Row Buffer Locality Aware Caching Policies for Hybrid Memories

HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding and Onur Mutlu
Carnegie Mellon University
{hanbinyoon,meza,rachata,onur}@cmu.edu, rhardin@mit.edu

***Abstract*—Phase change memory (PCM) is a promising technology that can offer higher capacity than DRAM. Unfortunately, PCM's access latency and energy are higher than DRAM's and its endurance is lower. Many DRAM-PCM hybrid memory systems use DRAM as a cache to PCM, to achieve the low access latency and energy, and high endurance of DRAM, while taking advantage of PCM's large capacity. A key question is what data to cache in DRAM to best exploit the advantages of each technology while avoiding its disadvantages as much as possible.**

We propose a new caching policy that improves hybrid memory performance and energy efficiency. Our observation is that both DRAM and PCM banks employ row buffers that act as a cache for the most recently accessed memory row. Accesses that are row buffer hits incur similar latencies (and energy consumption) in DRAM and PCM, whereas accesses that are row buffer misses incur longer latencies (and higher energy consumption) in PCM. To exploit this, we devise a policy that avoids accessing in PCM data that frequently causes row buffer misses because such accesses are costly in terms of both latency and energy. Our policy tracks the row buffer miss counts of recently used rows in PCM, and caches in DRAM the rows that are predicted to incur frequent row buffer misses. Our proposed caching policy also takes into account the high write latencies of PCM, in addition to row buffer locality.

Compared to a conventional DRAM-PCM hybrid memory system, our row buffer locality-aware caching policy improves system performance by 14% and energy efficiency by 10% on data-intensive server and cloud-type workloads. The proposed policy achieves 31% performance gain over an all-PCM memory system, and comes within 29% of the performance of an all-DRAM memory system (not taking PCM's capacity benefit into account) on evaluated workloads.

I. Introduction

Multiprogrammed workloads on chip multiprocessors require large amounts of main memory to support the working sets of many concurrently executing threads. This memory demand is increasing today as the number of cores on a chip continues to increase and data-intensive applications become more widespread. *Dynamic Random Access Memory (DRAM)* is used to compose main memory in modern computers. Though strides in DRAM process technology have enabled DRAM to scale to smaller feature sizes and thus higher densities (capacity per unit area), it is predicted that DRAM density scaling will become costly as feature size continues to reduce [33], [26], [16]. Satisfying increasingly higher memory demands with DRAM exclusively will become expensive in terms of both cost and energy.

Phase Change Memory (PCM) is an emerging random-access memory technology that offers a competitive alternative to DRAM. A PCM cell stores data as varying electrical resistance, which is more amenable to extreme scaling than a DRAM cell that stores data as a small amount of electrical charge. Hence, PCM is expected to scale to smaller feature sizes (higher capacities) than DRAM [26], [12], [11]. PCM's non-volatility also leads to opportunities for lowering OS overhead and increasing I/O performance through the use of persistent memory [3], [4]).

However, PCM has a number of disadvantages that prevent its adoption as a direct DRAM replacement. First, PCM exhibits higher access latencies compared to DRAM. Second, PCM has higher dynamic energy consumption than DRAM. Finally, PCM has finite write endurance (memory cells wear out with increasing write operations).

Hybrid memory systems comprising both DRAM and PCM aim to benefit from the large memory capacity offered by PCM, while achieving the low latency, low energy, and high endurance of DRAM. Previous proposals employ modestly-sized DRAM as a cache to large capacity PCM [24], or use the DRAM to store frequently written data [5], [36]. A key question in the design of a DRAM-PCM hybrid memory system is how to place data between DRAM and PCM to best exploit the strengths of each technology while avoiding its weaknesses as much as possible.

In this work, we develop new mechanisms for deciding how data should be placed in a DRAM-PCM hybrid memory system. Our main observation is that both DRAM and PCM banks employ row buffer circuitry. The row buffer acts as a cache for the most recently accessed row in the bank, and memory requests that hit in the row buffer incur similar latencies and energies in DRAM and PCM [11], [12]. However, requests that miss in the row buffer require loading (activating) the requested row from the memory cell array to the row buffer, which incurs higher latency and energy in PCM than in DRAM. As a result, placing data that mostly leads to row buffer hits (has high row buffer locality) in DRAM provides little benefit over placing the same data in PCM, whereas placing heavily reused data that leads to frequent row buffer misses (has low row buffer locality) in DRAM avoids the high latency and energy of PCM array accesses.

Based on this observation, we devise a hybrid memory caching policy which caches to DRAM the rows of data for which there are frequent requests (highly reused) that mostly miss in the row buffer. To implement this policy, the memory controller maintains a count of the row buffer misses for recently used rows in PCM, and places in DRAM the data of rows whose row buffer miss counts exceed a certain threshold (dynamically adjusted at runtime). When a row is written to, its row buffer miss count is incremented by a larger amount than when it is read, expediting the caching of write data to avoid the high write latency and energy of PCM. We observe that our mechanism (1) mitigates the high access latency and energy cost of PCM array accesses, (2) reduces memory channel bandwidth consumption caused by the frequent movement of data between DRAM and PCM, and (3) balances the memory

Rachael A. Harding is currently a PhD student at the Massachusetts Institute of Technology.

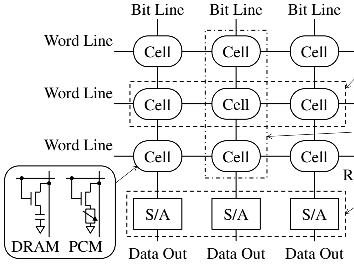


Fig. 1: Memory cells organized in a 2D array of rows and columns.

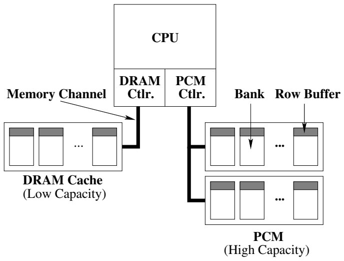


Fig. 2: Hybrid memory system organization.

access load between DRAM and PCM, altogether leading to improved system performance and energy efficiency.

Contributions. We make the following contributions:

- We identify row buffer locality as a key metric for making data placement decisions in a hybrid memory system.
- Based on our observation, we develop a row buffer locality-aware hybrid memory caching policy to selectively cache data with low row buffer locality and high reuse. Our scheme also facilitates timely caching for write data to account for PCM's high write latency and energy.
- We evaluate our proposed scheme using data-intensive server and cloud-type workloads on a 16-core system. Compared to a conventional caching policy that places frequently accessed data in DRAM (oblivious of row buffer locality), our scheme improves system performance by 14% and energy efficiency by 10%. Our scheme shows 31% performance gain over an all-PCM memory system, and comes within 29% of the performance of an all-DRAM memory system (not taking PCM's capacity benefit into account).

II. Background

A. Memory Device Architecture: Row Buffers

The organization of a memory device is illustrated in Figure 1. Cells (memory elements) are typically laid out in arrays of rows (cells sharing a common word line) and columns (cells sharing a common bit line). Accesses to the array occur in the granularity of rows. To read from the array, a word line is first asserted to select a row of cells. Then through the bit lines, the selected cells' contents are detected by sense amplifiers (S/A) and latched in peripheral circuitry known as the *row buffer*.

Once the contents of a row are latched in the row buffer, subsequent memory requests to that row are served promptly from the row buffer, without having to bear the delay of accessing the array. Such memory accesss are called *row buffer hits*. However, if a row different from the one latched in the row buffer is requested, then the newly requested row is read from the array to the row buffer (replacing the row buffer's previous contents). Such a memory access incurs the high latency and energy of activating the array, and is called a *row buffer miss*. *Row Buffer Locality (RBL)* refers to the repeated reference to a row while its contents are in the row buffer. Memory requests to data with high row buffer locality are served efficiently (at low latency and energy) without having to frequently activate the memory cell array.

B. Phase Change Memory

Phase Change Memory (PCM) is a non-volatile memory technology that stores data by varying the electrical resistance of a material known as chalcogenide [34], [26]. A PCM memory cell is programmed by applying heat (via electrical current) to the chalcogenide and then cooling it at different rates, depending on the data to be stored. Rapid quenching places the chalcogenide into an amorphous state which has high resistance, representing the bit '0', and slow cooling places the chalcogenide into a crystalline state which has low resistance, representing the bit '1'.

A key advantage of PCM is that it is expected to offer higher density (hence memory capacity) than DRAM. This is due to two reasons: (1) a PCM cell stores data in the form of resistance, which is expected to scale to smaller feature sizes than charge-based storage in a DRAM cell, (2) PCM offers multi-level cell capability (unlike DRAM), which stores more than one bit of data per memory cell by achieving more than two distinguishable resistance levels for each cell.

However, PCM has a number of disadvantages compared to DRAM. The long cooling duration required to crystallize chalcogenide leads to high PCM write latency, and read (sensing) latency for PCM is also longer than that for DRAM. A technology survey of nine recent PCM devices and prototypes [11] reported PCM read and write latencies of 4–6× and 6–32× those of DRAM, respectively. In addition, PCM read and write energies were found to be 2× and 10–140× those of DRAM, respectively. Furthermore, the repeated thermal expansions and contractions of a PCM cell during programming lead to finite write endurance, which is estimated at 10^8 writes, an issue not present in DRAM.

C. DRAM-PCM Hybrid Memory Systems

DRAM-PCM hybrid memory systems [24], [5], [36], [17] aim to combine the strengths of the two memory technologies. Typical designs employ DRAM as a small cache or write buffer to PCM of large capacity. PCM provides increased overall memory capacity that leads to reduced page faults in the system, while the DRAM cache serves a large portion of the memory requests at low latency and low energy with high endurance. The combined effect increases overall system performance and energy efficiency [24].

Figure 2 illustrates the organization of a DRAM-PCM hybrid memory system. Qureshi et al. [24] proposed using DRAM as a 16-way set associative cache to PCM with 4 KB block sizes. In their design, a 1 GB DRAM is managed by hardware to cache data for 32 GB of PCM, using a tag store of 1 MB in SRAM. More recent studies [15], [17] propose alleviating the overhead of an SRAM tag store by storing the tag information in (DRAM) memory, which also makes finer-grained caching (smaller block sizes) more feasible. For each DRAM cache block (4 KB), processor cache blocks (256 B) that become dirty are tracked and selectively written back to PCM when the DRAM cache block is evicted from DRAM. Known as *Line Level WriteBack*, this technique reduces the number of costly writes to PCM.

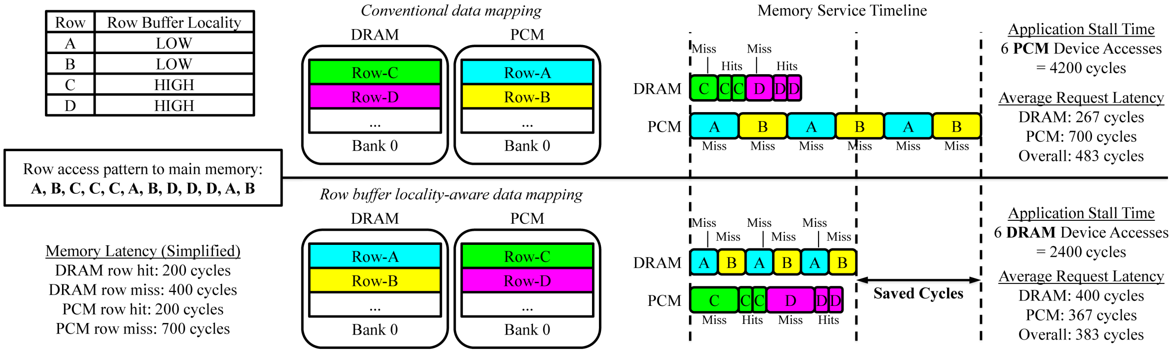


Fig. 3: Conceptual example showing the importance of row buffer locality-awareness in hybrid memory data placement decisions.

III. Motivation

A key question in designing a high-performance hybrid memory system is deciding what data to place in the DRAM cache. Unlike an on-chip L1/L2 SRAM cache, an off-chip DRAM cache exhibits significantly higher latencies (on the order of hundreds of processor clock cycles). Furthermore, data movements on the order of a kilobyte at a time occupy large amounts of memory bandwidth. Influenced by these additional factors, a hybrid memory system requires an effective DRAM cache management policy to achieve high performance. Current hybrid memory and on-chip cache management proposals seek to improve the reuse of data placed in the cache and reduce off-chip memory access bandwidth (e.g., [25], [7]). We show in this paper that the *row buffer locality* of data placed between the DRAM and PCM devices in a hybrid main memory system can play a crucial role in determining average memory access latency and system performance.

The example in Figure 3 illustrates how row buffer locality-oblivious data placement can result in suboptimal application performance. In this figure, there are twelve requests which arrive at the memory system in the *order* shown and we display the abstract service timelines of these requests assuming conventional data mapping and row buffer locality-aware data mapping. Note that the requests may *not* all be present at the memory controller's request buffer at the same time (i.e., the figure shows the order of requests, not necessarily arrival times), so a memory scheduling policy that prioritizes row buffer hit requests (e.g., [28], [37]) may not be able to take advantage of row buffer locality present in the requests.[1] For simplicity, we assume that requests from the processor access rows A and B in such a way that requests to those rows never hit in the row buffer (low row buffer locality) and access rows C and D in such a way that the requests frequently hit in the row buffer (high row buffer locality). If these requests were issued from the same core, serving them more quickly would allow the waiting core to stall less, resulting in improved performance; if these requests were from different cores, serving them more quickly would allow other requests to in turn be served, improving system performance. Without loss of generality, we assume, for this conceptual example, a single core system where DRAM and PCM each have a single bank.

With a *conventional data mapping* scheme which is unaware of row buffer locality (such as previously proposed caching policies [25], [7]), it is possible to map rows A and B to PCM and rows C and D to DRAM (as shown in the top half of Figure 3). Given the access pattern described, requests to rows A and B mostly miss in the PCM row buffer and access the PCM device at high array access latency and energy. Requests to rows C and D frequently hit in the row buffer and thus do not receive much benefit from being placed in DRAM, compared to when being placed in PCM where the row buffer hit latency and energy is similar [11], [12].

In contrast, a *row buffer locality-aware* caching mechanism would map rows with high row buffer locality in PCM and low row buffer locality in DRAM (bottom half of Figure 3). This decision leads to a reduced amount of time and energy taken to serve all memory requests. This is because the data for accesses which frequently lead to row buffer misses is placed in DRAM, where the array access latency and energy are less than in PCM. These requests are now served faster, and system performance and energy efficiency improves.[2]

The crucial observation is that *DRAM and PCM devices both employ row buffers which can be accessed at similar latencies and energies:* application stall time (and average request latency) can be reduced if data which frequently misses in the row buffer is placed in DRAM as opposed to PCM, while placing data which frequently hits in the row buffer in PCM does not increase application stall time compared to placing that data in DRAM.

Our goal is to develop a mechanism that identifies rows with low row buffer locality (and high reuse) which would benefit from being placed in DRAM. Based on the observation that both DRAM and PCM devices employ row buffers, we design a dynamic caching policy for hybrid main memories which identifies data that frequently incurs row buffer misses and caches such data in DRAM.

IV. Row Buffer Locality-Aware Caching

Overview. We propose a *Row Buffer Locality-Aware (RBLA)* caching mechanism that places in DRAM rows which have low row buffer locality, to benefit from the lower array access latency and energy of DRAM compared to PCM. To do so, the numbers of row buffer misses that rows in PCM experience are measured, and rows with high row buffer miss counts (i.e., rows with low row buffer locality and high reuse) are placed in DRAM. To determine what is a suitably high row buffer miss count that indicates sufficiently low row buffer locality, we develop a dynamic policy, *RBLA-Dyn*, which determines the appropriate row buffer miss count level at

[1] In all of our evaluations we assume an FR-FCFS memory scheduler [28], [37], which prioritizes row buffer hit requests.

[2] Even though the figure shows some requests being served in parallel, if the individual requests arrived in the same order at different times, the average request latency would still be improved significantly.

runtime. This scheme adapts the RBLA mechanism to varying workload and system characteristics of a many-core system.

A. Measuring Row Buffer Locality

The RBLA mechanism tracks the row buffer locality statistics for a small number of recently-accessed rows, in a hardware structure called the *stats store* in the memory controller. The stats store is structurally organized as a cache, however its data payload per entry is a single row buffer miss counter.

On each PCM access, the memory controller looks for an entry in the stats store using the address of the accessed row. If there is no corresponding entry, a new entry is allocated for the accessed row, possibly evicting an older entry. If the access results in a row buffer miss, the row's row buffer miss counter is incremented.[3] If the access results in a row buffer hit, no additional action is taken.

B. Triggering Row Caching

Rows that have low row buffer locality and high reuse will have high row buffer miss counter values. The RBLA mechanism selectively caches these rows by caching a row to DRAM when its row buffer miss counter exceeds a threshold value, `MissThresh`. Setting `MissThresh` to higher values causes rows with lower row buffer locality to be cached.

Caching rows based on their row buffer locality attempts to migrate data between PCM and DRAM only when it is useful to do so. This affects system performance in several ways. First, placing in DRAM rows which have low row buffer locality improves average memory access latency, due to the lower row buffer miss (array access) latency of DRAM compared to PCM. Second, by selectively caching data that benefits from being migrated to DRAM (data that frequently hits in the row buffer is accessed as efficiently in PCM as in DRAM), RBLA reduces unnecessary data movement between DRAM and PCM. This reduces memory bandwidth consumption, allowing more bandwidth to be used to serve demand requests, and enables better utilization of the DRAM cache. Third, allowing data that frequently hits in the row buffer to remain in PCM contributes to balancing the memory request load between the heavily contended DRAM cache and the less contended PCM.

We increase a row's row buffer miss count by a larger increment for a write access than for a read access (for which the increment is 1). This expedites the caching of write data to avoid incurring high PCM write latency and energy. We increment the row buffer miss count on a write access by 4, which is the approximate array write latency to array read latency ratio in the modeled PCM.

To prevent rows with low reuse from gradually building up large enough row buffer miss counts over a long period of time to exceed `MissThresh`, we apply a periodic reset to all of the row buffer miss count values. We set this reset interval to 10 million cycles empirically.

An appropriate `MissThresh` value may be determined through profiling the workload under a given system. However, we empirically determined that no single static value provides the best performance either within a single application or across different applications [35]. In addition, finding a single static value for every possible combination of co-running threads and system configuration is difficult for many-core systems. We next propose a mechanism for dynamically adjusting the value of `MissThresh` at runtime to adapt to workload behavior.

C. Dynamic Threshold Adaptation

The RBLA-Dyn mechanism improves the adaptability of RBLA to workload and system variations by dynamically determining `MissThresh`. The key observation behind RBLA-Dyn is that *the number of cycles saved by caching rows in DRAM should outweigh the bandwidth cost of migrating that data to DRAM*. RBLA-Dyn estimates, on an interval basis, the first order costs and benefits of employing certain `MissThresh` values, and increases or decreases the `MissThresh` value to maximize the net benefit.

Since data migration operations can delay demand requests, for cost, we compute the number of cycles spent migrating data across the memory channels (Equation 1). If these data migrations are eventually beneficial, the access latency to main memory will be reduced. Hence for benefit, we compute the number of cycles saved by accessing the data from the DRAM cache as opposed to PCM (Equation 2).

$$\begin{aligned} \mathit{Cost} &= \mathit{NumMigrations} \times t_{migration} && (1) \\ \mathit{Benefit} &= \mathit{NumReads}_{dram} \times (t_{read,pcm} - t_{read,dram}) + && (2) \\ &\quad \mathit{NumWrites}_{dram} \times (t_{write,pcm} - t_{write,dram}) \end{aligned}$$

Over the course of an interval (10 million cycles in our setup), a negative net benefit ($\mathit{Benefit} - \mathit{Cost}$) implies that with the assumed `MissThresh` value, more cycles are spent migrating data (preventing the memory bandwidth from being used to serve demand requests) than saved by placing data in the DRAM cache. In other words, the small benefit gained by having data cached does not justify the large cost of migrating that data. In such a situation, it is desirable to reduce the number of data migrations performed and selectively migrate the data that achieves higher benefit from being placed in the DRAM cache. This is done by increasing the value of `MissThresh`, which has the effect of requiring rows to have lower row buffer locality in order to be cached in DRAM.

Conversely, a positive net benefit implies that with the assumed `MissThresh` value, migrating data to DRAM saves more cycles than those expended to migrate the data, which benefits system performance. In such a situation, it could be the case that increasing `MissThresh` gains further benefit by more selectively migrating data and reducing the migration cost. However, it could also be the case that decreasing `MissThresh` (relaxing the row buffer locality condition on caching) gains further benefit by caching more data and having it accessed from DRAM, perhaps for a small amount of additional bandwidth cost. We leverage these observations to develop an algorithm that dynamically adjusts the value of `MissThresh`.

RBLA-Dyn uses a simple hill-climbing algorithm that is evaluated at the end of each interval to adjust `MissThresh` in the direction of increasing net benefit, eventually converging at the point of maximum net benefit (Algorithm 1). `MissThresh` is incremented when the net benefit is negative for the previous interval. If the net benefit is positive for the previous interval, we check to see whether it is greater than the net benefit for the interval before that. If so, the net

[3] Updating the stats store is not on the critical path of memory access.

benefit is increasing and `MissThresh` is adjusted further in the direction (increasing or decreasing) that it was heading in. Otherwise, if the net benefit is decreasing, `MissThresh` is steered in the opposite direction. This algorithm assumes that net benefit as a function of `MissThresh` is concave, which is the case across all the workloads we have tested.

Algorithm 1: Dynamic Threshold Adjustment

```
NetBenefit = Benefit - Cost
if (NetBenefit < 0)
  MissThresh++
else if (NetBenefit > PreviousNetBenefit)
  if (MissThresh was previously incremented)
    MissThresh++
  else
    MissThresh--
else
  if (MissThresh was previously incremented)
    MissThresh--
  else
    MissThresh++
PreviousNetBenefit = NetBenefit
```

V. Implementation and Hardware Cost

The primary hardware cost incurred in implementing a row buffer locality-aware caching mechanism on top of an existing hybrid memory system is the stats store. We model a 16-way 128-set LRU-replacement stats store using 5-bit row buffer miss counters, which occupies 9.25 KB.[4] This store achieves the system performance within 0.3% (and memory lifetime within 2.5%) of an unlimited-sized stats store for RBLA-Dyn.

Logic in the memory controller is required to implement dynamic threshold adjustment (Algorithm 1), and to trigger the migration of a row from PCM to DRAM when a row's row buffer miss count exceeds the `MissThresh` value.

VI. Experimental Methodology

We use a cycle-level in-house x86 multi-core simulator, whose front-end is based on Pin. The simulator models the memory hierarchy in detail. Table I shows the major system parameters used in our study.[5]

We formulate server- and cloud-type workloads for a 16-core system. We run a single-threaded instance of a benchmark on each core which is representative of many consolidated workloads for large CMP systems. 18 server-type workloads are formed using random selections of transaction processing benchmarks (TPC-C/H), and 18 cloud-type workloads are similarly formed using transaction processing, web server, and video processing benchmarks (tabulated in Table II). The representative phases of benchmarks were executed as profiled by PinPoints [22]. Results were collected for one billion cycles following a warmup period of one billion cycles.

We gauge multi-core performance using the *weighted speedup* metric [30], which is the sum of the speedups of the benchmarks when executed together on the multi-core system, compared to when executed alone on the the same system employing a frequency-based DRAM caching policy. For multi-core fairness, we use the *maximum slowdown* [1], [9], [10] metric, which is the highest slowdown (reciprocal of speedup) experienced by any benchmark.

Processor	16 cores, 4 GHz, 3-wide issue (maximum 1 memory operation per cycle), 128-entry instruction window.
L1 cache	Private 32 KB per core, 4-way, 128 B blks.
L2 cache	Shared 512 KB per core, 8-way, 128 B blks, 16 MSHRs.
Memory controller	2 controllers (DRAM and PCM) each with 64-bit channel, 128-entry read/write request queues and 128-entry migration buffer per controller, FR-FCFS request scheduling ([28], [37]), DRAM employed as 16-way LRU-replacement cache to PCM ([24]).
Memory	DRAM: 1 GB, 1 rank (8 banks), 16 KB row buffer. PCM: 16 GB, 1 rank (8 banks), 4 KB row buffer.
Timing	DRAM: row hit (miss) = 40 (80) ns. PCM: row hit (clean miss, dirty miss) = 40 (128, 368) ns.
Energy	DRAM: array read (write) = 1.17 (0.39) pJ/bit. PCM: array read (write) = 2.47 (16.82) pJ/bit. Both: row buffer read (write) = 0.93 (1.02) pJ/bit.

TABLE I: Major simulation parameters. Memory timings and energies are adapted from [11].

Benchmark	Description	Server	Cloud
TPC-C/H	TPC-C: Online transaction proc. TPC-H: Decision support, Qry 2 TPC-H: Decision support, Qry 6 TPC-H: Decision support, Qry 17	✓	✓
Apache	Web server workload		✓
H.264	Video processing (compression)		✓

TABLE II: Composition of server- and cloud-type workloads.

VII. Experimental Results

We compare our row buffer locality-aware caching policy (RBLA) against a policy which caches frequently accessed data (FREQ). This frequency-based caching policy is similar in approach to those of prior works in on-chip and hybrid memory caching, such as [7], [25], which aim to improve temporal locality in the cache and reduce off-chip bandwidth consumption by making a caching decision based on the monitored reuse of data. The key idea is that data which is accessed many times in a short interval will likely continue to be accessed and thus would benefit from being cached. Therefore, FREQ caches a row when the number of accesses to the row exceeds the threshold value, `FreqThresh`.

For both FREQ and RBLA, we show results for the best static threshold value (the best averaged across all benchmarks as profiled on a single-core system) and results with the dynamic threshold adjustment algorithm applied (FREQ-Dyn and RBLA-Dyn, substituting `MissThresh` in Algorithm 1 with `FreqThresh` for FREQ-Dyn).

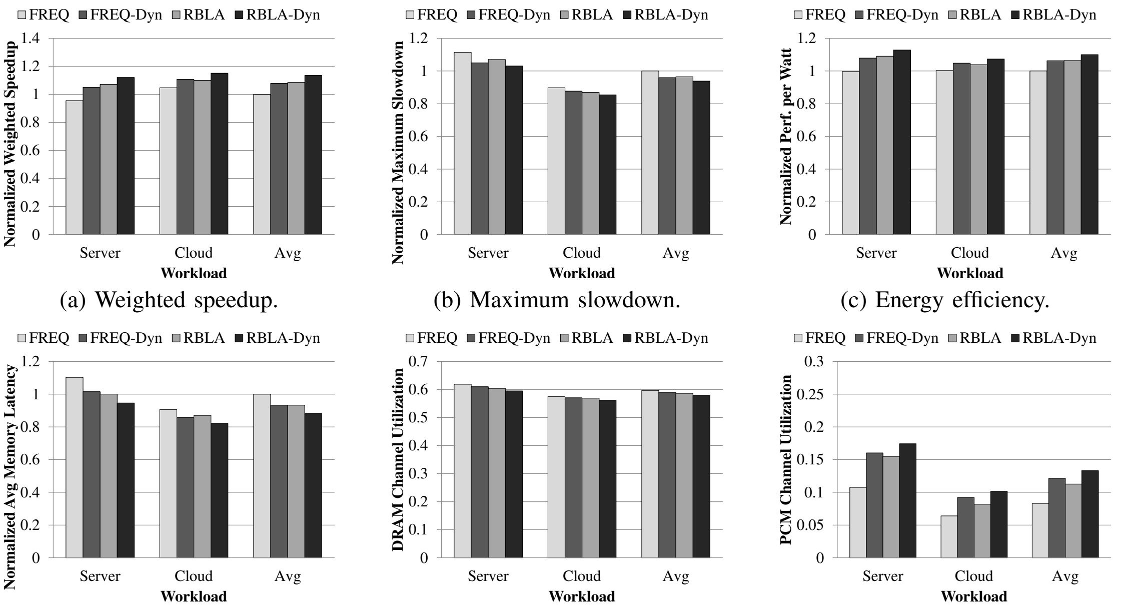
A. Performance

As shown in Figure 4(a), RBLA-Dyn provides the highest performance (14% improvement over FREQ) among all evaluated caching techniques. RBLA places data with low row buffer locality in DRAM where it can be accessed at the lower DRAM array access latency, while keeping data with high row buffer locality in PCM where it can be accessed at a low row buffer hit latency. This can be observed in Figure 5 that shows the total memory accesses and their types.[6] The policies that are row buffer locality-aware (RBLA, RBLA-Dyn) lead to more PCM row buffer hit accesses than the policies that are

[4] This is 0.11% of the total L2 cache size assumed in the evaluation. A 44-bit address space is assumed.

[5] We evaluated the sensitivity to PCM row buffer size and found that the trends of our findings remain the same.

[6] As we simulate for a fixed amount of time (cycles) the policies that perform better lead to more memory accesses.



(a) Weighted speedup.

(b) Maximum slowdown.

(c) Energy efficiency.

(d) Avg. Memory Latency.

(e) DRAM Channel Utilization.

(f) PCM Channel Utilization.

Fig. 4: Various metrics comparing the caching techniques: FREQ, FREQ-Dyn, RBLA, and RBLA-Dyn.

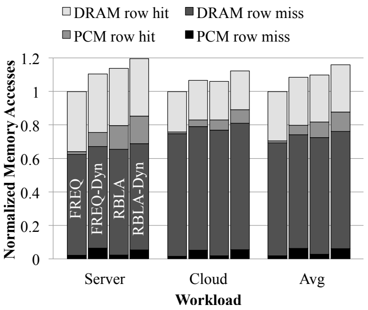


Fig. 5: Memory access breakdown (Normalized in each workload category).

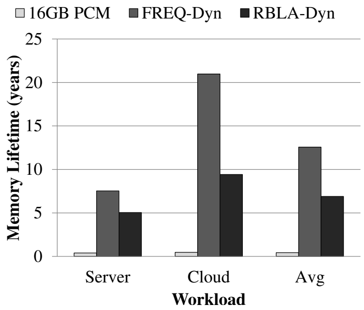


Fig. 6: Memory lifetime for all-PCM, FREQ-Dyn, and RBLA-Dyn memory systems.

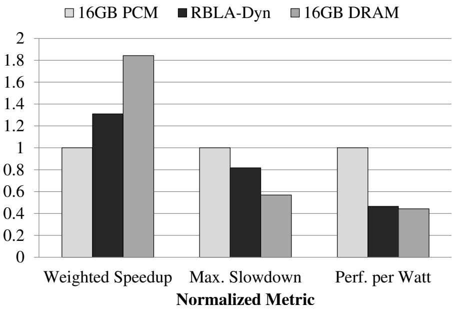


Fig. 7: Performance, fairness, and energy efficiency for all-PCM, RBLA-Dyn, and all-DRAM memory systems.

not (FREQ, FREQ-Dyn). The fraction of memory accesses that hit in the PCM row buffer is 6.6 times more with RBLA-Dyn than with FREQ. This leads to reduced average memory access latency (by 12% from FREQ to RBLA-Dyn as shown in Figure 4(d)), resulting in improved performance.

In addition, we find that the row buffer locality-aware techniques contribute to balancing the memory request load between DRAM and PCM, alleviating contention at the DRAM cache, which serves the majority of memory requests. Figures 4(e) and 4(f) show increased PCM channel utilization and decreased DRAM channel utilization for RBLA(-Dyn) compared to FREQ(-Dyn).

We observe that with the dynamic threshold adaptation, RBLA-Dyn achieves a 5% performance improvement over RBLA with the static threshold (which alone achieves a 9% performance improvement over FREQ). This is because with dynamic threshold adaptation, the caching threshold is adjusted to suit the demands of different workloads at runtime, as opposed to assuming one fixed threshold for all workloads.

B. Thread Fairness

RBLA-Dyn provides the highest thread fairness (6% improvement over FREQ) out of all the evaluated policies as shown in Figure 4(b) (lower is better). As mentioned in Section VII-A, the row buffer locality-aware caching techniques reduce contention on the DRAM cache bandwidth. Furthermore, RBLA-Dyn throttles back on unbeneficial data migrations, reducing the amount of memory bandwidth consumed due to such migrations. These qualities of RBLA-Dyn, combined with the reduced average memory access latency, reduce contention for memory bandwidth among co-running threads, providing increased thread fairness.[7]

C. Energy Efficiency

Figure 4(c) shows that RBLA-Dyn achieves the highest memory energy efficiency (10% improvement over FREQ) in terms of performance per Watt, compared to other techniques. This is because RBLA-Dyn places data which frequently misses in the row buffer in DRAM, thereby ensuring that the energy cost of row buffer miss accesses are lower than it would be if such data was placed in PCM. Additionally, RBLA-Dyn reduces the amount of unbeneficial data migrations, reducing energy consumption caused by such migrations.

D. Lifetime

Figure 6 shows the average memory lifetime in years across all workloads for three systems with 16 GB of PCM: an all-PCM[8] system, and systems with a 1 GB DRAM cache em-

[7] RBLA-Dyn also achieves an *average* slowdown of 3.99, compared to 4.17 for both RBLA and FREQ-Dyn, and 4.45 for FREQ.

[8] Equipped with 2 memory controllers, 1 rank each with 8 banks, for an equal-channel/rank/bank-bandwidth comparison.

ploying either the FREQ-Dyn or RBLA-Dyn caching policies. We assume a conservative cell lifetime of 10^6 writes and a wear-leveling scheme which evenly distributes writes across the entire PCM.

Results show that while employing PCM alone provides less than a year of lifetime, both of the configurations employing DRAM provide close to seven or more years of lifetime, averaging across all workloads.[9] This increase in lifetime is due to the coalescing of write data in the DRAM cache.

The higher lifetime of FREQ-Dyn compared to RBLA-Dyn is due to two reasons. First, FREQ-Dyn executes instructions at a slower rate than RBLA-Dyn (see Figure 4(a)), and therefore observes fewer writes from the processor than RBLA-Dyn in the same amount of time. Second, FREQ-Dyn performs more data migrations to DRAM than RBLA-Dyn (RBLA-Dyn does not migrate frequently accessed data to DRAM unless the data is responsible for frequent row buffer misses), increasing its ability to serve writes in the DRAM cache.

Though there are more row buffer hit accesses to PCM in RBLA-Dyn than there are in FREQ-Dyn, RBLA-Dyn has a higher total number of PCM accesses (balancing the memory request load between DRAM and PCM), which incurs increased wear in PCM.

E. Comparison to All-PCM and All-DRAM Systems

Figure 7 compares the performance, fairness, and energy efficiency of RBLA-Dyn to all-PCM[9] and all-DRAM[9] main memory systems. For the all-DRAM system, we model a 16 GB memory capacity on par with the all-PCM system for an equal-capacity comparison.

The difference in row buffer miss latency causes the all-DRAM system to outperform the all-PCM system and provide higher fairness. RBLA-Dyn efficiently bridges this gap in performance and fairness using a small (1 GB) DRAM cache by exploiting the strengths of both DRAM and PCM. Specifically, RBLA-Dyn achieves within 29% of the weighted speedup of an equal-capacity all-DRAM system. Compared to the all-PCM main memory, RBLA-Dyn improves weighted speedup by 31% and reduces maximum slowdown by 18%. RBLA-Dyn also achieves 5% higher energy efficiency than the all-DRAM system due to the lower static power consumption of the PCM device.[10]

VIII. Related Work

To our knowledge, this is the first work to observe that row buffer hit latencies are similar in different memory technologies, and uses this observation to devise a caching policy that improves the performance and energy efficiency of a hybrid memory system. No previous work, as far as we know, considered row buffer locality as a key metric for deciding what data to cache and what not to cache.

Caching Based on Data Access Frequency: Jiang et al. [7] proposed caching only the data that experiences a high number of accesses in an on-chip DRAM cache (in 4–8 KB block sizes), to reduce off-chip memory bandwidth consumption. Johnson and Hwu [8] used a counter-based mechanism to track 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. We propose taking advantage of row buffer locality in memory banks when employing off-chip DRAM and PCM. We exploit the fact that accesses to DRAM and PCM have similar average latencies for rows that have high row buffer locality.

Ramos et al. [25] adapted a buffer cache replacement algorithm to rank pages based on their frequency and recency of accesses, and placed the highly-ranking pages in DRAM, in a DRAM-PCM hybrid memory system. Our work is orthogonal because the page-ranking algorithm can be adapted to rank pages based on their frequency and recency of row buffer misses (not counting accesses that are row buffer hits), for which we expect improved performance.

Caching Based on Locality of Data Access: Gonzalez et al. [6] proposed placing data in one of two last-level caches depending on whether it exhibits spatial or temporal locality. They also proposed bypassing the cache when accessing large data structures with large strides (e.g., big matrices) to prevent cache thrashing. Rivers and Davidson [27] proposed separating out data without temporal locality from data with, and placing it in a special buffer to prevent the pollution of the L1 cache. These works are primarily concerned with on-chip L1/L2 caches that have access latencies on the order of a few to tens of processor clock cycles, where off-chip memory bank row buffer locality is less applicable.

There have been many works in on-chip caching to improve cache utilization (e.g., a recent one uses an evicted address filter to predict cache block reuse [29]), but none of these consider row buffer locality of cache misses.

Hybrid Memory Systems: Qureshi et al. [24] proposed increasing the size of main memory by adopting PCM, and using DRAM as a conventional cache to PCM. The reduction in page faults due to the increase in main memory size brings performance and energy improvements to the system. We propose a new, effective DRAM caching policy to PCM, and study performance effects without page faults present.

Dhiman et al. [5] proposed a hybrid main memory system that exposes DRAM and PCM addressability to the software (OS). If the number of writes to a particular PCM page exceeds a certain threshold, the contents of the page are copied to another page (either in DRAM or PCM), thus facilitating PCM wear-leveling. Mogul et al. [18] suggested the OS exploit metadata information available to it to make data placement decisions between DRAM and non-volatile memory. Similar to [5], their data placement criteria are centered around the write frequency to data. Our proposal is complementary and row locality information, if exposed, can be used by the OS in placing pages to DRAM or PCM.

Bivens et al. [2] examined the various design concerns of a heterogeneous memory system such as memory latency, bandwidth, and endurance requirements of employing storage class memory (PCM, MRAM, Flash, etc.). Their hybrid memory organization is similar to ours and that in [24], in that DRAM is used as a cache to a slower memory medium, transparently to software. Phadke et al. [23] proposed profiling the memory access patterns of individual applications in a multi-core system, and placing their working sets in the particular type of DRAM that best suits the application's memory demands.

[9] The minimum lifetimes observed for a workload are 3.23 and 2.89 years for FREQ-Dyn and RBLA-Dyn respectively.

[10] PCM's smaller row buffer size than DRAM leads to lower static energy consumption and higher memory energy efficiency.

In contrast, our mechanism dynamically makes fine-grained data placement decisions at a row granularity, depending on the row buffer locality characteristics.

Exploiting Row Buffer Locality: Many previous works exploit row buffer locality to improve memory system performance, but none (to our knowledge) develop a cache data placement policy that considers the row buffer locality of the block to be cached.

Lee et al. [11] proposed using multiple short row buffers in PCM devices, much like an internal device cache, to increase row buffer hit rate. Sudan et al. [32] presented a scheme of identifying frequently referenced sub-rows of data and migrating them to reserved rows. By co-locating these frequently accessed sub-rows, this scheme aims to increase the row buffer hit rate of memory accesses, and improve performance and energy consumption. DRAM-aware last-level cache writeback schemes [31], [13] have been proposed which speculatively issue writeback requests that hit in the row buffer. Our proposal is complementary to these works and can be applied together because it targets a different problem.

Row buffer locality is also commonly exploited in memory scheduling algorithms. The First-Ready First Come First Serve algorithm (FR-FCFS, assumed in our evaluations) [28], [37] prioritizes memory requests that hit in the row buffer, improving the latency, throughput, and energy cost of serving memory requests. Many other memory scheduling algorithms [21], [20], [9], [10], [14] build upon this "row-hit first" principle.

Finally, Muralidhara et al. [19] used a thread's row buffer locality as a metric to decide which channel the thread's pages should be allocated to in a multi-channel memory system. Their goal was to reduce memory interference between threads, and as such their technique is complementary to ours.

IX. Conclusion

This paper observes that row buffer access latency (and energy) in DRAM and PCM are similar, while PCM array access latency (and energy) is much higher than DRAM array access latency (and energy). Therefore, in a hybrid memory system where DRAM is used as a cache to PCM, it makes sense to place in DRAM data that would cause frequent row buffer misses as such data, if placed in PCM, would incur the high PCM array access latency. We develop a caching policy that achieves this effect by keeping track of rows that have high row buffer miss counts (i.e., low row buffer locality, but high reuse) and places only such rows in DRAM. Our final policy dynamically determines the threshold used to decide whether a row has low locality based on cost-benefit analysis. Evaluations show that the proposed row buffer locality aware caching policy provides better performance, fairness, and energy-efficiency compared to caching policies that only consider access frequency or recency. Our mechanisms are applicable to other hybrid memory systems consisting of different technologies, a direction we intend to explore in the future.

Acknowledgment

We acknowledge the support of AMD, HP, Intel, Oracle, and Samsung. This research was partially supported by grants from NSF (CCF-0953246, CCF-1212962), GSRC, Intel URO, and ISTC on Cloud Computing. HanBin Yoon is partially supported by the Samsung Scholarship. Rachata Ausavarungnirun is supported by the Royal Thai Government Scholarship.

References

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

[2] A. Bivens et al. Architectural design for next generation heterogeneous memory systems. Intl. Memory Workshop 2010.

[3] J. Coburn et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. ASPLOS 2011.

[4] J. Condit et al. Better I/O through byte-addressable, persistent memory. SOSP 2009.

[5] G. Dhiman et al. PDRAM: a hybrid PRAM and DRAM main memory system. DAC 2009.

[6] A. González et al. A data cache with multiple caching strategies tuned to different types of locality. ICS 1995.

[7] X. Jiang et al. CHOP: adaptive filter-based dram caching for CMP server platforms. HPCA 2010.

[8] T. L. Johnson and W.-m. Hwu. Run-time adaptive cache hierarchy management via reference analysis. ISCA 1997.

[9] Y. Kim et al. ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers. HPCA 2010.

[10] Y. Kim et al. Thread cluster memory scheduling: exploiting differences in memory access behavior. MICRO 2010.

[11] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. ISCA 2009.

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

[13] C. J. Lee et al. DRAM-aware last-level cache writeback: reducing write-caused interference in memory systems. Tech. report, UT-Austin, 2010.

[14] C. J. Lee et al. Prefetch-aware DRAM controllers. MICRO 2008.

[15] G. H. Loh and M. D. Hill. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. MICRO 2011.

[16] J. A. Mandelman et al. Challenges and future directions for the scaling of dynamic random-access memory (dram). *IBM J. Res. Dev.*, 46, 2002.

[17] J. Meza et al. Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. *Comp. Arch. Letters 2012*.

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

[19] S. P. Muralidhara et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning. MICRO 2011.

[20] O. Mutlu et al. Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems. ISCA 2008.

[21] O. Mutlu et al. Stall-time fair memory access scheduling for chip multiprocessors. MICRO 2007.

[22] H. Patil et al. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. MICRO 2004.

[23] S. Phadke et al. MLP aware heterogeneous memory system. DATE 2011.

[24] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. ISCA 2009.

[25] L. E. Ramos et al. Page placement in hybrid memory systems. ICS 2011.

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

[27] J. Rivers and E. Davidson. Reducing conflicts in direct-mapped caches with a temporality-based design. ICPP 1996.

[28] S. Rixner et al. Memory access scheduling. ISCA 2000.

[29] V. Seshadri et al. The evicted-address filter: A unified mechanism to address both cache pollution and thrashing. PACT 2012.

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

[31] J. Stuecheli et al. The virtual write queue: coordinating DRAM and last-level cache policies. ISCA 2010.

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

[33] The International Technology Roadmap for Semiconductors. Process integration, devices, and structures, 2010.

[34] H. Wong et al. Phase change memory. *Proc. of the IEEE*, 2010.

[35] H. Yoon et al. DynRBLA: A high-performance and energy-efficient row buffer locality-aware caching policy for hybrid memories. Technical report, Carnegie Mellon University, 2011.

[36] W. Zhang et al. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. PACT 2009.

[37] W. K. Zuravleff et al. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. U.S. Patent Number 5,630,096, 1997.