Row Buffer Locality-Aware Data Placement in Hybrid Memories

HanBin Yoon
hanbinyoon@cmu.edu

Justin Meza
meza@cmu.edu

Rachata Ausavarungnirun
rausavar@andrew.cmu.edu

Rachael Harding
rharding@andrew.cmu.edu

Onur Mutlu
onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University

SAFARI Technical Report No. 2011-005

September 5, 2011

Abstract

Phase change memory (PCM) is a promising alternative to DRAM, though its high latency and energy costs prohibit its adoption as a drop-in DRAM replacement. Hybrid memory systems comprising DRAM and PCM attempt to achieve the low access latencies of DRAM at the large capacities of PCM. However, known solutions neglect to assess the utility of data placed in DRAM, and hence fail to achieve high performance and energy efficiency.

We propose a new DRAM-PCM hybrid memory system that exploits row buffer locality. The main idea is to place data that cause frequent row buffer miss accesses in DRAM, and data that do not in PCM. The key insight behind this approach is that data which generally hit in the row buffer can take advantage of the large memory capacity that PCM has to offer, and still be accessed as quickly as if the data were placed in DRAM. We observe our mechanism (1) effectively mitigates the high access latencies and energy costs of PCM, (2) reduces memory channel bandwidth consumption due to the migration of data between DRAM and PCM, and (3) prevents data that exhibit low reuse from polluting DRAM.

We evaluate our row buffer locality-aware scheme and show that it outperforms previously proposed hybrid memory systems over a wide range of multiprogrammed workloads. Across 500 workloads on a 16-core system with 256 MB of DRAM, we find that our scheme improves system performance by 41% over using DRAM as a conventional cache to PCM, while reducing maximum slowdown by 32%. Furthermore, our scheme shows 17% performance gain over a competitive all-PCM memory system, and comes to within 21% of the performance of an unlimited-size all-DRAM memory system.

1 Introduction

The main memory in modern computers relies predominantly on *Dynamic Random Access Memory (DRAM)*. Though strides in DRAM process technology have enabled DRAM to continue to scale to smaller feature sizes and thus higher densities, DRAM scalability has been predicted to be approaching its limit [29, 26]. However, today's multiprogrammed workloads on chip multiprocessors (CMP) require larger amounts of main memory to support the working sets of many concurrently executing threads. As the number of cores on a chip continues to increase, supplying such memory capacity exclusively with DRAM will quickly become expensive in terms of both cost and energy.

Phase Change Memory (PCM) offers a competitive alternative to DRAM. PCM is an emerging byte-addressable random-access memory technology that is capable of scaling beyond the smallest feature sizes manufacturable for DRAM [26, 14, 13]. This is due to the fact that PCM stores information as varying electrical resistance, which is more amenable to extreme scaling than DRAM that stores information as a small amount of electrical charge. Furthermore, the non-volatility presented by PCM not only eliminates the need for periodic refresh of the memory cells, but also opens the door to system-level opportunities that may exploit this characteristic (such as lowering OS overheads and increasing I/O performance through the use of persistent memory [4, 5, 1]).

However, PCM has a number of shortcomings which prohibit its adoption as a drop-in DRAM replacement. First, PCM exhibits higher read and write latencies compared to DRAM. Second, the energy consumed in reading from or writing to an individual memory cell is higher for PCM than it is for DRAM. Finally, PCM cells wear out as they undergo write operations. Clearly, these disadvantages must be tackled in order for PCM to become a viable alternative for main memory.

Previous research [23, 6, 33] has suggested heterogeneous main memory systems comprising both DRAM and PCM to tackle these problems. These hybrid memory systems commonly employ a small amount of DRAM and a large amount of PCM, where blocks of data can be migrated to be placed in DRAM (*"promotion"*) and/or in PCM (*"demotion"*). The DRAM is used as a cache to PCM [23], or used to store data that is frequently written to [6, 33]. These mechanisms aim to counter high PCM latencies and energy, and to reduce and even out PCM wear. However, their data placement schemes are oblivious to the varying degree of performance benefits gained by promoting different blocks of data—i.e., it would be more beneficial to promote data that is read frequently, than to promote data that is not. Consequently, these solutions yield sub-optimal performance.

Our new hybrid memory system exploits this difference in the benefits gained from promoting different blocks of data, to achieve the goal of higher performance and lower energy consumption. Our scheme is based on several key observations.

First, previous studies [31, 28, 15, 16, 13] have shown that memory access latencies and energy consumption can be effectively reduced by exploiting data access locality at the internal data buffer of the memory device (known as the *row buffer*). When accessing data that has already been read into the row buffer, memory requests are serviced promptly by using only this peripheral circuitry, without accessing the memory cell array. Such memory accesses are called *row buffer hit* accesses, whereas for *row buffer miss* accesses data in the memory cell array must be read into the row buffer. Since DRAM and PCM have similar memory array organizations and buffer circuity are independent of memory technology, row buffer hit accesses in DRAM and PCM incur equal latency and energy costs [13]. However, row buffer miss accesses are much slower and consume higher energy to service in PCM than in DRAM. Hence, it is beneficial to place data that causes frequent row buffer misses in DRAM, and data that causes few row buffer misses in PCM. This way, the bulk of row buffer miss accesses that would otherwise suffer expensive PCM array access will be converted into cheaper DRAM array accesses. We observe that such a data placement scheme is effective in mitigating high PCM latencies and energy costs.

Second, recent research [11] has demonstrated that when caching large blocks of data (on the order of kilobytes), selectively promoting only a subset of the total referenced blocks prevents the memory channel bandwidth from becoming saturated with block migration operations. In a hybrid memory organization, frequently migrating blocks of data between DRAM and PCM incurs significant channel bandwidth and memory access costs, which delay servicing other memory requests. Therefore, it is prudent to promote only the data that yield latency and energy benefits that outweigh the promotion cost. We observe that this filters out data cachings that do more harm than good for system performance.

Finally, we find that different blocks of data exhibit varying degrees of reuse. In other words, certain blocks of data are frequently reused after being promoted, whereas other blocks are hardly, if ever, reused. Such blocks with low reuse pollute the limited DRAM capacity and cause thrashing to occur. Thus, our mechanism aims to promote blocks of data that will be frequently reused in DRAM. Under our scheme, we observe that the DRAM capacity is effectively utilized, and we compare our technique to previous studies [24, 7] that account for the degree of cache block reuse at the last level cache.

Overview of mechanism. We propose a new hybrid memory scheme that leverages the observations mentioned above. Our system identifies blocks of data that yield high benefits on latency and energy by being promoted. Ideal candidate blocks are those that are frequently accessed in a row buffer missing manner. These blocks are identified by tracking their access and row buffer miss counts on a quantum basis. If the access and row buffer miss counts exceed certain thresholds, the block is deemed to have sufficient reuse and row buffer miss properties, and is thus promoted. By selectively promoting only data that yield high latency and energy benefits, our scheme outperforms previous hybrid memory techniques.

This study does not directly address the problem of PCM wear-leveling. However, it is possible to use our proposed mechanism in conjunction with known wear-leveling methods such as Start-Gap [22].

Contributions. In this work, we make the following contributions:

- We introduce the notion of mitigating high PCM latency and energy costs by employing a data placement scheme that exploits row buffer locality in a hybrid memory system. We observe that data which seldom cause row buffer miss accesses may be placed in PCM, and still be accessed at the same cost as DRAM for row buffer hit accesses.

- We propose the selective promotion of data blocks that are frequently accessed in a row buffer missing manner, to a faster memory medium in a hybrid memory organization. We find that this scheme (1) mitigates the high latency and energy costs of the slower memory medium, (2) alleviates the pressure on memory channel bandwidth, and (3) avoids polluting the faster, smaller memory medium with low-reuse data. Though our study evaluates the particular memory combination of DRAM and PCM, our scheme is applicable to heterogeneous memory organizations in general, provided that row buffers (or equivalent peripheral circuitry) are employed in the memory devices. We show how varying the slower memory medium's access latency affects the our mechanism's performance in Section 6.2.
- We evaluate our proposed mechanism against known data placement schemes/caching policies [24, 7] across a wide variety of multiprogrammed workloads on a 16-core system. We show that our scheme outperforms existing techniques, and that it improves system performance (weighted speedup) by 41% while reducing maximum slowdown by 32% over using DRAM as a conventional cache to PCM.

2 Background and motivation

2.1 Phase change memory

PCM is a non-volatile memory technology that stores information by varying the resistance of a material known as chalcogenide [32, 26]. To perform a write to a PCM cell, the chalcogenide is first heated up to its melting point. Depending on whether a binary value of '0' or '1' is to be written to the cell, the molten chalcogenide is either cooled down quickly or slowly, respectively. It is this difference in cooling rates that determines whether the chalcogenide solidifies into an amorphous high resistance state or a crystalline low resistance state, which are used to represent binary values of '0' and '1' respectively. Multi Level Cell (MLC) PCM controls this cooling rate to achieve more than two resistance levels. This allows the representation of more than a single bit per cell, increasing PCM's density opportunities in a way which known DRAM technologies cannot offer.

Though PCM offers scalability and non-volatility benefits over DRAM, it also has its drawbacks. The melting and cooling of chalcogenide cause PCM to incur higher access latency and energy consumption than DRAM, and furthermore lead to cell wear-out after many writes. A technology survey on nine recent PCM devices and prototypes [13] reported that reading a PCM cell requires approximately 4–6$\times$ the read latency of a DRAM cell, while writing incurs approximately 6–32$\times$ the write latency of a DRAM cell. In addition, reading a PCM cell requires approximately 2$\times$ the read energy of a DRAM cell, while writes incur approximately 10–140$\times$ the write energy of a DRAM cell. Furthermore, whereas DRAM cells do not experience wear-out, the lifetime of a future PCM cell is estimated at 10^8 writes.

2.2 Row buffer locality

Memory devices typically have a cell array that is organized into rows and columns, as illustrated in Figure 1. A memory request specifies the address of its data by indicating the row index and the offset of the column within the row from which to start accessing data. Upon a read or write request, the data content of the specified row is read by sense amplifiers and latched, in peripheral circuitry known as the *row buffer*. A read request is then serviced by transmitting the data from the row buffer to an output port. In the case of a write request, the incoming data from the input port is written to the row buffer.

Once the data content of a particular row is latched in the row buffer, subsequent memory requests to different columns in the same row can be serviced quickly from the row buffer. Such a memory access is termed a *row buffer hit*, whereas if a different row is requested and the content of the row buffer needs to be changed, the memory access is termed a *row buffer miss*. For row buffer hits, only the peripheral circuitry is utilized, and no access is necessary to the underlying memory cell array, whether it be DRAM or PCM. *Row Buffer Locality (RBL)* refers to the reuse of the same row while it is buffered, thus being able to service memory requests efficiently without having to replace the content of the row buffer.

Our hybrid memory system exploits row buffer locality by placing data that are likely of causing fewer row buffer miss accesses in PCM. This way, the large PCM capacity can be accessed at the same cost as accessing DRAM, for row buffer hit accesses. Data that generate frequent row buffer miss accesses are placed in DRAM, to avoid compounding high PCM latency and energy costs. Figure 2 illustrates this point by showing how data in DRAM and PCM result in

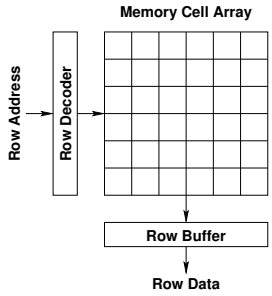


Figure 1: Memory cell array.

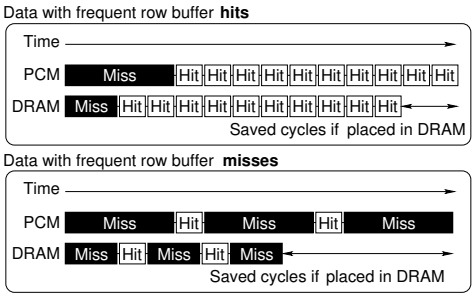


Figure 2: Data placement affects service time.

different access latencies for varying row buffer locality. It can be seen that the amount of cycles saved by placing the same data in DRAM rather than PCM is greater when the accesses to the data are frequently missing in the row buffer.

2.3 Data placement in hybrid memory

We motivate our new hybrid memory scheme by comparing the performance of a system with all-PCM main memory (no DRAM) to another system that employs a small DRAM cache to the PCM memory (our experimental setup and benchmark selection is discussed in Section 3). Figure 3 shows the performances of these two memory configurations in terms of instructions per cycle (IPC), for a single-core system. While the DRAM cache improves the performance of some applications, for certain benchmarks such as libquantum, omnetpp, and xalancbmk, the system augmented with DRAM exhibits lower performance. This phenomenon occurs due to the high cost of migrating large blocks (discussed in Section 3.1) of data between DRAM and PCM. In order to migrate a block of data from one memory medium to another, the block must be read from one medium, transmitted over the memory channel, and then written into the other medium (this last write portion may be off the critical path). The memory latencies involved in such a task are not trivial, and the memory channel bandwidth consumed is high.

This simple case study illustrates that we cannot rely on conventional caching policies alone to implement an effective DRAM cache to PCM. Furthermore, we find that it is not in the best interest of system throughput to indiscriminately promote every block of data that is referenced. For example, a block may not be frequently reused after promotion. Additionally, if a certain block experiences only row buffer hit accesses after being promoted, it could have been just as efficiently accessed from PCM, without incurring migration costs. All of the described blocks do not return sufficient benefits over the migration costs invested in them, and only serve to pollute the valuable DRAM cache space.

Our goal is to develop a high performance, low power DRAM-PCM hybrid memory system. To this end, we propose a scheme that selectively promotes only those blocks of data that are predicted to yield high benefits in terms of latency and energy. Such blocks are those that generate frequent row buffer miss accesses and are reused many times after promotion.

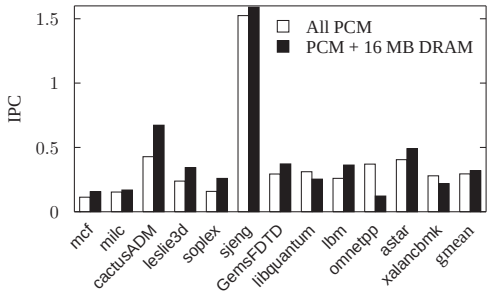


Figure 3: Effects of a simple caching scheme.

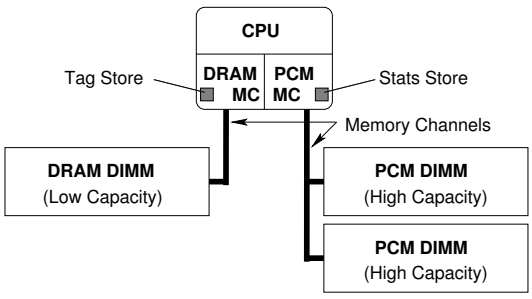


Figure 4: Hybrid main memory organization.

3 System organization

3.1 Hybrid memory organization

Figure 4 shows the organization of our hybrid memory system[1]. DRAM is used as a bypassable 16-way set-associative cache to PCM, and we propose row buffer locality-aware data promotion schemes for its effective utilization in Section 4. We use a large DRAM cache block size equal to the DRAM and PCM row size (2 KB) for two reasons: (1) to allow for a reasonably-sized DRAM cache tag store (as in [11]), and (2) to assign each block its own row for the purpose of maintaining the block's RBL characteristic throughout promotion and demotion (we also adopt a row-interleaving data layout for this reason[2]). A hardware structure in the memory controller (shaded gray in Figure 4) tracks access and row buffer miss counts for recently accessed rows in PCM. This information is utilized by our data placement schemes to predict the benefits of promoting different blocks (rows) from PCM. The write data buffer in the memory controller is used as temporary storage for the migration of blocks (rows) during promotion and demotion (Section 5 discusses implementation). We refer to DRAM cache blocks as "rows" hereafter, to differentiate them from L1/L2 cache blocks.

We implement DRAM as an inclusive cache to PCM. In other words, all data promoted to DRAM has its original copy located in PCM. This design choice was made in order to adopt *Line Level WriteBack* [23] (known as *Partial Writes* in [13]), which we implement by having the memory controller track the dirtiness of promoted rows at a sub-row granularity (128 B in this work, equal to the size of L1/L2 cache blocks), and write back only the dirty sub-rows on demotion to PCM. We observe that this approach greatly reduces the PCM write energy consumed on the demotion of rows. This is because many sub-rows do not experience writes while promoted, hence the amount of data that is written back to PCM on the demotion of rows is reduced.

3.2 Experimental methodology

We use an in-house CMP simulator to evaluate our proposed mechanisms. Table 1 describes the details of the processor and memory models, and lists key system parameters of our simulations. We do not model a disk, as it is not our goal to measure the benefits of reduced page faults owing to a larger main memory capacity provided by PCM (this effect was the focus of the study in [23]). Rather, our focus is on evaluating different data promotion schemes for the DRAM cache to PCM. Hence, the PCM capacity is set to be large enough to contain the working sets of the workloads in the simulations.

Our workload selection is from the SPEC CPU2006 benchmark suite. We use 200 million instruction traces of the benchmarks that are representative of the major phase of the program, generated by PinPoints [20]. We categorize benchmarks into two groups depending on the size of their working sets, which we regard as the aggregate size of unique rows an application accesses throughout its trace. The 14 benchmarks in the SMALL working set category have working sets small enough to fully fit into a 16 MB DRAM cache (if the DRAM cache were fully-associative). The 12 benchmarks in the LARGE working set category have working set sizes greater than 16 MB. Table 2 lists the benchmarks along with their key application characteristics, according to the working set size category they fall into.

In the following section (Section 4), we show illustrative results for single-core executions of the individual benchmarks under various data promotion schemes. This is to observe the isolated behavior of different data promotion policies in the absence of interfering memory requests issued by other processing cores. Our main evaluation is presented in Section 6, where we show results for a 16-core system. In addition, we examine the effects of varying the DRAM size, PCM access latencies, and the number of processing cores in the system. Furthermore, we compare the performance of our mechanism to known caching schemes such as the *Dynamic Insertion Policy (DIP)* [24] and probabilistic caching [7].

[1] We situate DRAM and PCM on separate memory channels, as building a memory controller to implement both DRAM and PCM timing characteristics could lead to increased complexity. If DRAM and PCM are located on the same channel, the problem of memory channel contention due to migrations is exacerbated, which our proposed techniques effectively mitigate.

[2] With block-interleaving, our technique can be combined with others such as Micro-Pages [31] to co-locate in the same row the cache lines that are accessed in a frequently row buffer missing manner.

System organization	1–16 cores; 2 on-chip memory controllers (1 for DRAM, 1 for PCM).
Processor pipeline	Out-of-order issue; 128-entry instruction window; fetch/execute/commit 3 instructions per cycle (max. 1 memory op per cycle).
L1(L2) cache	Private(Shared); 32(512) KB per core; 4(8)-way set-associative; 128 B blocks.
Memory controller	DDR3 1066 MHz; 128-entry request buffer; 128-entry write data buffer; FR-FCFS [28] scheduling; open row policy; 512 cycles to migrate a 2 KB row.
DIMMs	**DRAM.** 1 rank; 8 DRAM chips on a DIMM. **PCM.** 2 ranks (for 16 or 8 cores in system) or 1 rank (for 4, 2, or 1 cores in system); 8 PCM chips on a DIMM.
DRAM	256, 128, 64, or 32 MB for 16, 8, 4, or 2 cores, respectively; 8 banks per chip; 2 KB row buffer per bank; 0.0016 pJ/bit/cycle/bank static energy. **Latency.** Row buffer hit 40 ns (200 cycles); row buffer miss 80 ns (400 cycles). **Energy.** Row buffer read 0.93 pJ/bit; row buffer write 1.02 pJ/bit; cell read 1.17 pJ/bit; cell write 0.39 pJ/bit.
PCM	8, 4, 2, 1 GB, or 512 MB for 16, 8, 4, 2, or 1 core(s), respectively; 8 banks per chip; 2 KB row buffer per bank; 0.0016 pJ/bit/cycle/bank static energy. **Latency.** Row buffer hit 40 ns (200 cycles); clean row buffer miss 128 ns (640 cycles); dirty row buffer miss 368 ns (1840 cycles). **Energy.** Row buffer read 0.93 pJ/bit; row buffer write 1.02 pJ/bit; cell read 2.47 pJ/bit; cell write 16.82 pJ/bit.

Table 1: Simulator model and parameters. Memory latencies and energies are based on [13]. The latencies conservatively include overheads for interconnect traversal from the L2 cache to the memory controller, synchronization and clocking overhead at the DRAM interface, and ECC overhead in DRAM.

4 Hybrid memory promotion policies

Overview. Recall from Section 2 that the goal of our mechanism is to promote data which have little row buffer locality and exhibit high reuse. To identify such data, the stats store counts the number of total memory accesses and the number of row buffer misses to each row in PCM. Depending on the promotion policy in use (we discuss several in this section), if either or both of these counts for a row exceed particular thresholds, the row is promoted. We observe that (1) promoting data which are the source of frequent row buffer misses reduces the latency of subsequent accesses which also miss in the row buffer, and, (2) promoting only rows with high reuse reduces the number of migrations performed, reducing the amount of memory channel contention and memory access load.

Though the purpose of the stats store is to identify rows in PCM that have high reuse and frequent row buffer misses, we observe that rows that are not so can slowly accumulate large enough access and row buffer miss counts over a long time to qualify for promotion. We therefore zero the counts in the stats store every 10 million cycles to only consider recent accesses.

4.1 A row reuse-aware promotion policy

We first consider a promotion policy which only uses row reuse information. Similar to prior works [12, 11, 25], the A-COUNT ("access count") policy tracks the number of accesses to each row in PCM, and promotes rows if the number of accesses they receive exceeds a static threshold A. We predict that a row that has been accessed in the recent past will be accessed again in the near future. Instead of promoting rows on their first access (as in conventional caching), A-COUNT assures that only the rows with high reuse are promoted by delaying the promotion of the row until A accesses are observed.

Figure 5 compares the performance of several promotion policies for the LARGE benchmarks on a single core system. We compare how the policies perform by examining the number of cycles the DRAM and PCM memory channels spend in each of four states: (1) idle, (2) receiving data over the channel that has been read from memory, (3) transmitting data over the channel to be written to memory, and (4) migrating data from memory due to a promotion or demotion. For policies with static thresholds such as A-COUNT, we profile the benchmarks and show results for using the optimized thresholds for each benchmark. We do this for analysis and later develop a mechanism that dynamically determines the promotion threshold. The height of the bars in Figure 5 is representative of the execution time of an

Benchmark	RBHR	MPKI	WS (MB)	L/S	Benchmark	RBHR	MPKI	WS (MB)	L/S
milc	0.56	13.0	359.6	L	gobmk	0.48	0.60	8.0	S
astar	0.52	4.3	268.7	L	gromacs	0.61	0.65	6.3	S
GemsFDTD	0.41	13.1	255.3	L	gcc	0.46	0.16	4.9	S
lbm	0.86	25.0	180.4	L	bzip2	0.69	3.50	3.8	S
leslie3d	0.55	11.0	73.9	L	perlbench	0.59	0.05	3.0	S
sjeng	0.21	0.4	70.3	L	h264ref	0.79	0.99	2.9	S
omnetpp	0.10	18.1	54.7	L	hmmer	0.48	2.79	2.1	S
cactusADM	0.14	3.1	32.7	L	dealII	0.75	0.07	1.8	S
libquantum	0.94	13.2	32.0	L	namd	0.78	0.07	1.7	S
xalancbmk	0.44	15.1	29.0	L	wrf	0.80	0.14	1.4	S
soplex	0.73	22.6	22.3	L	calculix	0.67	0.03	1.0	S
mcf	0.13	57.0	22.1	L	povray	0.72	0.01	0.5	S
sphinx3	0.53	7.43	13.6	S	tonto	0.78	0.01	0.4	S

Table 2: Benchmark characteristics (RBHR = Row Buffer Hit Rate; MPKI = last-level cache Misses Per Kilo Instructions; WS = Working Set size; L/S = LARGE/SMALL WS; sorted in descending order of working set size).

application under a particular promotion policy, because we execute each of the benchmarks for the same number of instructions regardless of the promotion policy in use. Shorter bars correspond to higher performance. Table 3 shows the number of data migrations performed by the different promotion policies, normalized to the amount performed by conventional caching. Memory access type distributions (whether to DRAM or PCM; row buffer hit or miss) are tabulated in Table 4, and Table 5 shows the average CPU stall cycles due to off-chip memory access.

The A-COUNT policy promotes only rows with high amounts of reuse, reducing the amount of contention at the memory channels. This benefits applications such as omnetpp and xalancbmk which, under conventional caching, experience large amounts of bus contention (as seen by the large fraction of cycles spent on "migrating" in Figure 5). Other applications such as milc, leslie3d, and lbm take longer to execute under the A-COUNT policy because too few data are promoted, requiring PCM to service a large number of these requests (hence the larger CPU stall times in Table 5). libquantum has minimal performance change because it streams through its data, accessing rows enough times to be promoted under the A-COUNT policy, but not accessing them in a row buffer missing manner while they reside in the DRAM cache, resulting in the same useless migrations as conventional caching.

The A-COUNT policy, though able to improve performance in some benchmarks by reducing the migration of data that has low reuse, ignores the difference in row buffer miss latencies between DRAM and PCM. Most of the working set of libquantum, for example, could have been serviced just as quickly from PCM without incurring expensive migration costs, due to its high row buffer hit rate. Imposing a static access count threshold for promotion causes unnecessary migrations of data that frequently hit in the row buffer, whose memory accesses would have been serviced

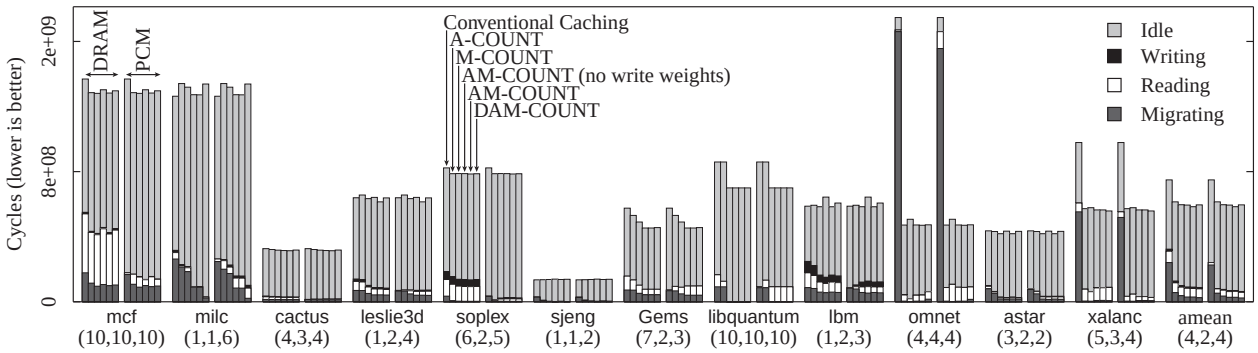


Figure 5: Number of cycles during which the DRAM channel (left six bars) and PCM channel (right six bars) are idle, receiving data being read from memory, transmitting data being written to memory, and migrating data, under different promotion policies. Numbers below are the best static thresholds for the A-, M-, and AM-COUNT policies.

Policy	mcf	milc	cactus	leslie3d	soplex	sjeng	Gems	libquantum	lbm	omnetpp	astar	xalanc	amean
A-cnt	0.64	0.83	0.97	1.00	0.15	0.23	0.99	1.00	1.00	0.00	0.65	0.00	0.62
M-cnt	0.54	0.72	0.93	0.76	0.08	0.04	0.71	0.00	0.70	0.00	0.19	0.00	0.39
AM-cnt	0.56	0.35	0.91	0.63	0.06	0.04	0.58	0.00	0.70	0.00	0.19	0.01	0.34

Table 3: Amount of data migrations performed, normalized to the amount performed by conventional caching (ConvCh = Conventional Caching; A-cnt = A-COUNT; M-cnt = M-COUNT; AM-cnt = AM-COUNT).

Policy	Type	mcf	milc	cactus	leslie3d	soplex	sjeng	Gems	libquantum	lbm	omnet	astar	xalanc	amean
ConvCh	DH	0.11	0.42	0.12	0.49	0.71	0.05	0.41	0.86	0.81	0.02	0.30	0.28	0.38
	DM	0.86	0.36	0.83	0.44	0.27	0.54	0.54	0.07	0.16	0.51	0.49	0.43	0.46
	PH	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	PM	0.03	0.23	0.05	0.08	0.02	0.42	0.05	0.07	0.03	0.48	0.21	0.28	0.16
A-cnt	DH	0.09	0.15	0.13	0.47	0.72	0.05	0.39	0.39	0.70	0.13	0.15	0.41	0.31
	DM	0.75	0.17	0.74	0.42	0.25	0.29	0.32	0.07	0.14	0.19	0.32	0.31	0.33
	PH	0.02	0.24	0.03	0.02	0.02	0.12	0.20	0.47	0.13	0.00	0.30	0.07	0.14
	PM	0.14	0.44	0.11	0.09	0.02	0.54	0.09	0.07	0.03	0.68	0.23	0.22	0.22
M-cnt	DH	0.10	0.21	0.12	0.31	0.63	0.03	0.29	0.00	0.52	0.08	0.15	0.27	0.23
	DM	0.73	0.24	0.73	0.36	0.24	0.13	0.28	0.00	0.10	0.06	0.27	0.30	0.29
	PH	0.03	0.24	0.07	0.23	0.11	0.16	0.35	0.94	0.33	0.07	0.37	0.20	0.26
	PM	0.14	0.31	0.08	0.10	0.01	0.68	0.08	0.06	0.04	0.79	0.20	0.22	0.23
AM-cnt	DH	0.10	0.03	0.12	0.30	0.62	0.03	0.25	0.00	0.52	0.13	0.15	0.40	0.22
	DM	0.74	0.08	0.68	0.29	0.23	0.13	0.14	0.00	0.10	0.19	0.27	0.37	0.27
	PH	0.03	0.44	0.09	0.30	0.12	0.16	0.50	0.94	0.33	0.01	0.37	0.07	0.28
	PM	0.13	0.45	0.11	0.11	0.02	0.68	0.11	0.06	0.04	0.67	0.20	0.16	0.23

Table 4: Memory access type distributions (DH = DRAM row Hit access; DM = DRAM row Miss access; PH = PCM row Hit access; PM = PCM row Miss access; Refer to Table 3 for full policy names).

efficiently in PCM (cf. Figure 2).

4.2 A row buffer locality-aware promotion policy

We next examine a novel promotion policy which considers row buffer locality information. The M-COUNT ("miss count") policy tracks the number of accesses to each row in PCM that miss in the row buffer when accessing data, and promotes rows if they experience a static threshold of M row buffer miss accesses. We predict that if a row has observed many row buffer misses in the recent past, it will continue to see more in the future. The M-COUNT policy assures that only rows that incur frequent row buffer misses are promoted by delaying the promotion of rows until M row buffer misses are observed.

By promoting rows which cause frequent row buffer misses, the M-COUNT policy promotes rows which benefit from the lower row buffer miss latencies of DRAM. In Figure 5, we see that the M-COUNT policy reduces the number of cycles spent migrating data, while still performing many beneficial (row buffer missing) accesses to DRAM (cf. Table 4). This is achieved by identifying and denying promotion of data which are mainly serviced from the row buffer in PCM (at the same latency as in DRAM). This improves the performance of most applications (CPU stall times are also reduced), including libquantum (which has a very high row buffer hit rate), milc, leslie3d, and lbm, all where A-COUNT is ineffective. Other applications, such as omnetpp, xalancbmk, and sjeng, perform more poorly compared to the A-COUNT policy because they contain a large number of rows which are promoted to DRAM under the M-COUNT policy but receive little reuse once there. We find optimized M thresholds to be typically smaller than A thresholds, thus accounting less for data reuse. Our findings suggest that an approach which takes into account *both* row buffer locality and row reuse can achieve the best of both worlds.

4.3 A combined row buffer locality/reuse promotion policy

We next present a promotion policy which considers both row buffer locality and row reuse for its promotion decisions. For this policy, we track the number of accesses to a row (as in A-COUNT) as an indicator of row reuse, and the number of times data miss in the row buffer when accessing a row (as in M-COUNT) as an indicator of frequent row buffer misses. We call this the AM-COUNT policy, and it promotes rows to DRAM if they have been accessed A times *and* their data incurs M row buffer misses.

This technique comes in two versions. To reduce the number of costly write backs performed in PCM, one version

Policy	mcf	milc	cactus	leslie3d	soplex	sjeng	Gems	libquantum	lbm	omnetpp	astar	xalanc	gmean
ConvCh	103	547	239	229	152	507	169	316	92	468	428	310	254
A-cnt	97	549	231	235	142	446	153	316	93	102	377	145	202
M-cnt	95	474	225	224	142	462	139	217	89	111	351	147	191
AM-cnt	95	486	220	215	141	462	125	217	89	101	351	143	186

Table 5: CPU stall cycles due to memory, per off-chip access (Refer to Table 3 for full policy names).

assigns a heavier weight to writes than reads when accounting for the number of accesses to a row (i.e., write accesses count as $W = 3$ read accesses). Figure 5 shows that this technique improves performance compared to the AM-COUNT version with no write weights. This allows for the timely promotion of frequently written data to DRAM, where their writes can be coalesced.

For most benchmarks, AM-COUNT with write weights provides the most benefit compared to the previous policies, performing the least amount of data migrations, yet showing twice the amount of PCM row buffer hit accesses compared to A-COUNT, and furthermore exhibiting the smallest CPU stall times. This is due to (1) the reduction of low-reuse data migrations (provided by A-COUNT), (2) the reduction in migrations of frequently row buffer hitting data (provided by M-COUNT), and (3) the write coalescing provided by accelerating the promotion of frequently-written data to DRAM.

4.4 A dynamic threshold promotion policy

One drawback of the previously mentioned AM-COUNT policy is that the row access (A) and row buffer miss (M) thresholds must be determined statically by profiling a set of workloads. We propose a *dynamic threshold* policy, DAM-COUNT, which adjusts the value of A by performing a cost-benefit analysis of cycles spent migrating data versus cycles saved due to the migrated data every time quantum and applying a simple hill-climbing algorithm to adjust A in a way that maximizes the net benefit.

We reduce the search space of our algorithm by dynamically adjusting only the A threshold. We empirically observed that for our benchmarks, $M = 2$ achieves the best average performance for all $A, M \in [0, 10]$. Intuitively, values of $M > 2$ require suffering additional row buffer misses in PCM before promoting a row, and those row buffer misses could have been serviced from DRAM had the row been promoted more quickly. In addition, all rows will not be present in the row buffer the first time they are accessed, so setting $M = 1$ is equivalent to the A-COUNT policy, which makes promotion decisions with no row buffer locality information. We find setting $M = 2$ to offer a good indication of frequent row buffer misses without suffering many row buffer misses before making a promotion decision.

Each quantum, our technique approximates the *first order* costs and benefits resulting from the access threshold (A) which was used. Cost is the number of cycles the channels were occupied while migrating rows between PCM and DRAM and benefit is the reduction in latency (number of cycles saved) resulting from data located in DRAM which miss in the row buffer. Our algorithm adjusts the row access threshold A in a way which maximizes net benefit ($Benefit - Cost$).

Cost. We compute cost as the number of migrations performed during the previous quantum times the number of cycles the busses must remain occupied during a migration (a constant system parameter):

$$Cost = Migrations \times t_{migration}. \quad (1)$$

Benefit. We compute benefit as the product of DRAM array reads performed and the difference between the latency of performing an array read on PCM versus DRAM (i.e., the number of cycles saved by performing the array read on DRAM), **plus** the product of DRAM array writes performed and the difference between the latency of performing an array write on PCM versus DRAM (i.e., the number of cycles saved by performing the array write on DRAM). Note the latencies are constants. We consider reads and writes separately because of the different array access latencies in PCM they can incur.

$$Benefit = Reads_{DRAM} \times (t_{read,PCM} - t_{read,DRAM}) + Writes_{DRAM} \times (t_{write,PCM} - t_{write,DRAM}). \quad (2)$$

Maximizing net benefit. We perform a simple hill-climbing algorithm each quantum (Figure 6) to maximize net benefit ($Benefit - Cost$). Our algorithm works as follows. If, during the previous quantum, the net benefit is negative (Line 2), the cost of the current access threshold in terms of cycles spent migrating data outweighs the benefit

```
   Each quantum (10 million cycles):
 1 Net = Benefit − Cost
 2 if Net < 0 then
 3       A++
 4 else
 5       if Net > PreviousNet then
 6             A++
 7       else
 8             A--
 9       end
10 end
11 PreviousNet = Net
```

Figure 6: Cost-benefit maximization algorithm.

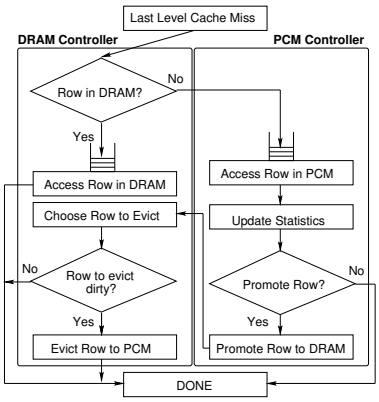


Figure 7: Memory controller promotion policy support.

of migrating such data, and the access threshold A is incremented to reduce the number of low-benefit migrations performed. Else, during the previous quantum the net benefit was positive. If the net benefit from the previous quantum is greater than the net benefit from the quantum *before* that (*PreviousNet* on Line 5), we check if net benefit will continue increasing by incrementing the access threshold A. Otherwise, net benefit is decreasing and we revert to the previous access threshold (which had the largest net benefit). Our algorithm assumes net benefit as a function of A is concave, which, across all the benchmarks we have tested, is the case.

Notice the DAM-COUNT policy closely tracks the performance of the best static threshold policy for most benchmarks without requiring any profiling information. In real world scenarios (and in our multicore evaluation in Section 6) the ability of DAM-COUNT to adjust to a suitable row access threshold at runtime is essential because profiling all workload mixes for the best static threshold is infeasible.

4.5 A note on other promotion policies

We have highlighted promotion policies which make their decisions based on data reuse and row buffer locality, but promotion criteria need not be limited to such memory access characteristics. We explored several more complex promotion policies which use the instruction address (PC) of memory requests to predict whether data should be promoted or not. Depending on whether rows first accessed by a memory request from a particular instruction eventually led to frequent row buffer misses, we predict whether promoting rows might be beneficial on their first access. The policies we designed provided similar performance to our DAM-COUNT mechanism, while incurring additional complexity. Due to space limitations we do not include these results and leave such techniques a future direction to explore.

5 Implementation and hardware cost

We will briefly discuss three micro-architecture changes necessary to support promotion policies in a hybrid memory system: we must (1) track which rows reside in DRAM, (2) track row reuse and row buffer locality information for a subset of rows in PCM, and (3) enforce promotion policies.

Tracking row location. To determine if a row is located in DRAM, we use a tag store in the DRAM memory controller (much like an on-chip tag store). Such a tag store, for 16 MB of DRAM per core, 2 KB rows, 16-way set-associativity, 16 Line Level WriteBack bits per row (one for each L1/L2 cache block in a row, cf. Section 3.1), and a 44-bit address space, would require $[(16\text{ MB})/(2\text{ KB rows})] \times (44\text{ address bits} - 20\text{ index and offset bits} + 16\text{ LLWB bits} + 1\text{ valid bit} + 4\text{ LRU bits}) = 45$ KB of storage per core. As ongoing research, we are investigating ways to efficiently store DRAM cache tag data.

Tracking row buffer locality and reuse. We store row buffer locality and row reuse information in a hardware structure called the *statistics (or stats) store*. The stats store tracks information for a *subset* of the rows located in PCM and its structure is similar to the DRAM tag store, but additionally stores row buffer locality and row reuse information about each of its entries.

We present results for a stats store of unlimited size. We do this to observe the effects of different promotion policies in isolation of implementation-dependent stats store parameters. For reference, across 200 randomly-generated

16-core workloads, we find that a 16-way 32-set LRU-replacement stats store (∼3.3 KB) with LRU replacement achieves performance within 4% of an unlimited-sized stats store.

Enabling promotion policies. Figure 7 shows the steps taken by a memory controller to support promotion policies. When a memory request is issued from the CPU, the DRAM memory controller's tag store is indexed with the row address to see if the requested row resides in DRAM or PCM. The request is then placed in the appropriate request queue, where it will stay until scheduled.

If a request is scheduled in PCM, the address of the row containing the data is used to index the stats store and a value corresponding to the number of row accesses is incremented if the request is a read and increased by a value W if the request is a write (cf. Section 4.3); if the row content is not present in the row buffer, a value corresponding to the number of row buffer misses is also incremented.

After the requested memory block from PCM (whose access is on the critical path) is sent back to the CPU, the promotion policy is invoked. If the number of row accesses is greater than a threshold A *and* the number of row buffer misses is greater than a threshold M, then the row is copied to the write buffer in the DRAM memory controller (i.e., promoted). A replacement policy[3] in DRAM is used to determine a row to replace. Note that if the row was not modified while in DRAM, it does not need to be written back to PCM, otherwise, only the *dirty* contents of the row are sent to PCM to be written back.

Updating the stats store and invoking the promotion policy are not on the critical path, because they can be performed in parallel with accessing and transferring the critical word from PCM. If needed, the stats store's associative comparison logic can take multiple cycles to access, as the time taken to read data from PCM is on the order of hundreds of CPU cycles.

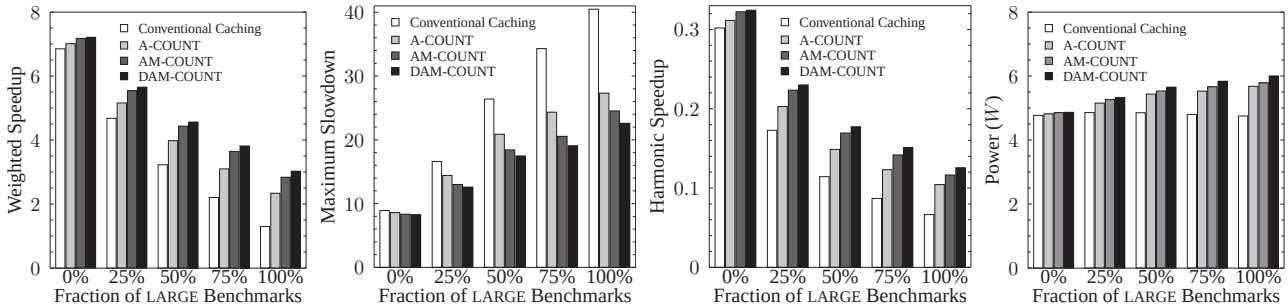
6 Experimental results

6.1 16-core results

We first analyze the performance and fairness of our technique, DAM-COUNT, on a 16-core system compared to three promotion policies: conventional caching, A-COUNT, and AM-COUNT. Note that for A- and AM-COUNT results, it was not feasible to find the best static A threshold for the large number of workloads surveyed. We instead show results for $A = 4$ and $M = 2$, which were found to be effective over a wide range of workloads. We show the A-COUNT results for reference to illustrate the limitations of a promotion policy that only considers row reuse (and not row buffer locality).

Data were collected for the initial 100 million cycles of a simulation run following a 10 million cycle warm-up period. The applications we use have relatively small working set sizes at reasonable simulation lengths and to ensure that we study the problem of data placement in hybrid memory systems properly, we set the DRAM size such that the working sets do not reside mainly in the DRAM cache. Longer simulations showed results consistent with shorter ones.

[3]We find least *frequently* used to perform the best, however, the performance of least *recently* used follows very closely.



(a) Weighted speedup. (b) Maximum slowdown. (c) Harmonic speedup. (d) Power.

Figure 8: Performance, fairness, and power results for a 16-core system.

Performance. We use the *weighted speedup* metric [30] to compute multicore performance. Weighted speedup is the sum of the IPCs of the benchmarks in a workload when run together compared to when run alone on the same system, $WeightedSpeedup = \sum \frac{IPC_{together}}{IPC_{alone}}$. Figure 8(a) presents the weighted speedup for 100 randomly-generated workload mixes in each working set fraction category[4]. Unless otherwise noted, IPC_{alone} is measured on a system with conventional caching. Three observations are worth noting.

First, as the fraction of LARGE benchmarks in the workload increases, the benefit from DAM-COUNT increases. More LARGE workloads, by definition, generate a larger number of data references to unique addresses. The conventional caching policy promotes all rows the first time they are accessed and generates a large amount of memory channel contention to promote data that may not have much reuse nor miss in the row buffer frequently. The A-COUNT policy reduces the amount of channel contention by filtering out the promotion of data with little reuse. The AM- and DAM-COUNT policies are able to offer further benefits by promoting only rows which exhibit high reuse and experience frequent row buffer misses as well, thus making better use of the limited DRAM space.

Second, DAM-COUNT performs better than the static AM-COUNT policy. This is because DAM-COUNT is able to adjust to a better *A* threshold for a given workload at runtime as opposed to assuming one fixed threshold for every type of workload.

Third, the A-, AM-, and DAM-COUNT policies are beneficial even for the 0% LARGE workloads because they distribute memory requests between both the DRAM and PCM channels, by keeping data with high row buffer locality in PCM where it can be accessed with little penalty compared to DRAM. This can reduce the contention caused by requests being serviced mainly from DRAM (as is the case with conventional caching which promotes all data to DRAM) and can allow applications to benefit from *channel-level parallelism* by simultaneously servicing memory requests from both DRAM and PCM.

Fairness. For fairness, the *maximum slowdown* [2] metric is used. Maximum slowdown is the largest slowdown seen by a benchmark in a workload when run alone versus when run together with other benchmarks on the same system, $MaxSlowdown = \max \frac{IPC_{alone}}{IPC_{together}}$. In addition, the *harmonic speedup* metric [18] (or the harmonic mean of individual benchmark speedups) has been proposed to measure both multicore performance and fairness, where $HarmonicSpeedup = \frac{NumBenchmarks}{\sum \frac{IPC_{alone}}{IPC_{together}}}$.

Our technique is able to provide both the smallest maximum slowdown (Figure 8b) as well as the best balance between performance and fairness according to the harmonic speedup metric (Figure 8c).

The A-, AM-, and DAM-COUNT policies improve fairness by a greater amount as the fraction of LARGE benchmarks in a workload increases. This is because such workloads generate a larger number of data references, to different addresses, and cause greater interference between different threads accessing shared resources (such as the memory channels or the DRAM cache). Conventional caching allows all threads to use the shared resources in a free-for-all manner: every access of every thread promotes a row, utilizing channel bandwidth and DRAM space, even if it does not have much reuse or miss frequently in the row buffer. Our row reuse- and row buffer locality-aware schemes predict which data will provide benefit from being placed in the DRAM cache and limit the use of the shared memory channels and shared DRAM cache to only those data, reducing resource contention among the threads.

Power. Since our 16-core workloads run for a fixed amount of time (100 million cycles) and not a fixed amount of work, we cannot make a definitive statement about the energy efficiency (work done per Joule) of different policies. Therefore, we instead compare the steady-state power of our different techniques (see Table 1 for power model information). We only model main memory energy, yet main memory energy may contribute a sizable amount to total system energy—up to 40% in some servers [9, 17].

Figure 8(d) shows that our techniques increase main memory power (notice, though, that we increase performance by a factor that is two-fold of that in power). Our techniques leave more data on PCM compared to conventional caching, reducing channel contention but causing a greater proportion of accesses to the PCM array which requires more energy compared to DRAM. This effect increases as the number of LARGE benchmarks contending for the DRAM cache increases, though the reduction in migration operations and write-aware policies help mitigate this power cost.

Versus all PCM and all DRAM. Figure 9 shows performance, fairness, and power for the DAM-COUNT policy compared to aggressive all PCM and all DRAM main memory systems, averaged across 200 randomly-generated

[4]From a data placement perspective, we expect multi-threaded workloads to resemble multi-programmed workloads consisting of homogeneous threads. Though our techniques benefit from heterogeneity in row buffer locality and reuse, our dynamic thresholding mechanism adapts to the homogeneity of multi-threaded workloads by more selectively caching data.

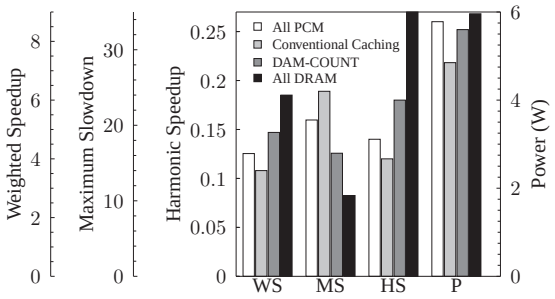


Figure 9: Versus DRAM/PCM.

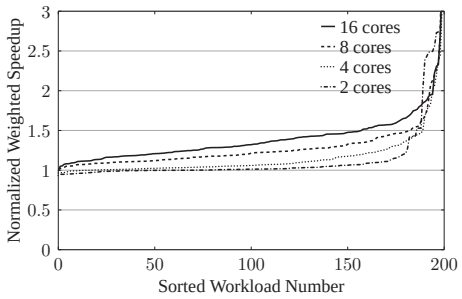


Figure 10: Number of cores.

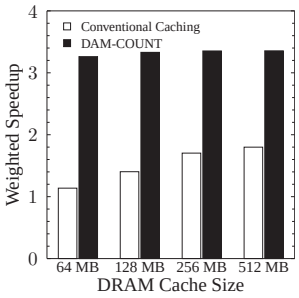


Figure 11: Effects of DRAM size.

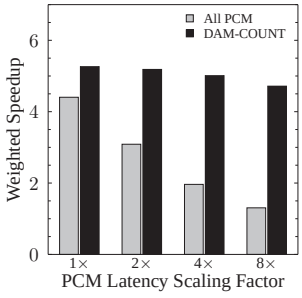


Figure 12: Effects of PCM latency.

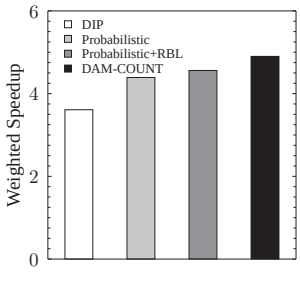


Figure 13: Related techniques.

workloads. For the all PCM and all DRAM systems, we model infinite memory capacity to fit the entire working sets of the workloads. Data are mapped to two ranks, totaling sixteen banks, across two memory controllers.

In the all PCM system, benchmarks always pay the high cost (in terms of latency and energy) for accessing PCM on a row buffer miss. On the other hand, workloads benefit from the lower latencies and energy consumptions for accessing DRAM on a row buffer miss in the all DRAM system. A hybrid memory system employing a naïve caching scheme—such as conventional caching—can have worse performance and fairness compared to an all PCM system. This is due to the high amount of channel contention introduced and the inefficient use of the DRAM cache. The same system with the DAM-COUNT policy makes efficient use of the small DRAM cache by placing in it only the rows that exhibit high reuse and frequently miss in the row buffer.

With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.

6.2 Robustness to architectural configurations

Different numbers of cores. Figure 10 shows the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup.

Our technique achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

DRAM cache size. Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads *consisting of 100%* LARGE *benchmarks*, to exercise the DRAM cache. There are two things to note.

First, even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.

Second, for smaller cache sizes, DAM-COUNT provides the most performance benefit compared to conventional caching. Prudent use of limited DRAM cache space is important; however, conventional caching promotes every row accessed, placing data with little reuse and infrequent row buffer misses the DRAM cache. Our technique, on the other hand, is less sensitive to the size of the DRAM cache because, in response to the different DRAM cache sizes, it is able to dynamically adjust the amount of data promoted to the DRAM cache (via adjusting the A threshold), ensuring that only the data with the most reuse and frequent row buffer misses get promoted.

PCM latency. Figure 12 shows the performance of our technique as the technology-dependent latencies associated with PCM (array read and array write) are varied from to $1\times$ to $8\times$ the values assumed in Table 1 (the results for $1\times$ in Figure 12 correspond to the data in Figure 9). We compare our technique to the all PCM system described in Section 6.1. (For the all PCM results, the IPC_{alone} component of weighted speedup is measured for the benchmark run alone on the all PCM system.) Results are averaged across 200 randomly-generated workloads. There are two trends to notice.

First, as the latency required to access the PCM memory array increases, the performance of the all PCM system decreases less than linearly (compared to the scaling factor). This is because accesses that hit in the row buffer reduce the number of PCM cell array accesses, and multiple banks in the PCM ranks service memory requests simultaneously.

Second, our technique performs well even when the latencies associated with PCM increase. For example, in Figure 12, when the latencies associated with PCM increase by a factor of $8\times$, our technique only slows down by 10%. This is because our technique only leaves data which mainly experience row buffer hits in PCM, where they can be serviced at the same latency as DRAM.

6.3 Comparison to L1/L2 caching schemes

The *Dynamic Insertion Policy (DIP)* [24] inserts data blocks into the LRU position of a cache with high probability (otherwise inserting it into the MRU position), such that blocks which are not reused are quickly evicted from the cache. Figure 13 compares the performance of DAM-COUNT to using DIP to manage the DRAM cache. We find DIP to perform similarly to conventional caching in a hybrid memory system. This is because although DIP is able to quickly evict rows that exhibit low reuse in the DRAM cache, it still promotes any row on its first access like a conventional cache. Thus, it is unable to reduce the memory channel contention and memory access overheads that result due to data migrations that yield low benefits. We find that DAM-COUNT increases system performance by 36% compared to using DIP across 200 randomly-generated workloads.

Probabilistic caching [7] inserts data blocks into the cache with a certain probability, such that blocks with higher reuse are more likely to be cached. In addition to normal probabilistic caching, we introduce a variant that is row buffer locality aware. The RBL-aware scheme probabilistically promotes rows only when they experience row buffer miss accesses. This causes rows that generate frequent row buffer misses to have a higher chance of being inserted into the DRAM cache. Figure 13 shows that a probabilistic scheme that is RBL and reuse aware outperforms a similar scheme that is only aware of data reuse. However, DAM-COUNT outperforms the RBL-aware probabilistic scheme by better accounting for the frequency of row buffer miss accesses experienced by rows, through the explicit bookkeeping of a stats store. Probabilistic caching performs better than DIP because it makes its decision not to cache data before the data is transferred, whereas with DIP, each row must be transferred to the DRAM cache (occupying memory channels and banks) even if it is inserted into the LRU position and promptly evicted. DAM-COUNT improves system performance compared to probabilistic caching and its RBL-aware variant by 12% and 8% respectively, across 200 randomly-generated workloads.

7 Related work

To the best of our knowledge, our study is the first to propose a data placement scheme that exploits row buffer locality in a heterogeneous memory system.

Exploiting row buffer locality. Lee et al. [13] proposed employing multiple short row buffers in PCM devices, much like an internal device cache. Their findings indicate that such a row buffer organization mitigates access latencies while reducing energy costs. This scheme alone does not take into account the degree of data reuse at the row buffers. However, employed together with our proposed mechanism, it may contribute to improved system performance by increasing the PCM row buffer hit rate.

Sudan et al. [31] presented a scheme of increasing the row buffer hit rate of memory accesses by co-locating frequently referenced sub-rows of data in a small number of reserved rows. However, it is still possible for a substantial amount of row buffer miss accesses to occur, if the sub-rows of data are accessed in a manner that causes the row buffer contents to frequently switch between the reserved rows. Our proposed mechanism works cooperatively with this scheme in a hybrid memory system, by shifting these row buffer miss accesses to occur in the DRAM cache, rather than in PCM.

Row buffer locality is commonly exploited in memory scheduling algorithms. The *First-Ready First Come First Serve algorithm (FR-FCFS)* [28] prioritizes memory requests that hit in the row buffer, therefore improving the latency, throughput, and energy cost of servicing memory requests. Lee et al. [15] proposed a memory controller that prioritizes prefetch requests that hit in the row buffer. Another study by Lee et al. [16] proposed a last-level cache writeback scheme that aggressively issues writeback requests that hit in the row buffer. All of these works are orthogonal to our proposal.

Hybrid memory systems. Qureshi et al. [23] proposed increasing the size of main memory by replacing DRAM with PCM, assuming that 4 times as much capacity can be supplied at the same cost. In addition, a DRAM cache is used to the PCM main memory. While the DRAM cache employs a conventional LRU-replacement caching policy that is oblivious to row buffer locality and the degree of cache block reuse, it is the reduction in page faults owing to the increase in main memory size that brings performance and energy improvements to the system. In our work, we study the effects past the reduced page faults, and are concerned with the effectiveness of DRAM caching to PCM.

Dhiman et al. [6] proposed a hybrid main memory system that exposes DRAM and PCM addressability to software (OS). The scheme tracks the number of writes to different pages in PCM. If the number of writes to a particular PCM page exceeds a certain threshold, the contents of the page is copied to another page, either in DRAM or PCM. This scheme mainly facilitates PCM wear-leveling, with no explicit mechanism in place to optimize for system performance. Mogul et al. [19] suggest the OS exploit metadata information available to it to make data placement decisions between DRAM and non-volatile memory. Similar to [6], their data placement criteria are centered around write frequency to the data (time to next write, or time between writes to page). The choice of data placement in neither of these schemes are row buffer locality-aware, which is what our mechanism targets specifically.

Bivens et al. [3] examine the various design concerns of a heterogeneous memory system such as memory latency, bandwidth, and endurance requirements, etc. of employing storage class memory (PCM, MRAM, and Flash, etc.). Their hybrid memory organization is similar to ours and that in [23], in that DRAM is used as a cache to a slower memory medium, transparently to software. Phadke et al. [21] propose profiling the memory access patterns of individual applications in a multicore system, and placing their working sets in the particular type of DRAM that best suits the application's memory demands. The decision to place an application's working set in DRAM that is optimized for either latency, bandwidth, or power, depend on the application's memory intensity and memory-level parallelism (MLP). In constrast, our mechanism dynamically makes fine-grained data placement decisions at a block (row) granularity, depending on the block's row buffer locality characteristics.

Exploiting data reuse. There is a large body of work aimed at improving data locality in processor caches by caching data that is reused frequently or accessed nearby each other [8, 27, 12, 24, 10, 7]. Gonzalez et al. [8] proposed predicting and segregating data into either of two last-level caches depending on whether they exhibit spatial or temporal locality. They also proposed bypassing the cache when accessing large data structures (i.e., vectors) with large strides to prevent cache thrashing. Additionally, Rivers and Davidson [27] proposed separating data with temporal reuse from data without temporal reuse. If a block that has not been reused is evicted from the L1 cache, a bit indicating lack of temporal reuse is set for that block in the L2 cache. If blocks with such a bit set are referenced again, they are placed in a special buffer instead of the L1 cache to prevent them from polluting the L1 cache. These works are primarily concerned with L1/L2 caches that have access latencies on the order of a couple CPU clock cycles, and do not exploit the benefits of row buffer locality in a hybrid memory system.

Johnson and Hwu [12] used a counter-based mechanism to keep track of data reuse at a granularity larger than a cache block. Cache blocks in a region with less reuse bypass a direct-mapped cache if that region conflicts with another that has more reuse. Jiang et al. [11] presented a scheme of selectively caching large blocks of data on an on-chip DRAM cache to reduce off-chip memory bandwidth. The data blocks chosen for caching are those with high access counts. Ramos et al. [25] adapted a buffer cache replacement algorithm to rank pages on their frequency and recency of accesses, and placed the highest-ranked pages in DRAM, in a DRAM-PCM hybrid memory system. All of these works do not consider row buffer locality, and promote data to DRAM depending on their access characteristics, similar to our A-COUNT mechanism. However, we have shown that access characteristics alone are not sufficient criteria for making caching decisions in a hybrid memory due to the impact of row buffer locality on performance.

Our DAM-COUNT mechanism is able to achieve significantly higher performance (by 14.6%) and fairness (by 14.5%) than A-COUNT by factoring in row buffer locality into the decision on which blocks to insert into the cache.

8 Conclusion

We presented row buffer locality-aware data placement schemes for hybrid memories. In particular, we proposed DAM-COUNT, which achieves the best system performance, energy efficiency, and fairness for a hybrid DRAM-PCM memory system. DAM-COUNT selectively promotes data that generate frequent row buffer misses, to DRAM. This scheme allows data that is mostly accessed as row buffer hits to leverage the large PCM capacity while being accessed efficiently. Furthermore, data migrations that are unbeneficial for performance are pruned. This mechanism also makes the best use of limited DRAM capacity by only filling it with data that repeatedly necessitate memory cell array accesses. To our knowledge, our data placement schemes are the first to exploit row buffer locality in hybrid memories.

We evaluated our proposed mechanism and showed that it outperforms related techniques. We also demonstrated its robustness to high PCM latencies and restricted DRAM capacities. Furthermore, DAM-COUNT is able to dynamically adapt to workloads and system configurations to maintain high system throughput. We conclude that DAM-COUNT is a high performance, energy efficient means to manage hybrid memories for multicore systems.

Acknowledgments

We gratefully acknowledge members of the SAFARI research group and CALCM for many insightful discussions on this work. HanBin Yoon was supported by a Samsung Scholarship PhD fellowship while conducting this work. This research was partially supported by an NSF CAREER Award CCF-0953246, NSF Grant CCF-1147397, Gigascale Systems Research Center, Intel Corporation ARO Memory Hierarchy Program, and Carnegie Mellon CyLab. We also acknowledge equipment and gift support from Intel, Samsung, and Oracle.

References

[1] K. Bailey et al. Operating system implications of fast, cheap, non-volatile memory. HotOS, 2011.

[2] M. Bender et al. Flow and stretch metrics for scheduling continuous job streams. Symp. on Discrete Algorithms, 1998.

[3] A. Bivens et al. Architectural design for next generation heterogeneous memory systems. Intl. Memory Workshop '10.

[4] J. Coburn et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. ASPLOS'11.

[5] J. Condit et al. Better I/O through byte-addressable, persistent memory. SOSP, pages 133–146. ACM, 2009.

[6] G. Dhiman et al. PDRAM: a hybrid PRAM and DRAM main memory system. DAC, pages 664–469. ACM, 2009.

[7] Y. Etsion et al. L1 cache filtering through random selection of memory references. PACT, pages 235–244, 2007.

[8] A. González et al. A data cache with multiple caching strategies tuned to different types of locality. ICS '95.

[9] H. Huang et al. Improving energy efficiency by making DRAM less randomly accessed. ISLPED, 2005.

[10] A. Jaleel et al. Adaptive insertion policies for managing shared caches. PACT, pages 208–219. ACM, 2008.

[11] X. Jiang et al. CHOP: Adaptive filter-based dram caching for CMP server platforms. HPCA, pages 1–12, 2010.

[12] T. L. Johnson and W.-m. W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. ISCA '97.

[13] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. ISCA, pages 2–13. ACM, 2009.

[14] B. C. Lee et al. Phase-change technology and the future of main memory. *IEEE Micro*, 30:143–143, January 2010.

[15] C. J. Lee et al. Prefetch-aware dram controllers. MICRO, pages 200–209. IEEE Computer Society, 2008.

[16] C. J. Lee et al. DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. Technical report, The University of Texas at Austin, 2010.

[17] C. Lefurgy et al. Energy management for commercial servers. *IEEE Computer*, 36, December 2003.

[18] K. Luo et al. Balancing thoughput and fairness in smt processors. ISPASS, pages 164 –171, 2001.

[19] J. C. Mogul et al. Operating system support for NVM+DRAM hybrid main memory. HotOS, 2009.

[20] H. Patil et al. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. MICRO'04.

[21] S. Phadke et al. MLP aware heterogeneous memory system. DATE, pages 1–6, march 2011.

[22] M. K. Qureshi et al. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. MICRO'09.

[23] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. ISCA'09.

[24] M. K. Qureshi et al. Adaptive insertion policies for high performance caching. ISCA, pages 381–391. ACM, 2007.

[25] L. E. Ramos et al. Page placement in hybrid memory systems. ICS '11, pages 85–95.

[26] S. Raoux et al. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.*, 52:465–479, July 2008.

[27] J. Rivers and E. Davidson. Reducing conflicts in direct-mapped caches with a temporality-based design. ICPP '96.

[28] S. Rixner et al. Memory access scheduling. ISCA, pages 128–138. ACM, 2000.

[29] Semiconductor Industry Association. The international technology roadmap for semiconductors. `http://www.itrs.net/Links/2010ITRS/2010Update/ToPost/2010Tables_LITHO_FOCUS_D_ITRS.xls`, 2010.

[30] A. Snavely et al. Symbiotic jobscheduling for a simultaneous multithreading processor. ASPLOS, 2000.

[31] K. Sudan et al. Micro-pages: increasing DRAM efficiency with locality-aware data placement. ASPLOS, 2010.

[32] H. Wong et al. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[33] W. Zhang et al. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. PACT, pages 101–112. IEEE Computer Society, 2009.