000 times faster than NAND flash memories and 10 times denser than DDR memories [18]. It has been mentioned that 3D XPoint stores data by the change of the memory cell material itself. [1] Our study should be applicable to this new device since we explore a wide range of design space for NVMs in our research.

Heterogeneous Memory Architecture (HMA) system [19] is the basis for our study. Only 3D-DRAM + DDR4

[1]The research community believes that the memory cells will be some version of PCM and therefore latency and bandwidth problems could still persist.

was used in that architecture. We extend the HMA system by adding PCM memories. Both studies assume a single contiguous physical address space spanning different memory devices and assume necessary hardware and software to access appropriate device for a given data request.

As in the previous study [19], we also assume that program execution is divided into fixed intervals or epochs of 0.1 seconds. The number of accesses to a page (i.e. number of LLC misses) is recorded per epoch by hardware and stored within page table entries (PTE). The memory reference count serves as an indicator of how frequently each page is accessed in a given epoch, and if the count exceeds a threshold (say 32) the page is considered *hot*. In the Hot Page Policy [19], after each epoch, *hot* pages are sorted (if necessary) based on their access counts, and top *hot* pages (based on available space in the 3D-DRAM) are migrated from DDR4 to 3D-DRAM. Thus program execution is halted during the migration and may cause performance penalties which may be overcome by faster access time to *hot* pages. Additional polices such as FTHP, which maximizes the number of pages migrated, and FTHP-HB, FTHP-FB, which minimize the hardware for tracking access counts, were reported in [19].

We extend the HMA study by evaluating their page migration policies for homogeneous multicore environments (instead of heterogeneous multicore systems). We propose and evaluate several new policies that take into account the post-migration usage of pages, limit the number of pages migrated at each epoch, and change the hotness threshold based on the device where a page currently resides. In addition we compare the policies with no page migration. We also investigate locking LLC lines of *hot* pages as a possible optimization and as an alternative to page migration. Cache Locking is a technique that has been used commonly to lock cache lines to improve timing predictability of Real Time Systems [26, 27]. Here we employ this technique from a different perspective, i.e. to reduce memory traffic between LLC and main memory system.

2.2 Other Related Work

Heterogeneous main memory systems can be broadly categorized in two classes depending on the allotment of physical address space. One category can be viewed as hybrid memory, where each of the different memory technologies used in the system main memory are assigned to a single physical address space. Different studies have shown that with intelligent page/data allocation and migration techniques, hybrid memory systems comprising of different types of DRAM [28], 3D-DRAM+DDR [29, 19], and DRAM + PCM [1, 11, 12] may provide overall performance gains and energy savings compared to conventional homogeneous main memory systems. The other category can be called hierarchical memory, where one or more technologies used in the main memory systems serve as cache/buffer for the other memory devices used in the heterogeneous system. The memory devices used as buffer/cache are not visible in the physical address space [13, 4, 14, 15, 16].

3. PAGE MIGRATION IN HETEROGENEOUS MEMORY

As previously indicated, emerging workloads including HPC and Big Data applications are requiring ever increasing amounts of main memory. Some of these applications do not benefit from large caches and memory hierarchies [30]. Large memory capacities cannot be satisfied solely by using DDR4 while being effective in terms of access latencies and within given power budgets. We feel that no single memory technology can address latency, bandwidth, capacity, and energy requirements of emerging applications. New memory technologies are becoming available that may present solutions in addressing these needs. They include 3D-stacked DRAM with favorable latencies [29, 31] and higher bandwidth (five to tewleve) [21, 7] as compared to conventional DRAM, and PCM with higher densities [8] while being faster than flash memories. PCM may also have comparable read latencies to DDR memory but suffers from high write latencies (4x-32x higher than DRAM read latency) and high write energy [3]. PCM supports lower bandwidth than DDR4 (particularly write bandwidth) [3, 22]. This generated interest in using a combination of these memory devices together as a heterogeneous main memory. These memory devices are organized to represent a single (or flat) contiguous physical address space, albeit with non-uniform memory accesses.

3.1 Page migration issues

To improve access latencies of flat-addressable heterogeneous memories, it may be beneficial to migrate heavily accessed (*hot*) pages to faster memories such as the 3D-DRAM. The transfer incurs overhead both in terms of execution time and energy. The key challenge is to identify which pages should be migrated to the faster memory. The decision process should try to minimize the migration overheads (both in terms of time and energy) and maximize applications' performance. That is the focus of our research presented here.

Tracking page usage To determine which pages are *hot*, we count the number of access to pages (resulting from misses in LLC) over a given interval (or epoch). A threshold on the number of accesses is used to categorize pages as *hot* or cold. In [19] the threshold is set to 32 for a two memory memory system with 3D-DRAM + DDR4. The access counts are associated with pages, and may be stored with PTE entries, as proposed in [19, 12].

Page migration mechanism and overheads. In current implementation of memory systems, the migration of a page also changes its physical address, requiring changes to TLB and possibly invalidating cache memory entries (since they are physically tagged). Another issue to consider is the actual migration mechanism (how the pages are swapped/migrated): use a separate DMA channel, or use CPU (by executing kernel code) to read and write data of the page being migrated [1]. In the latter case, the entire cache hierarchy may be impacted, while in the former case, only the data from the swapped pages will be affected. In order to migrate a page, data needs to be read from one memory device and written to another one. This introduces substantial amount of energy consumption for the migration itself. The total time it takes to migrate the pages will be bound by the lower available bandwidth, while the energy consumption will depend on the total number of pages migrated and the access energy of the memories in the system. Additional overheads such as TLB shootdown, interrupt times, OS memory management, etc. also contribute to the total execution time. However, the main overhead contributor is the data migration itself. We quantify and discuss the overheads in Section 5.

Post-migration usage of pages. Another issue that affects the performance of any page migration policy is the post-migration usage of the pages. A page may become cold (not heavily accessed) immediately after being migrated to a faster memory. The result is that the benefits of faster memory accesses are outweighed by the cost of page migration (due to afformentioned overheads). We will see that for some multi-programmed workloads this is the case, since many migrated pages soon become cold. Thus, a more intelligent placement policy, which takes into account the probability of the transferred page remaining *hot* is needed.

Page usage versus device latencies. While transferring the pages from high latency, low bandwidth memory (such as PCM), the time it takes to migrate a page can be high. If we use a fixed hotness threshold, e.g. 32 accesses, for classifying a page being *hot* (accessed 32 times) will not suffice to hide the migration overhead. It is necessary to take into account the relative differences in the latencies and bandwidth of the memory devices, particularly when the differences are large. For example, it may not be worthwhile transferring a *hot* page from a slower memory to a faster memory unless the number of accesses of the *hot* page in the slower memory is two times greater than that of a page being displaced from the faster memory.

Limiting the number of pages migrated per epoch. The page transfer overheads may outweigh the benefits of migration when very large number of pages are transferred at each epoch, particularly in terms of energy overheads. Our studies have shown that in some cases the energy due to page transfers may account for nearly 50% of the total energy consumed by an application (see Section 5). Therefore, the number of pages migrated at each epoch should be limited in order to balance the migration overhead against performance benefits.

3.2 Page migration policies

Here we describe the various page migration policies that are evaluated in our study. These include newly proposed policies which attempt to overcome the aforementioned overheads.

NO_TRANSFER. We use this policy as our baseline. This policy assumes that the heterogenous memory is populated linearly, starting with faster memory and moving to slower memories once the capacity of the faster memory is exceded. Once assigned pages are not migrated across the devices. Meswani et. al. [19] did not evaluate such a heterogeneous memory system and we feel that it is a significant omission. Our analyses show that for some benchmarks, the performance is better when pages are statically allocated to the memory devices, and pages are not migrated, when compared to the BASIC_TRANSFER policy described below. Some SPEC2006 benchmarks tend to access mostly memory pages that were allocated earlier in the program execution and less frequently pages allocated later in the execution. In such cases, it is likely that heavily accessed pages are going to be assigned to faster memories and less frequently accessed pages to slower memories; and thus migration of pages may not be very beneficial.

BASIC_TRANSFER. It is the same as the Hot Page Policy in the HMA system [19]. Any page with more accesses than the defined threshold in a single epoch is considered *hot*. The threshold is set to 32 accesses in the HMA study after a sensitivity analysis, and we also use the same threshold. The *hot* pages are sorted based solely on the access counts and the top N *hot* pages are transferred to the fastest memory (3D-DRAM), where N is the total number of physical frames in the 3D-DRAM.

PRIORITY. We propose this policy in order to minimize the impact of migrating pages which are not likely to remain *hot* after the migration. In order to implement this policy, we need to acquire information about a page after migration; that is, will the page remain *hot* in the epoch(s) following the migration. This information can be used when considering a page for migration at a later time. We use a *moved* bit to see if a page was migrated in the previous ecpoch. Two additional bits are used to keep a count of the number times a page remained *hot* after previous migrations. The count is incremented if the *moved* bit is set and the page is *hot* in the current epoch. This implies that the migration was usefull since the page remained *hot* after its migration. The count is decremented if the *moved* bit is set and the page is not *hot* in the current epoch. The count ranges between 0-3, where 0 indicates that the page is not likely to be *hot* after migration and 3 indicates that the page is very likely going to remain *hot* after migration. This count is used to prioritize *hot* pages selected for migration (hence the name PRIORITY). Our results show that this policy performs better than the BASIC_TRANSFER policy. The *moved* bit and 2-bit count can be kept with PTEs or in separate data structure associated with pages.

Table 1: PRIORITY+ set different hotness thresholds for different memory devices. The thresholds depend on how many accesses are needed to a page in order to hide the migration overhead. NV_1x-8x are different configurations of Non-Volatile memories (find more detail in Sect. 4.2.1).

Memory Type	NV_1x	NV_2x	NV_4x	NV_8x
Latency (ns)	60	120	240	480
R BW (GB/s)	12.8	6.4	3.2	1.6
W BW (GB/s)	3.2	1.6	0.8	0.4
Accesses	80	40	32	29

PRIORITY+. sets hotness thresholds for different memories. The thresholds depend on how many accesses are needed to a page in order to hide the migration overhead. This policy takes into account the relative differences in latencies and bandwidths of the devices involved and scales the hotness threshold for pages residing in different memories. On top of the PRIORITY policy, we also consider the overhead of page migration, which differs for different memory configurations, and we calculate the number of accesses to that page which are needed to cover the overhead. For example, a page which is considered for migration from PCM needs 80 accesses in order to hide its overhead, whereas a page migrated from DDR4 needs only 20 accesses. Therefore, the policy enforces different hotness thresholds for different devices (and not a fixed value of 32 as in the BASIC_TRANSFER policy). A page is considered *hot* only if it exceeds the threshold set for the device where the page is currently located. The hotness thresholds can be set at the system startup, and do not have to change over time. Table 1 shows example values for thresholds. NV_1x-NV_8x means different NVMs with different read/write latencies and bandwidths relative to those of DDR's. These are different configurations of NVMs those we have analysed in our study (more detail in Sect. 4.2.1). The thresholds correspond to the number of accesses needed to hide the migration overhead which primarily depends on the memory parameters (latency and bandwidth). The hotness thresholds can be set at the system startup, and do not have to change over time.

3.3 PAGE MIGRATION LIMIT

Hard limit on the number of pages migrated. Selecting a subset of all *hot* pages for transfer minimizes the overheads and reduces energy consumed by the application. We can use a fixed limit on the number of pages migrated per epoch instead of allowing as many pages as the capacity of 3D-DRAM to migrate at each epoch. This improves scalability, especially from the energy point of view, because the total number of pages which can be migrated per epoch does not change with the 3D-DRAM size.

TLB migration limit. Unlike the previous case, we limit the number of *hot* pages transferred at each epoch to only those pages that currently have valid TLB entries (and also satisfy our priority criteria regarding their future usefulness). We feel that this further improves the effectiveness of migration policies because pages with valid TLB entries indicate their recency of accesses. In this study, we assume a per-core TLB with 512 entries for a total of 2048 entries in a 4-core system. Since each transfer involves 2 pages (one page moving to 3D-DRAM, displacing a 3D-DRAM page), we transfer at most 4096 pages at each epoch. Our experiments show that in most cases, approximately 1000 pages are transferred at each epoch using this policy.

3.4 LOCKING LLC LINES

Hot pages are frequently accessed in the main memory because they are evicted from the cache hierarchy. We can reduce the total number of memory accesses to these pages, and also improve their access latency, by locking the data from *hot* pages in the LLC. This may potentially improve the total execution time because the number of memory references will be reduced. Further, it may not be necessary to transfer pages from slower memories if their data is locked in the LLC. However, the locking cache lines may cause conflicts for the unlocked lines, increasing cache misses for other pages (potentially making them *hot* in future epochs). We explore this option as an additional improvement to the placement policies outlined above as well as an alternative to page migration.

4. METHODOLOGY

We ran our experiments using an open-source trace-driven cache simulator on top of which we developed a heterogeneous flat-addressable memory module in order to simulate accesses to the main memory. Below we describe our experimental setup in more details.

4.1 Trace Generation

We generated the execution traces from a server with two Intel Xeon-E2640 processors and 32GB of physical memory. We used the Pin tool [32] to generate the execution traces. We collected traces for 12 SPEC CPU2006 benchmarks, where 6 benchmarks are capacity limited and 6 are latency limited [29]. We simulated single and multi-programmed workloads. Table 2 shows the benchmarks and multi-programmed mixes used in our evaluations.

Table 2: Benchmarks and Memory Footprint

Benchmark	Memory Footprint (MB)
bwaves	930
omnetpp	162
cactusADM	628
soplex	522
gcc	64
xalancbmk	116
GemsFDTD	831
zeusmp	504
lbm	412
mcf	1679
libquantum	98
milc	569
SMALL (2xgcc, 2xlibq)	317
MEDIUM (omnetpp, xalanc, gcc, libquantum)	433
LARGE (lbm, milc, soplex, zeusmp)	2001
VERY_LARGE (mcf, bwaves, GemsFDTD, cactusADM)	4062

4.2 Memory System Simulator

We developed a heterogeneous memory system simulator on top of an open-source trace-driven multi-core cache simulator [33]. We chose this tool because it was easy to configure to meet our needs. We verified the claims of the simulator's accuracy and compared it to real hardware. We model a cache hierarchy with 32KB L1-I/D, 256KB L2, and a shared 8MB L3. The memory simulator models a heterogeneous single flat-addressable physical memory with different memory devices, their latency, energy and bandwidth parameters are listed in Table 3.

We assume a 4-core CPU with a 2GHz clock and a typical 3-level cache hierarchy. We set the hotness threshold to 32 accesses during an epoch (unless changed by the PRIORITY+ policy), which is set to 0.1s.

We assume that our memory is sufficient to contain the entire footprint of applications. Since the memory footprints of the benchmarks vary, specifically for multi-programmed benchmarks (see Table 2), we scale the total memory capacity to fit the needs of the benchmarks. We scale the memory sizes depending on their total memory footprints such that all benchmarks exert pressure on all memory devices (i.e, 3D-DRAM, DDR4 and PCM). For two-level memory systems we assume 1:4 ratio for the capacities of the faster and slower memories (3D-DRAM + DDR4 or 3D-DRAM + PCM), and for a three-level system we assume a 1:4:8 ratio for capacities of the memory devices (3D-DRAM + DDR + PCM).

Table 3: Memory performance and energy parameters. We use 3D-DRAM parameters with references to [29, 31, 7, 34, 35] , DDR4 parameters [13, 36] and PCM (NV_1x) parameters with reference to [22].

	Acc. latency	Acc. Energy	BW
3D-DRAM	40 ns	8.5 pj/bit	160 GB/s
DDR4	60 ns	35 pj/bit	25.6 GB/s
PCM Read (NV_1x)	1x DDR	1.2x DDR	1/2x DDR
PCM Write (NV_1x)	4x DDR	4x DDR	1/8x DDR

4.2.1 Memory performance and energy parameters

For our simulation purposes we used the memory parameters listed in Table 3, 4. We calculate the energy consumed using the statistics gathered from the simulation and the parameters listed in Table 3. We use PCM as a representative NVM device. We vary the access latencies to PCM relative to DDR4 latencies. In our NV_1x configuration, which is the optimal NVM configuration we assume, PCM read access latency is 1x and write latency is 4x than those of DDR4. We also assume that for the NV_1x configuration, PCM read bandwidth is two times lower than DDR4 read bandwidth and PCM write bandwidth is eight times lower than that of a DDR4 [22]. We also evaluated other configurations: NV_2x, NV_4x, NV_8x, where the PCM (read) latencies are 2, 4 and 8 times slower than that of a DDR4, the write latencies and bandwidth are adjusted proportionately. We take this approach partly because published literature differs in their assumptions regarding PCM latencies [4, 3, 22] and partly to represent other types of NVMs[2]. For the purpose of this paper, we linearly scale down PCM bandwidths when using higher latencies, but we also assume that writes will be buffered and do not cause delays in the critical path of execution, unless limited by the write bandwidth. We assume PCM uses write cancellation and write pausing techniques [24] such that the interference of write traffic with read traffic is minimized.

[2]The NV_1x vonfiguration may closely resemble 3D XPoint parameters.

Table 4: Page migration overheads

Per page cache flush time	4 μs
Whole cache flush time	550 μs
TLB shootdown time	4 μs
Memory transfer time per page	First block access latency + transfer time limited by the lower BW memory
Energy	We calculated per block access energy for each read and write

4.2.2 Estimating overheads

The time taken to flush a page from the cache hierarchy was estimated by measuring the total time it takes to flush a cache line from the cache hierarchy by issuing CLFLUSH x86 instruction to each of the 64 cache lines (for a 4KB page). The overhead for flushing the entire cache is estimated by executing WBINVD x86 instruction. Since the measured time includes only the time it takes to write back and invalidate the internal caches (approximately 250 μs), we add additional time needed for write-backs from LLC to main memory. In the worst-case the entire 8MB from LLC need to be written back to main memory. We estimated 300μs for writing back to DDR4, thus the total for entire cache flush is estimated at 550μs. If the write back from LLC is to some other memory device (3D or PCM), we use appropriate write-back overheads. Table 4 shows the overheads used for page migration in our experiments.

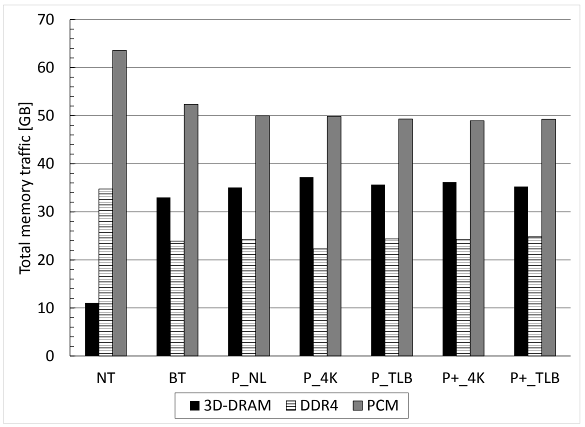

Figure 1: Total memory traffic in GB for a 3-level memory system composed of 3D-DRAM + DDR4 + PCM. The total traffic is averaged across the multi-programmed workloads for each of the policies. Notice that PRIORITY+ policy has 1% less traffic to PCM than PRIORITY.

5. EXPERIMENTS AND RESULTS

In this section, we analyze the performance and energy impact of the different page migration policies for different memory configurations. Below we summarize the policies and the labels we use for the policies in our figures. **NO_TRANSFER (NT).** We use this policy as the baseline

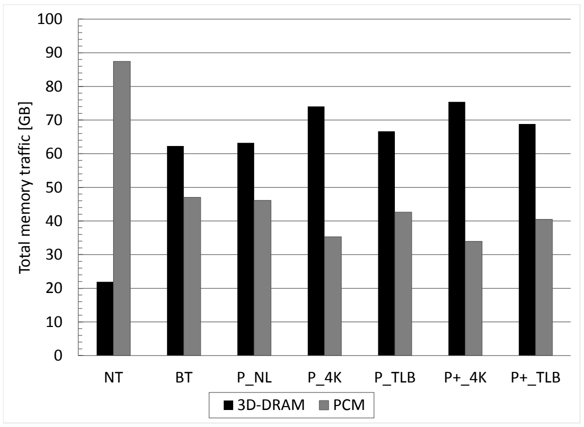

Figure 2: Total memory traffic in GB for a 3-level memory system composed of 3D-DRAM + PCM. The total traffic is averaged across the multi-programmed workloads for each of the policies.

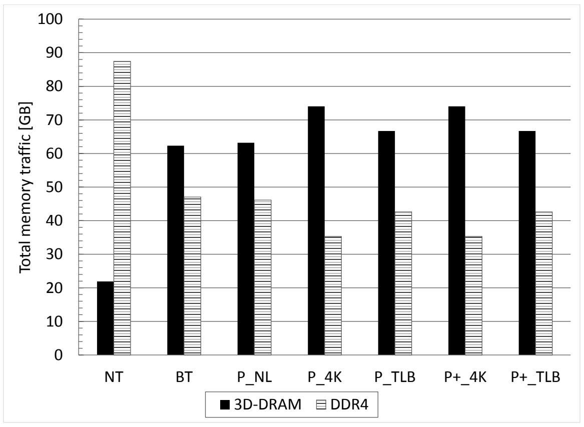

Figure 3: Total memory traffic in GB for a 3-level memory system composed of 3D-DRAM + DDR4. The total traffic is averaged across the multi-programmed workloads for each of the policies. Here PRIORITY and PRIORITY+ behave exactly the same because there is no threshold scaling for DDR4.

for comparisons. Pages are not migrated during program execution. **BASIC_TRANSFER (BT).** It is same as the Hot Page Policy. **PRIORITY_NL (P_NL).** This is the PRIORITY policy as described in Section 3 without any limits on the number of pages migrated. **PRIORITY_4K, TLB (P_4K, P_TLB).** Here we evaluate the PRIORITY policy with a limit on the number of pages migrated at each epoch. We evaluate other limits on the number of pages migrated per epoch, ranging between 4K-16K pages, as well as the case where we transfer only the hot pages which have valid TLB entries. We present only data for 4K and TLB limits since they page limit with 8K and 16K always behaved worse than 4K limit, so we omit 8K and 16K result. **PRIORITY+4K, TLB (P+_4K, P+_TLB).** We evaluate the PRIORITY+ policy as described in Section 3 with a 4K and TLB limit. Here the hotness threshold is varied based on the device where the page currently resides. **LOCKING LLC LINES.** We evaluate this option as an additional improvement to the placement policies outlined above as well as an alternative to page migration. We discuss the experiment results but omit figures due to limited space available.

We first discuss the improved memory accesses to 3D-DRAM and reduction in memory accesses to the slower memories (DDR4, PCM). We also show the number of pages transferred during a program execution. We present the speedups and energy savings as a consequence of improved accesses to 3D-DRAM and reduced number of pages transferred. We first show the results for the three level memory organization (3D-DRAM + DDR4 + PCM) for single and multi-programmed workloads. Second, we show results for two level memories (3D-DRAM + DDR4, 3D-DRAM + PCM) where we omit the figures for single-programmed workloads (but summarize results). In addition to using the NO_TRANSFER policy as a baseline, we compare the migration policies with systems that contain only 3D-DRAM and only DDR4. Finally, we show the results when slower non-volatile memories are used, which are 2, 4 or 8 times slower than DDR4 (i.e NV_2x, NV_4x and NV_8x configurations).

Improved memory accesses. The main goal of any page migration policy is to maximize the number of accesses to the fastest memory (3D-DRAM) and minimize the number of accesses to slower memories (DDR4, PCM). Figures 1 through 3 show the total traffic (in GB) to fast and slower memories for the different memory configurations. The traffic is averaged across all benchmarks for a given memory configuration. Note that our policies result in more accesses to the faster (3D-DRAM) memory and fewer accesses to the slower memory when compared to the BASIC_TRANSFER as well as NO_TRANSFER policy. Figure 1 shows the memory traffic for 3D+DDR+PCM configuration. We observe that our PRIORITY with 4K limit results in 3.3 times more accesses to the fast memory when compared to NO_TRANSFER and 12.8% more accesses when compared to BASIC_TRANSFER. Our PRIORITY+ with TLB limit policy reduces accesses to the slowest memory (PCM in this case) by 22.6% when compared to NO_TRANSFER and 6% when compared to BASIC_TRANSFER. Interestingly, although PRIORTY+ with TLB limit results in least amount of traffic to PCM, it does not maximize traffic to 3D-DRAM. This is due to the fact that PRIORITY with 4K limit migrates more pages to 3D-DRAM but since it is not latency aware it will not migrate the most useful PCM pages. Figure 2 shows memory traffic for two-level 3D+PCM configuration. PRIORTY+ with 4K limit behaves best in this case. We see 3.4 times more traffic to 3D-DRAM when compared to NO_TRANSFER and 21% more when compared to BASIC_TRANSFER. We also observe 62% less traffic to PCM when compared to NO_TRANSFER, and 28% less when compared to BASIC_TRANSFER. This is important because more access to 3D-DRAM leads to faster execution times. The PRIORITY+ appears better

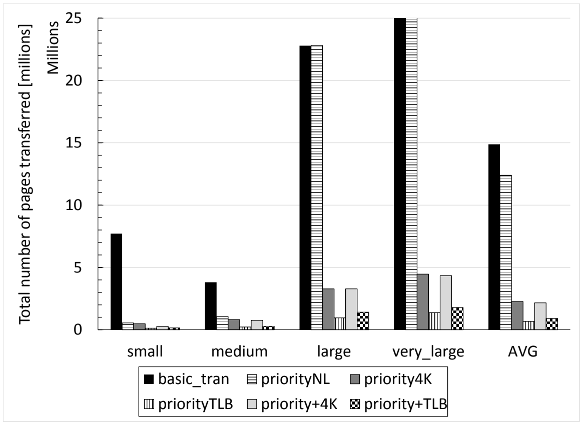

Figure 4: Total number of pages transferred in a 3-level memory system. The number of pages transferred goes down by an order of magnitude in most cases which translates to huge energy savings.

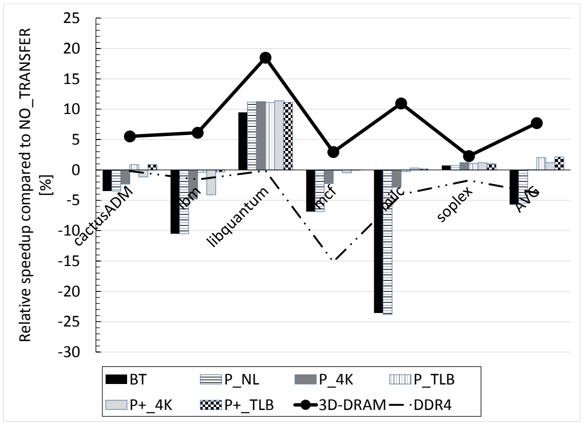

Figure 5: Speedup over NO_TRANSFER for single-programmed workloads in a 3D+DDR4+PCM system.

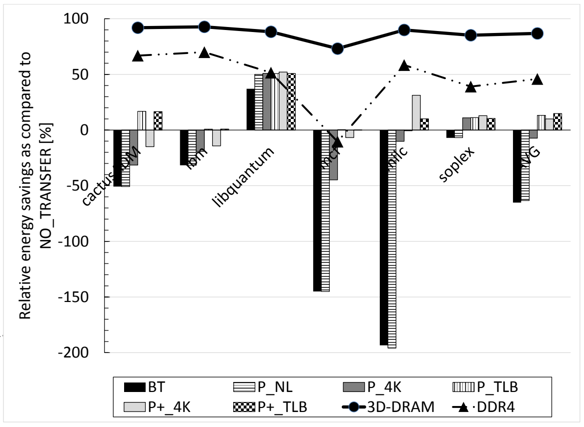

Figure 6: Energy savings over NO_TRANSFER for single-programmed workloads in a 3D+DDR4+PCM system.

than other policies because it is latency/bandwidth aware (in setting hotness threshold). Figure 3 shows the memory traffic for two-level 3D-DRAM+DDR4 configuration. We see 3.4 times more traffic to 3D-DRAM when compared to NO_TRANSFER 18.8% compared to BASIC_TRANSFER for our PRIORITY with 4K limit and PRIORITY+ with 4K limit (this is because the hotness threshold DDR4 is fixed at 32). Memory traffic to DDR4 is reduced 60% when compared to NO_TRANSFER and 25% compared to BASIC_TRANSFER

Number of pages transferred. Another key characteristic of a migration policy is the total number of pages migrated during a program execution. The fewer pages we transfer the higher the energy savings (and also overhead time savings). However, if we do not transfer as many useful pages as possible, we may lose some potential performance gains. Figure 4 shows the total number of pages transferred (for the whole program execution) for the benchmarks in our experiments.

It can be seen that our policies transfer an order of magnitude fewer (10 times fewer) pages when compared to the BASIC_TRANSFER policy, leading to reduced overheads. As we will see shortly, the increased traffic to faster memory (3D-DRAM) and the reduced number of pages migrated directly correspond to improved performance and large energy savings with our policies, when compared to the BASIC_TRANSFER policy [19].

5.1 Speedup and energy savings

3D-DRAM + DDR4 + PCM. Here we report results for a three level system that uses 3D-DRAM, DDR4 and PCM together as main memory. As stated previously, we vary the capacities of these devices based on applications' memory requirements and use a 1:4:8 ratio for the capacities of the three memory devices. Even though we ran all 12 benchmarks for single-programmed workloads, we show results for three latency bound (libquantum, milc, soplex) and three capacity bound (cactusADM, lbm, mcf) benchmarks in order to make the figures more legible. Figure 5 shows the average speedup as compared to NO_TRANSFER for the single programmed workloads. The capacity bounded benchmarks do not benefit from page migration. This is due to the fact that many pages after migration become cold We see minor improvement for PRIORITY with TLB and PRIORITY+ with TLB limit (up to 2%) over NO_TRANSFER. On the other hand, the latency bound benchmarks benefit more from the migration, specifically libquantum, where we observe 11% speedup over NO_MIGRATION policy and 2% speedup over BASIC_TRANSFER policy. The idealistic case, where sufficiently 3D-DRAM is the only memory in the system, is only up to 5% faster than the capacity bounded workloads (cactusADM, lbm, mcf) and up to 10% faster for latency bounded workloads (libquantum, lbm, mcf). This is because the single-programmed workloads do not create enough memory traffic to actually observe the benefits of having a 3D-DRAM only memory. It is impor-

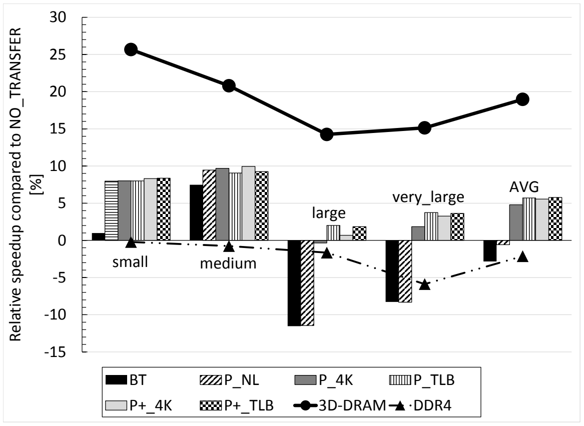

Figure 7: Speedup over NO_TRANSFER for multi-programmed workloads in a 3D+DDR4+PCM system.

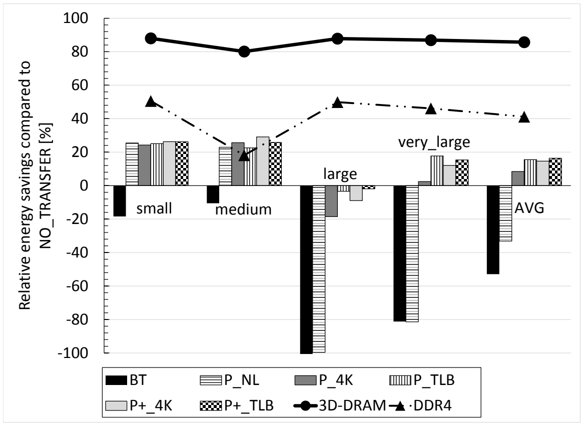

Figure 8: Energy savings over NO_TRANSFER for multi-programmed workloads in a 3D+DDR4+PCM system.

tant to notice that NO_TRANSFER always compares better than the system with sufficiently large DDR4 memory even with just small amounts of 3D-DRAM in the memory system. Therefore, it is important to compare the policies against the NO_TRANSFER policy and not just against the DDR4 only, because any system with small amounts of 3D-DRAM would outperform the system with DDR4 only.
Figure 6 shows the energy savings with our policies when compared to NO_TRANSFER. In every case, except for libquantum, BASIC_TRANSFER consumes more energy than the NO_TRANSFER policy. This is also true for PRIORITY policy with no limit on the number of pages migrated. However, with PRIORITY and PRIORITY plus, both with 4K limit on number of pages transferred, we can save up to 50% energy when compared to NO_TRANSFER (libquantum) and in some cases up to 230% when compared to BASIC_TRANSFER (milc). The idealistic 3D-DRAM only system consumes the least amount of energy due to low access energy per-bit. DDR4 consumes less energy than the heterogenous system because of the PCMs high write access energy (in our case 4x that of DDR4). This could be minimized with optimizations such as having DDR as a buffer for PCM writes and not as part of the addressable physical memory [13, 15].

Figure 7 shows the speedup over NO_TRANSFER policy for multi-programmed workloads. For the small and medium workloads, the benchmarks are mostly latency bound so we observe performance improvement over NO_TRANSFER policy. For the large and very large workloads, which are mostly capacity bound, we can see that the BASIC_TRANSFER policy behaves worse than NO_TRANSFER. Our PRIORITY and PRIORITY+ with page migration limits perform better than NO_TRANSFER. However, for the large workload NO_TRANSFER consumes the least amount of energy. The workloads come within 10%-20% of the performance exhibited by the 3D-DRAM only system. Figure 8 shows the energy savings over NO_TRANSFER policy. We can see that BASIC TRANSFER always consumes more energy than NO TRANSFER, while priority based policies with page migration limits show energy savings over both policies. The energy consumption is much higher for larger workloads because of the large number of pages being migrated during the lifetime of program execution. On average, NO_TRANSFER policy is 2.3% faster than the BASIC_TRANSFER policy and consumes 30% less energy. Our PRIORITY with TLB limit policy, on average shows 8% speedup compared the BASIC_TRANSFER and 5.7% compared to NO_TRANSFER, and consumes 67% less energy than BASIC_TRANSFER and 16% less energy than NO_TRANSFER.

3D-DRAM + PCM. Here we evaluate a two level system that uses 3D-DRAM and PCM. Figure 9 shows the relative speedup for the policies as compared to NO_TRANSFER. Figure 10 shows the energy savings. On average, the NO TRANSFER policy requires 2.3% more execution time than the BASIC TRANSFER policy, but it consumes 1.1% less energy. When compared to BASIC_TRANSFER, larger scale workloads with PRIORITY+ and 4KB limit exhibits signif-

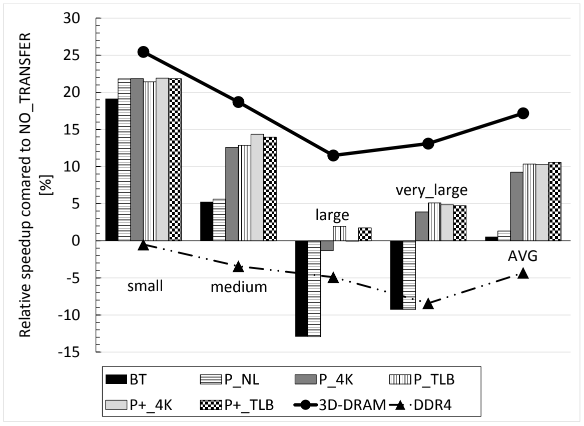

Figure 9: Speedup over NO_TRANSFER for multi-programmed workloads in a 3D+PCM system.

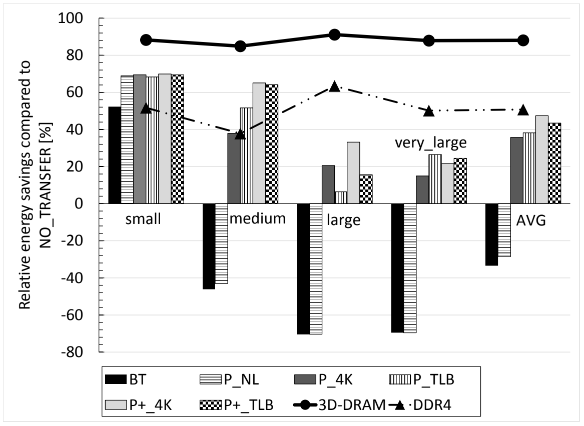

Figure 10: Energy savings over NO_TRANSFER for multi-programmed workloads in a 3D+PCM system.

icant performance improvements, up to 15% (large) and energy savings of up to 100% (large and very large). Our PRIORITY+ policy that limits the number pages transferred to 4K, on average, results in a performance gain of 9.3% compared to the BASIC_TRANSFER policy and also results in 56.9% energy savings.

3D-DRAM + DDR4. Figure 11 shows the relative speedup for the policies as compared to NO_TRANSFER. Figure 12 shows the energy savings. On average, the NO_TRANSFER policty requires 9% more execution time than the BASIC TRANSFER policy, but consumes 33% less energy. On average, our best policy that uses priority and limits the number of pages transferred to 4096 (that is, PRIORITY_4K), requires 2.7% less execution time and consumes 65% less energy when compared to the BASIC_TRANSFER policy. Compared to the NO_TRANSFER policy, on average, our PRIORITY_4K policy achieves 11.8% speedup and consumes 30% less energy.

For the two level sysem with 3D-DRAM plus DDR4, the BASIC_TRANSFER policy, on average, peforms better than the NO_TRANSFER policy (in terms of execution). However, in some cases the NO_TRANSFER policy performs better than the BASIC_TRANSFER policy by a small fraction. This is due to the fact that the BASIC_TRANSFER policy transfers a large number of pages and therefore incurs excessive overhead, outweighing the performance gains from transferring pages to faster memory (specifically true for large and very large workload). The NO_TRANSFER policy always consumes less energy than the BASIC _TRANSFER policy because there is no energy wasted for page migration. Actually this applies to any page migration policy that does not balance the number of pages transferred relative to the energy overheads versus improved performance due to transfers. The policies that limit the number of pages transferred at each epoch perform better than the NO_TRANSFER policy, because the overhead due to transfers is outweighed by improved performance.

User vs. Overhead time. For our optimal policies (PRIORITY and PRIORITY+) the total overhead time is rela-

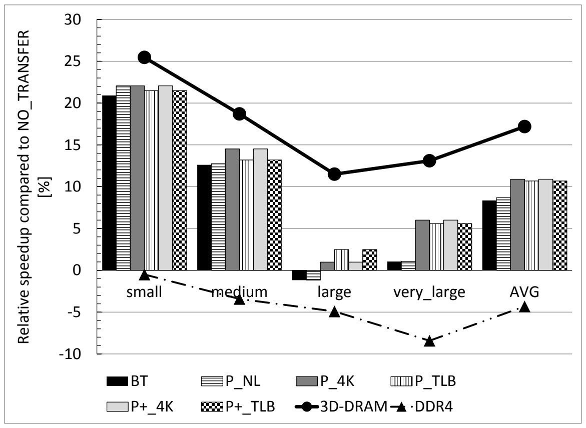

Figure 11: Speedup over NO_TRANSFER for multi-programmed workloads in a 3D+DDR4 system.15pt
15pt

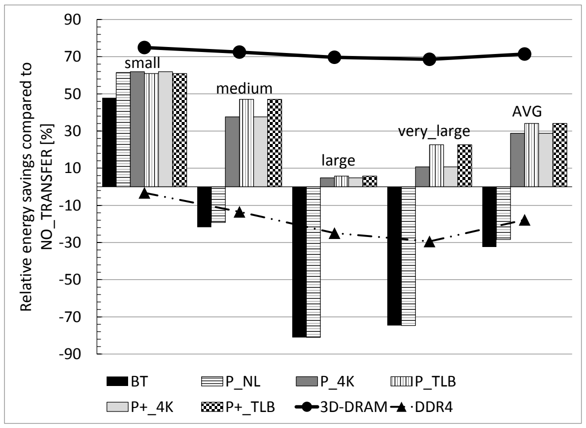

Figure 12: Energy savings over NO_TRANSFER for multi-programmed workloads in a 3D+DDR4 system.

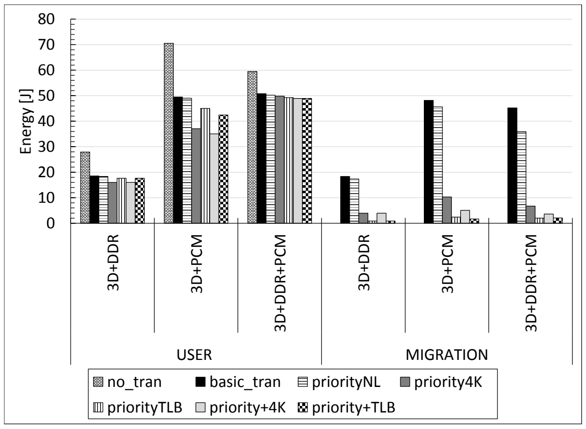

Figure 13: User vs. overhead energy averaged across all benchmarks for the 3 different memory systems

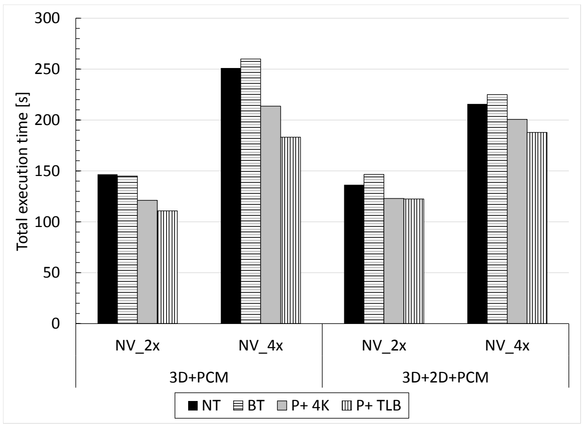

Figure 14: Sensitivity analysis for slower NVMs.

tively small compared to total execution time (0.5%-3% of total execution time). BASIC_TRANSFER policy incurs as much as 12% of the total execution times for overhead and thus our policies are significantly better than the BASIC_TRANSFER policy. The majority of the overhead is due to the page migration itself and only a small fraction of total time overhead is due to cache flushing on page migration (less than 10% of total overhead time). The minimal overhead time raises questions whether or not it is worth exploring possibilities of overlapping the page migration with regular program execution.

User vs. Overhead Energy. Figure 13. shows the user and overhead energy averaged across all benchmarks for the 3 different memory systems.The reduction in user energy is small, 2-3% on average and can be as high as 30% in some cases (for 3D+PCM, using PRIORITY_4K policy). The reduction in total overhead energy can be up to 95%. This especially true for systems with PCM as part of main memory because PCM consumes high write energy per-bit; since for the policies with a page migration limit we write less frequently to PCM during page migration.

Slower NVMs. Figure 14 shows heterogeneous systems with slower non-volatile memories that are two times (NV_2x) and four times (NV_4x) slower than DDR4. For NV_2x, we see no real advantage keeping DDR4 in the memory system since the non-volatile memories are only two times slower. However when using even slower NVMs (NV_4x), DDR4 can be beneficial in a heterogeneous systems. As the latencies of PCMs improve, we may rely on 3D-DRAM and PCMs only.

Locking LLC lines. Locking LLC lines did not show any performance or energy improvement for all of the policies. On average we get performance degradation of 2% and energy losses of 5% as compared to the cases without locking. Also, NO_TRANSFER with locking LLC lines did not show improvement which leads us to conclusion that it is not worth using this policy as an alternative to page migration.

6. CONCLUSION

The capacities of main memories needed by HPC and emerging applications are increasing exponentially. Such large capacities cannot be satisfied solely by DDR4 memories, while meeting latency and energy budgets. Thus there is an interest in heterogeneous memory systems that are built using multiple memory technologies with varying latencies, bandwidths and power requirements. The heterogeneous memory presents a single flat physical address (and not a hierarchical memory system). Yet it may be beneficial to migrate pages to faster memories in order to improve execution performance. However, the page migration incurs execution and energy overheads. In this paper we evaluated several different page migration policies that carefully trade-off migration overheads against performance gains. We evaluated our policies for two level systems with 3D-DRAM + DDR4 or 3D-DRAM + PCM as well as three level systems that use 3D-DRAM + DDR4 + PCM. We compared our policies with a previously published study that uses only page access counts, with fixed page hotness thresholds to decide which pages should be kept transferred to faster memories (i.e., BASIC_TRANSFER policy). We also compare our policies with heterogeneous memories that do not migrate pages.

One policy we explored tracks the usefulness of a page after it was migrated to faster memories in making decisions about future migrations of that page. We also explored the policies that limit the number of pages transferred at each epoch, and transferring only pages with valid entries in TLB. Our experiments show that these policies perform better than the BASIC_TRANSFER policy as they limit overheads and only migrate pages that are likely to be accessed after migration.

Although it is not clear which non-volatile memory technology is used in the recently announced 3D Xpoint memory, we feel our results are applicable to heterogeneous memories that include this new technology, since we varied the ratios of latencies of PCM relative to DDR4.

We also observe that knowledge about an application's page access behavior can be used to statically place pages and eliminate the need for page migration. In addition, we conclude that locking LLC lines is not worth using as an alternative to page migration.

7. REFERENCES

[1] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: a hybrid pram and dram main memory system," in *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pp. 664–669, IEEE, 2009.

[2] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[3] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 40, 2014.

[4] H. A. Khouzani, C. Yang, and J. Hu, "Improving performance and lifetime of dram-pcm hybrid main memory through a proactive page allocation strategy," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 508–513, IEEE, 2015.

[5] G. Loh, M. Schulte, M. Ignatowski, N. Jayasena, S. Gurumurthi, B. Beckmann, G. Rodgers, W. Brantley, I. Paul, and S. Reinhardt, "Achieving exascale capabilities through heterogeneous computing," 2015.

[6] S. Graham, "Hmc overview," *memcon Proceedings*, 2012.

[7] H. Consortium, "Hybrid Memory Cube Consortium." `http://www.hybridmemorycube.org/`, 2015. [Online; accessed September-09-2015].

[8] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.

[9] Intel, "Intel SSD DC P3700 Series Specifications." `https://www-ssl.intel.com/content/www/us/en/solid-state-drives/ssd-dc-p3700-spec.html`, 2015. [Online; accessed September-09-2015].

[10] D. Technologies, "MEMORY1." `http://www.diablo-technologies.com/memory1/`, 2015. [Online; accessed September-09-2015].

[11] Y. Park, S. K. Park, and K. H. Park, "Linux kernel support to exploit phase change memory," in *Linux Symposium*, vol. 2010, pp. 217–224, Citeseer, 2010.

[12] Y. Park, D.-J. Shin, S. K. Park, and K. H. Park, "Power-aware memory management for hybrid main memory," in *Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on*, pp. 82–85, IEEE, 2011.

[13] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[14] T. J. Ham, B. K. Chelepalli, N. Xue, and B. C. Lee, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 424–435, IEEE, 2013.

[15] H. G. Lee, S. Baek, C. Nicopoulos, and J. Kim, "An energy-and performance-aware dram cache architecture for hybrid dram/pcm main memory systems," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pp. 381–387, IEEE, 2011.

[16] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing pcm main memory lifetime," in *Proceedings of the conference on design, automation and test in Europe*, pp. 914–919, European Design and Automation Association, 2010.

[17] S. Lee, H. Bahn, and S. H. Noh, "Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures," *Computers, IEEE Transactions on*, vol. 63, no. 9, pp. 2187–2200, 2014.

[18] Intel, "Fun Facts: How Fast and Robust is 3D XPointŹ Technology?." `http://www.intel.com/newsroom/kits/nvm/3dxpoint/pdfs/NextGen_NVM_FunFacts.pdf`, 2015. [Online; accessed September-09-2015].

[19] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 126–136, IEEE, 2015.

[20] B. Black, "Die stacking is happening," in *46th IEEE/ACM International Symposium on Microarchitecture Keynote*, 2013.

[21] J. E. D. E. Council, "3D ICs." `http://www.jedec.org/category/technology-focus-area/3d-ics-0`, 2015. [Online; accessed September-09-2015].

[22] Numonix, "Phase Change Memory." `http://www.pdl.cmu.edu/SDI/2009/slides/Numonyx.pdf`, 2009. [Online; accessed September-09-2015].

[23] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "Preset: improving performance of phase change memories by exploiting asymmetry in write times," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 380–391, 2012.

[24] M. K. Qureshi, M. M. Franceschini, L. Lastras-Monta, *et al.*, "Improving read performance of phase change memories via write cancellation and write pausing," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–11, IEEE, 2010.

[25] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ACM SIGARCH computer architecture news*, vol. 37, pp. 14–23, ACM, 2009.

[26] Y. Liang and T. Mitra, "Instruction cache locking using temporal reuse profile," in *Proceedings of the 47th Design Automation Conference*, pp. 344–349, ACM, 2010.

[27] H. Ding, Y. Liang, and T. Mitra, "Integrated instruction cache analysis and locking in multitasking real-time systems," in *Proceedings of the 50th Annual Design Automation Conference*, p. 147, ACM, 2013.

[28] S. Phadke and S. Narayanasamy, "Mlp aware heterogeneous memory system," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pp. 1–6, IEEE, 2011.

[29] C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1–12, IEEE Computer Society, 2014.

[30] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "A case for specialized processors for scale-out workloads," *Micro, IEEE*, vol. 34, no. 3, pp. 31–42, 2014.

[31] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, M. Meswani, M. Nutter, and M. Ignatowski, "A new perspective on processing-in-memory architecture design," in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, p. 7, ACM, 2013.

[32] Intel, "Pin - A Dynamic Binary Instrumentation Tool." `https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool`, 2012. [Online; accessed September-09-2015].

[33] C. F. Shelor and K. M. Kavi, "Moola: Multicore cache simulator,"

[34] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: throughput-oriented programmable processing in memory," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pp. 85–98, ACM, 2014.

[35] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, F. Li, *et al.*, "Ndc: Analyzing the impact of 3d-stacked memory+ logic devices on mapreduce workloads," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pp. 190–200, IEEE, 2014.

[36] S. Borkar, "Exascale computing-a fact or affliction," *keynote presentation at IPDPS*, 2013.