To Update or Not To Update?: Bandwidth-Efficient Intelligent Replacement Policies for DRAM Caches

Vinson Young and Moinuddin K. Qureshi
Georgia Institute of Technology
{vyoung,moin}@gatech.edu

***Abstract*—This paper investigates intelligent replacement policies for improving the hit-rate of gigascale DRAM caches. Cache replacement policies are commonly used to improve the hit-rate of on-chip caches. The most effective replacement policies often require the cache to track and update per-line reuse state to inform their decision. A fundamental challenge on DRAM caches, however, is that stateful policies would require significant bandwidth to maintain per-line DRAM cache state. As such, DRAM cache replacement policies have primarily been stateless policies, such as always-install or probabilistic bypass. Unfortunately, we find that stateless policies are often too coarse-grain and become ineffective at the size and associativity of DRAM caches. Ideally, we want a replacement policy that can obtain the hit-rate benefits of stateful replacement policies, but keep the bandwidth-efficiency of stateless policies.**

We perform our study on a DRAM cache design similar to the one in Knights Landing, and find that tracking per-line reuse state can enable an effective replacement policy that can mitigate the common thrashing patterns seen in gigascale caches. We propose a stateful replacement/bypass policy called *RRIP Age-On-Bypass (RRIP-AOB)*, that tracks reuse state for high-reuse lines, protects such lines by bypassing other lines, and *Ages the state On cache Bypass*. Unfortunately, such a stateful technique requires significant bandwidth to update state. To this end, we propose *Efficient Tracking of Reuse (ETR)*. ETR makes state tracking efficient by accurately tracking the state of only one line from a region, and using the state of that line to guide the replacement decisions for other lines in that region. ETR reduces the bandwidth for tracking replacement state by 70%, and makes stateful policies practical for DRAM caches. Our evaluations with a 2GB DRAM cache, show that our RRIP-AOB and ETR techniques provide 18% speedup while needing <1KB SRAM.

I. Introduction

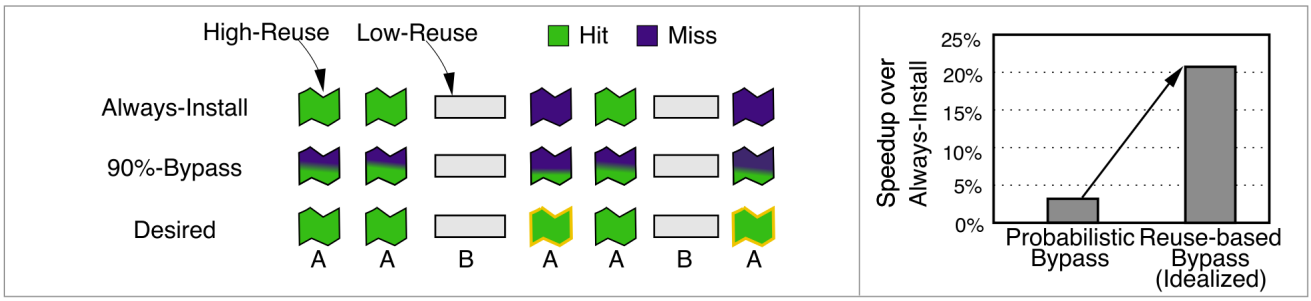
DRAM caches are important for enabling effective heterogeneous memory systems that can transparently provide the bandwidth of high bandwidth memories [1], and the capacity of high capacity memories [2], [3]. Designs for DRAM cache organize the tag-store such that the tags can be kept in DRAM (to reduce storage overheads) and yet the tags can also be obtained with low latency and low bandwidth overheads [4], [5]. For example, Intel's Knights Landing product organizes its DRAM cache as a direct-mapped cache with tags stored alongside each data-line, so that one access can retrieve both tag and data. This direct-mapped design is effective for enabling low latency and bandwidth-efficient tag access [5]; however, such a direct-mapped design can have significant conflict misses. One could consider increasing associativity to improve hit-rate, but, increasing associativity also substantially increases bandwidth usage and degrades performance for many workloads. Fortunately, cache bypassing [6], [7], [8] offers a way to both improve hit-rate and decrease bandwidth consumption, while still maintaining a direct-mapped organization. We investigate the extent to which an intelligent bypass policy can reduce conflict misses for DRAM caches. We perform our evaluations on a direct-mapped DRAM cache similar to the one in KNL [4], [5].

We would like to use the most effective replacement policies to improve DRAM cache hit-rate. However, intelligent replacement policies [8], [9], [10], [11] often require the cache to track per-line state that needs to be updated on cache events. On a DRAM cache, managing this per-line state is difficult as tracking even 2 bits of state per line would require multi-megabyte storage. As such, DRAM cache designs would need to keep this state in the DRAM array, and spend offchip bandwidth to update state. Prior replacement policies proposed for DRAM caches have avoided this per-line state with stateless policies [4], [5], [6]. The DRAM cache in KNL [4], [5], for example, employs an *Always-Install* policy. Along the same lines, Chou et. al [6] propose a policy that bypasses the cache with 90% probability (we call this policy *90%-Bypass*). However, such stateless policies often fail to capture the reuse patterns commonly seen in large caches. We show how such policies are often inadequate with an example.

Let us consider replacement policies for a common access pattern where the workload has repeated accesses to high-reuse data (labeled *A*) interspersed with accesses to low-reuse data that is not re-referenced while it is in the cache (labeled *B*), as shown in Figure 1(a). For the baseline *Always-Install* policy, accesses to A will install A and enable subsequent accesses to A to hit; however, accesses to low-reuse B will evict A and cause the subsequent access to A to miss. In this case, always-installing lines allows low-reuse B to evict high-reuse A, and this results in degraded hit-rate and wasted install bandwidth. For a *90%-Bypass* policy, references to A will install some A lines and marginally improve hit-rate, and references to B will install a few lines and marginally degrade hit-rate. In this case, 90%-Bypass offers some working-set protection; however, it is indiscriminate in deciding which lines to protect and may not enable high hit-rate. Figure 1(b) shows that such a probabilistic bypass policy has poor speedup potential of 3%. Ideally, we desire a bypass policy that can remember and protect individual lines that have high reuse (i.e., A), and bypass other lines (i.e., B). Figure 1(b) shows that if we are able to formulate such a *reuse-based bypass* policy while avoiding the bandwidth cost for state update, we could enable up to 20% speedup.

Our approach to improving DRAM cache performance is to (1) design a *reuse-based bypass* policy to improve DRAM cache hit-rate, and to (2) reduce the bandwidth cost of state update to further improve performance.

In this paper, we use *Re-Reference Interval Prediction (RRIP)* [9] as a representative example of a replacement policy that is designed to exploit reuse [8], [9], [10], [12], [13]. RRIP



(a) Replacement Policies (b) Speedup Potential

Fig. 1. (a) Always-Install, 90%-Bypass, and Desired replacement policies under mixed high-reuse low-reuse access pattern. (b) Potential for speedup: Probabilistic Bypass [6], and Ideal Reuse-based Bypass with no state update cost.

requires that each line is equipped with metadata bits (two-bit counter called RRPV) to track reuse. RRPV is set to 0 on a hit, and the victim line is identified as a line that has an RRPV of 3. And, if no lines have an RRPV of 3, all counters in the set are incremented and victim-selection is repeated. While RRIP is effective for set-associative caches, it becomes ill-defined for direct-mapped caches, as such a cache would have only one counter in the set. Following the algorithm for selecting a victim will always cause the resident line to get evicted, even if the line had an RRPV of 0. Similarly, bypassing the incoming line if the resident line has an RRPV=0 will mean that such lines will never get evicted from the cache.

To enable reuse-based replacement policies for direct-mapped DRAM caches, we propose a bypass version of RRIP, which we call *RRIP-AOB*. The key mechanism in RRIP-AOB, is to *Age the counters On cache Bypass*. For example, if a good victim cannot be found (no RRPV are 3), we bypass the incoming line and age the reuse counter (increment RRPV). After several bypass+age events, a resident line that is no longer useful will have its RRPV reach 3 and become a candidate for eviction. This enables the cache to protect lines that have had reuse via bypassing, but also provides a path to eventually victimize cold lines. Our insight makes RRIP (and other reuse-based policies) applicable to DRAM caches.

Another practical obstacle in implementing reuse-based policies for DRAM caches is the high state update cost of maintaining replacement state in DRAM. A straight-forward way of implementing RRIP-AOB in DRAM cache is to extend the tag-entry of the line to incorporate the bits for tracking the replacement state of the line. However, it incurs bandwidth overhead for performing update of the replacement state: resetting the RRPV counter on a hit (*promotion*), and incrementing the RRPV counter on a bypassing miss (*demotion*). Note that these accesses for updating the replacement state are not present in the baseline and for designs that do bypassing without tracking per-line state. If we can completely remove with state update cost with Ideal RRIP-AOB, we can achieve up to 20% speedup. To reduce the state update cost of maintaining per-line counters in DRAM, we propose *Efficient Tracking of Reuse (ETR)*.

ETR reduces the bandwidth consumed in performing updates of the replacement state by doing the updates for only a subset of the lines and using their replacement state to infer the replacement state of the other lines. ETR is based on two key properties that we observe in DRAM caches: *Coresidency* and *Eviction-Locality*. Coresidency indicates that at any given time if a line is present, then several other line belonging to that 4KB region are also present in the cache. Eviction-Locality indicates that when a line gets evicted from the cache, the replacement-state of the other coresident lines belonging to that region tend to have similar replacement-state as the line being evicted. We show strong levels of coresidency and eviction-locality with RRIP-AOB. ETR exploits the properties of coresidency and eviction-locality to reduce the updates for tracking the replacement state. Rather than updating the replacement-state for all lines in the cache, ETR simply updates the replacement-state for one of the coresident lines of the region, and uses the state of this line to guide the replacement decisions of other coresident lines of the region. ETR reduces the bandwidth overhead of state updates by 70% and enables RRIP-AOB to achieve 18% speedup, nearing Ideal RRIP-AOB performance. These benefits are obtained with a storage overhead of less than 1KB SRAM.

> **Note:** A cache implementing ETR still fundamentally employs line-based replacement – it simply opportunistically exploits spatial locality when it exists to reduce state update costs. We compare with alternative page-based [14] designs in Section VII-C, and grouped-metadata [15] approaches in Section VII-B.

Overall our paper makes the following contributions:

Contribution-1: To our knowledge, this is the first paper to investigate intelligent replacement / bypass policies for direct-mapped DRAM caches. We propose a bypass version of RRIP *(RRIP-AOB)* suitable for caches with limited associativity. However, we find an effective replacement policy for DRAM caches must optimize not only hit-rate but also state update cost. We introduce two properties, *coresidency* and *eviction-locality*, that can be exploited to reduce state update cost for implementing intelligent replacement.

Contribution-2: We propose *Efficient Tracking of Reuse (ETR)*, a design that performs updates for only a subset of lines and uses their state to guide the replacement decisions of other lines. ETR reduces bandwidth overhead of updates by 70%, improves speedup to 18%, and requires only 512-bytes.

Contribution-3: We show that RRIP-AOB and ETR are general techniques also applicable to set-associative implementations of DRAM caches.

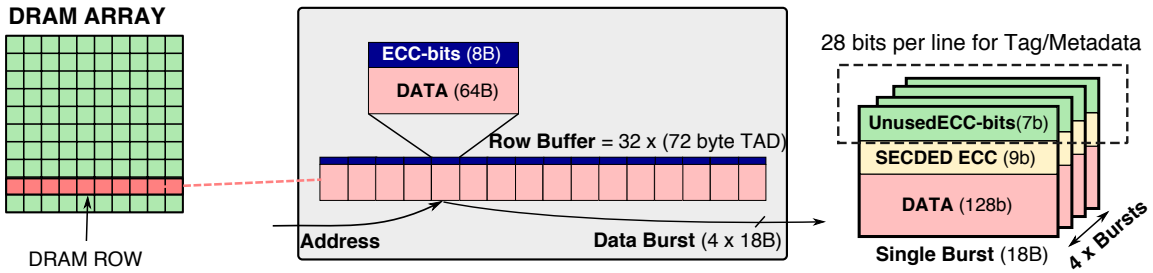


Fig. 2. Organization of the DRAM cache in KNL. DRAM cache is organized at a linesize of 64 bytes, is direct-mapped, and tags are kept with the data-line. On an access, the DRAM cache transfers 72 bytes using four bursts on an 18-byte bus (16-bytes for data + 2-bytes for ECC). We need only 9-bits for SECDED on 16-bytes of data, which leaves 7 unused ECC bits in each burst that can be used to store metadata (KNL uses these 28 unused ECC bits to store tags).

II. Background and Motivation

We present the organization of our DRAM cache and discuss the storage and bandwidth constraints that make it challenging to apply intelligent replacement policies.

A. Organization of a DRAM Cache (KNL)

As the tag storage required for gigascale DRAM caches is large, DRAM cache designs often store tags in DRAM and intelligently organize their structure to enable efficient tag-access. The baseline we use for this study is the direct-mapped, tags-in-ECC organization used in *Intel's Knights Landing (KNL)* design [4], [5]. Figure 2 shows the organization of the DRAM cache in KNL. The DRAM cache places each tag information in the unused bits in the ECC space and streams out the data and tag (contained in ECC) on each access. The tag information is used to determine cache hit or miss. On a tag match, the data is available to service the request immediately, without any additional latency. Thus, co-locating the tag and data allows the DRAM cache access to be serviced in just one DRAM request, which makes the cache hit operation both low-latency and bandwidth-efficient [5]. Our goal is to increase the hit-rate of such DRAM caches. In fact, the DRAM cache only uses about 8-10 bits from the unused 28 ECC bits, so we have 18-20 bits per line available for managing the DRAM cache intelligently. We leverage these bits to build intelligent replacement policies.

B. Replacement / Bypass Policies for 1-Way

Typically, cache replacement policies are discussed in the context of a set associative cache, as the set contains multiple lines and there is a choice of the line to evict. For a direct-mapped cache, the set contains only one line, so if we want to install, there is exactly one place the line can go, and we do not have a choice in selecting the victim. However, we could choose to bypass the line, so the binary choice for a direct-mapped cache becomes, whether to evict the resident line or to bypass the incoming line. We can improve the hit-rate by making this binary decision intelligently. We explain different replacement strategies for a direct-mapped cache.

Probabilistic Replacement: The simplest policy is to bypass the incoming line with a certain probability. For example, Bandwidth-Aware Bypass (BAB) [6], [8] bypasses the incoming line with 90% probability to reduce install bandwidth, as long as hit-rate remains unaffected. Figure 5 shows that such global bypassing policies are coarse-grain and miss out on bypassing opportunities that exploit per-line information.

Recency-Based Replacement: LRU [16] installs incoming lines with the highest priority, based on the heuristic that recently-used lines are more likely to be re-used. On a direct-mapped cache, LRU degenerates into an *Always-Install* design, as the incoming line is the most recent. Enhancements of LRU, such as DIP [17], degenerate to probabilistic bypass.

Reuse-Based Replacement: Replacement policies that exploit reuse (also called re-reference or frequency) are resilient to thrashing and scans [9], [18], [19]. Such policies can protect the direct-mapped DRAM cache from thrashing when multiple pages are mapped to the same set of the DRAM cache. We discuss *Re-Reference Interval Prediction (RRIP)* [9] policy.

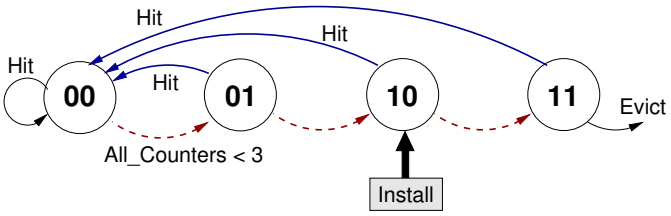


Fig. 3. Re-Reference Interval Prediction (RRIP).

Re-Reference Interval Predictor [9] is a thrash-and-scan-resistant replacement policy often used in last-level caches. As shown in Figure 3, each line is equipped with a 2-bit counter to track the *Re-Reference Interval Prediction Value (RRPV)*. On a hit to the line, the RRPV is *Promoted* to 0. On a miss, the victim is found by searching from way 0 and finding the first line in the set with RRPV of 3. If no such line is found, the RRPV of all lines in the set is *Demoted* (i.e., incremented) and the search is repeated. Lines are installed in RRPV=2 to protect the lines that were re-used.

Challenge in Using RRIP for Direct-Mapped Cache: Just like other replacement policies based on reuse-information, RRIP operates by comparing the counter values of multiple candidates in the set. It becomes ill-defined for a direct-mapped cache, where there is only one counter, which means the resident line will always get evicted regardless of the past behavior. Thus, for a direct-mapped cache RRIP degenerates into always-install (or always-bypass if the incoming line is bypassed unless the RRPV of the resident line equals 3). We propose extensions that make reuse-based policies viable for direct-mapped and two-way caches, and implementations that reduce the cost of tracking the RRPV state for gigascale DRAM caches. We discuss our solution after methodology.

III. Methodology

A. Framework and Configuration

We use USIMM [20], an x86 simulator with detailed memory system model. We extend USIMM to include a DRAM cache. Table I shows the configuration used in our study. We model a configuration similar to a Intel Knights Landing (KNL) Sub-NUMA Cluster (one-eighth size). We assume a four-level cache hierarchy (L1, L2, L3 being on-chip SRAM caches and L4 being off-chip DRAM cache). All caches use 64B line size. We model a virtual memory system to perform virtual to physical address translations. The L4 is a 2GB DRAM cache [5], [21], which is direct-mapped and places tags with data in the unused ECC bits. The parameters of our DRAM cache is based on HBM technology [1]. The main memory is based on non-volatile memory (PCM and 3D-XPoint [3], [22], [23], [24], [25], [26], [27]): the read latency is 4X that of DRAM [2], and write bandwidth is worse than read bandwidth. We perform evaluations with DRAM-based memory in Section VI-C.

TABLE I
System Config (KNL ⅛ Sub-NUMA Cluster)

Processors	8 cores; 3.0GHz, 2-wide OoO
Last-Level Cache	8MB, 16-way
DRAM Cache	
Capacity	2GB
Bus Frequency	500MHz (DDR 1GHz)
Configuration	4 channel, 128-bit bus
Aggregate Bandwidth	64 GB/s
tCAS-tRCD-tRP-tRAS	13-13-13-30 ns
Main Memory (PCM)	
Capacity	64GB
Bus Frequency	1000MHz (DDR 2GHz)
Configuration	1 channel, 64-bit bus
Aggregate Bandwidth	16 GB/s
tCAS-tRCD-tRP	13-128-8 ns
tRAS-tWR	143-160 ns

B. Workloads

We use a representative slice of 2-billion instructions selected by PinPoints [28], from benchmarks suites that include SPEC 2006 [29], GAP [30], and HPC. For SPEC, we pick a sample of high intensity workloads that have at least two miss per thousand instructions (MPKI). The evaluations execute benchmarks in rate mode, where all eight cores execute the same benchmark. In addition to rate-mode workloads, we also evaluate 4 mixed workloads, which are created by randomly choosing 8 of the 15 SPEC workloads that have at least two MPKI. Table II shows L3 miss rates, and memory footprints for the 8-core rate-mode workloads in our study.

We perform timing simulation until each benchmark in a workload executes at least 2 billion instructions. We use weighted speedup to measure aggregate performance of the workload normalized to the baseline and report geometric mean for the average speedup across all the 21 workloads (11 SPEC, 4 SPEC-mix, 5 GAP, 1 HPC).

TABLE II
Workload Characteristics

Suite	Workload	L3 MPKI	Footprint
SPEC	soplex	35.3	1.8 GB
	leslie	22.1	623 MB
	libq	30.1	256 MB
	gcc	108.5	1.5 GB
	omnet	29.1	1.2 GB
	wrf	10.4	1.1 GB
	zeus	7.0	1.6 GB
	xalanc	7.4	1.5 GB
	mcf	101.1	13 GB
	milc	31.2	4.5 GB
	sphinx	15.0	146 MB
GAP	cc twitter	116.8	9.3 GB
	bc twitter	101.2	13.5 GB
	pr twitter	126.6	15.3 GB
	pr web	24.8	15.1 GB
	cc web	11.4	9.3 GB
HPC	nekbone	13.71	44 MB

IV. RRIP: Age-On-Bypass

If we want to use RRIP on direct-mapped DRAM caches, we have to solve two issues: how do we formulate RRIP as a bypassing policy suitable for caches with limited associativity, and how can we mitigate the state update cost of maintaining per-line reuse state in DRAM.

A. RRIP as a Bypassing Policy

We design a version of RRIP for limited-associativity caches, called *RRIP: Age-On-Bypass (RRIP-AOB)*. The key insight in RRIP-AOB is to use the episode of cache bypassing to age / update the RRPV information associated with the line. Figure 4 shows the overview of our design. RRIP-AOB needs to similarly track lines that have reuse, so RRIP-AOB Promotes state (sets RRPV to 0) on hit. RRIP-AOB can protect these reused lines by bypassing when reuse has been seen (bypass when RRPV is 0, 1, or 2). However, reused lines can now stay stuck in high priority state. We now need a mechanism to age older lines so that new lines can eventually be installed. We choose to implement aging by Demoting (increment RRPV) state when an incoming line is bypassed. This allows lines to naturally age to RRPV of 3, and be evicted in favor of an incoming line. Similar to RRIP, RRIP-AOB uses 2 bits per line to track RRPV. A practical design must address where to store the RRPV bits and address the bandwidth needed to track the per-line RRPV.

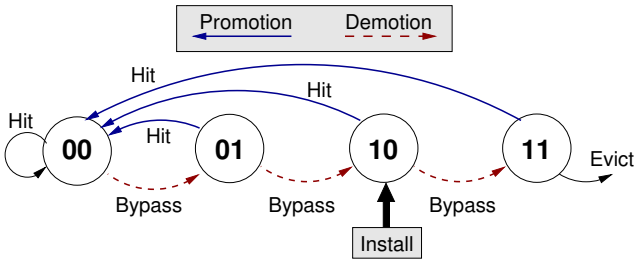


Fig. 4. Overview of RRIP: Age-On-Bypass (RRIP-AOB). The transition from one state to another is accomplished with replacement-state update operation. Such updates may consume significant bandwidth.

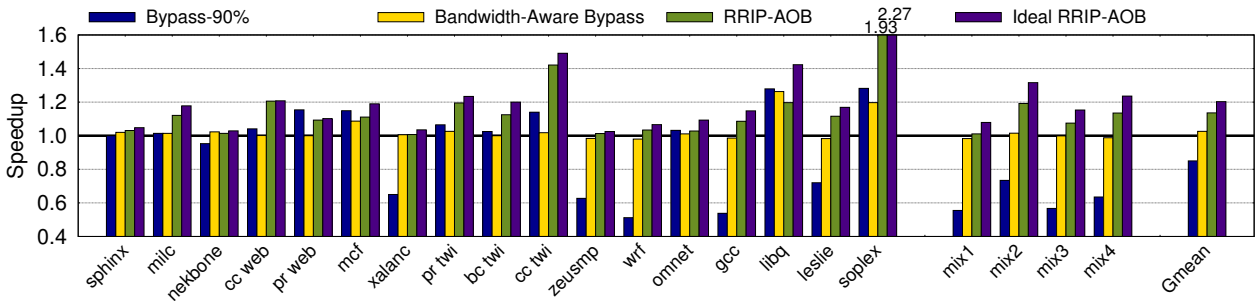


Fig. 5. Speedup from different replacement policies over the baseline always-install direct-mapped DRAM cache. (a) Bypass-90% causes 15% degradation, (b) Bandwidth-Aware Bypass provides 3% speedup, (c) RRIP-AOB that maintains state in DRAM provides 13% speedup, and (d) Ideal RRIP-AOB with no state update cost provides 20% speedup

B. Storing RRPV in DRAM

A straight-forward way of incorporating RRIP into a DRAM cache is to extend the tag-entry of the line to incorporate RRPV bits. We refer to this design as simply *RRIP-AOB*. However, such a design incurs bandwidth overhead to update the replacement state. Note that these accesses for updating the replacement state are not present in the baseline and for designs that do bypassing without tracking per-line state.

Alternatively, we can avoid the bandwidth of replacement updates by storing the replacement state in a dedicated SRAM array. Unfortunately, for our 2GB DRAM cache, maintaining 2 bits of RRPV per line would need 8MB of SRAM, which is impractically large. We call this design *Ideal RRIP-AOB*.

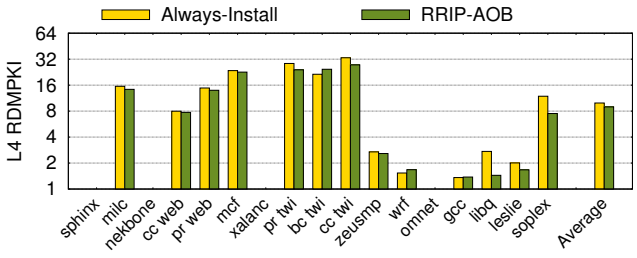


Fig. 6. MPKI of baseline DRAM cache and RRIP-AOB. RRIP-AOB reduces misses by 10%.

C. Benefits from Reuse-Based Replacement

Intelligent replacement policies improve performance by reducing cache misses. Figure 6 shows the Misses Per Thousand Instructions (MPKI) for our baseline DRAM cache and with RRIP-AOB. RRIP-AOB reduces 10% of the misses on average. However, the speedup from RRIP-AOB also depends on bandwidth used in replacement-state updates.

Figure 5 shows the speedup from different bypassing policies implemented on our 2GB DRAM cache. Performance numbers are normalized to the always-install policy. Indiscriminately bypassing 90% of the lines (Bypass-90%) causes a degradation of 15%. The adaptiveness of Bandwidth-Aware-Bypass (BAB) [6], [8] avoids slowdowns; however, the average speedup is only 3%. With RRIP-AOB, the performance benefits is 13%, whereas with Ideal RRIP-AOB the speedup could be 20%. Thus, there is significant room for performance improvement with reuse-based replacement policies. Unfortunately, obtaining this benefit in a practical manner is challenging as maintaining accurate per-line state in DRAM requires significant bandwidth for state updates.

D. Dissecting BW of Replacement-Updates

To highlight the bandwidth differences between Always-Install and RRIP-AOB, we show the bandwidth needed to implement replacement policy for Always-Install and RRIP-AOB. Always-Install simply has install bandwidth, whereas RRIP-AOB additionally needs bandwidth to *promote* and *demote* state. Figure 7 shows the replacement bandwidth of RRIP-AOB, normalized to the replacement bandwidth of Always-Install. Of particular note, *RRIP-AOB has the potential to save 76% of the install bandwidth* (due to bypass), which can improve performance. However, it has overall increased bandwidth consumption due to *promotion* and *demotion*. If we want to obtain most of the benefits of RRIP, we must develop methods to reduce this bandwidth overhead.

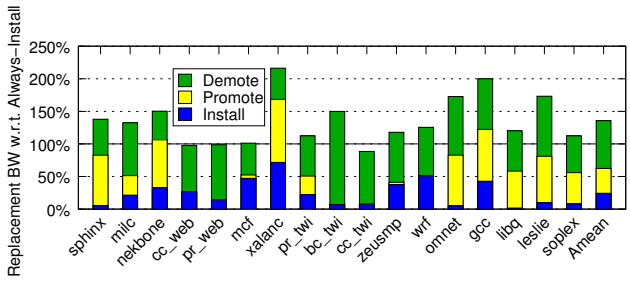


Fig. 7. Replacement bandwidth (Install, Promote, Demote) of RRIP-AOB, normalized to replacement bandwidth (Install) of Always-Install. RRIP-AOB reduces install bandwidth but incurs state update bandwidth.

E. Potential for Improvement

RRIP-AOB with state in DRAM is a practical design as it does not require any SRAM overheads, and can be implemented without any changes to the DRAM cache (the extra bits for RRPV are taken from the unused ECC bits). However, it has two-thirds the speedup compared to the potential benefit of Ideal RRIP-AOB with no state update costs. Ideally, we would like to get speedup similar to Ideal RRIP-AOB, while needing low SRAM cost similar to RRIP-AOB. The goal of the next section is to develop such a solution.

RRIP-AOB simply suffers from high DRAM state update cost. If we can find effective ways to mitigate this bandwidth overhead, we can get most of the benefits at little cost. We develop an insight that if we can do replacement updates in an efficient manner for only a subset of the lines, then we can reduce the bandwidth for replacement updates and still retain most of the benefits.

V. Efficient Tracking of Reuse

Demoting state on every cache bypass incurs significant bandwidth overheads–even if we choose to bypass the line, we still have to spend bandwidth to *demote* the replacement-state. We can avoid state update costs if we have an effective way to infer an RRPV state. Our design reduces the bandwidth consumed in performing updates of the replacement state by doing the updates for only a subset of the lines and using their replacement state to infer the replacement state of the other lines. Our solution is based on two key properties, *Coresidency* and *Eviction-Locality*, which we describe next.

A. Insight: Coresidency and Eviction-Locality

Coresidency indicates that at any given time if a line is present, then several other lines belonging to that region are also present in the cache. Coresidency indicates that there is some spatial locality in the reference stream, even if such spatial locality is not perfect. A 4KB region contains 64 lines each of 64 bytes. Therefore, the maximum number of coresident lines for a region would be 63. Figure 8 shows the level of coresidency for our workloads. In general, the workloads have between 16 to 45 coresident lines. We note that although this is lower than perfect spatial locality, there are still a large number of lines coresident (even 4 coresident lines can amortize 75% state update cost). This shows there is potential for using one line to infer replacement state of many coresident lines.

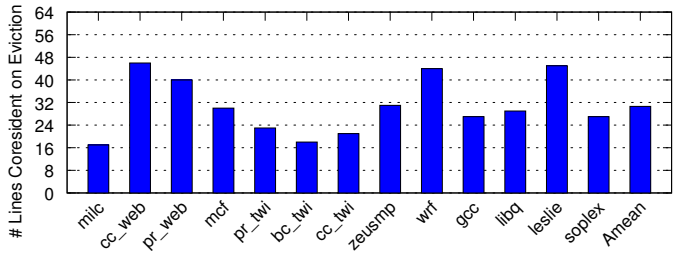


Fig. 8. Coresidency in DRAM caches. Average number of coresident lines in a 4KB region on first line evicted from a region (workloads with L4 MPKI>1).

Eviction-Locality indicates that when a line gets evicted from the cache, then the replacement-state of the other coresident lines belonging to that region tend to have similar replacement-state as the line being evicted. Figure 9 shows the distribution of the RRPV of coresident lines, on an eviction from L4 (on the first line evicted from a region).

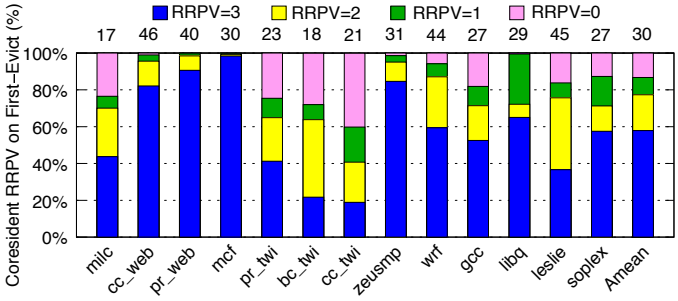


Fig. 9. Distribution of RRPV of coresident lines on first line evicted from a 4KB region, for workloads with L4 MPKI>1. Average number of coresident lines shown above workloads. Eviction of one line indicates other lines in a region are likely to be evicted soon (RRPV≥2).

Combined insight: On eviction, we typically observe 30 or more lines are coresident. In addition, we find the coresident lines often have similar RRPV state (77% have RRPV≥2). Together, this means that if we maintain accurate RRPV for just one of the coresident lines, then we can infer RRPV state for the rest of the coresident lines in the region with reasonable accuracy. Our solution exploits this insight.

B. Insight: Update Only the Representative

We propose *Efficient Tracking of Reuse (ETR)* to reduce bandwidth overheads of doing replacement updates in DRAM. We implement ETR on RRIP-AOB as an example. ETR exploits coresidency and eviction-locality. Instead of updating replacement state for all the lines in a region, ETR updates the state of only one *Representative-Line* among all the coresident lines. The state of the Representative-Line is then used to guide the replacement policy of the coresident lines. The design of ETR consists of three parts: (1) Selecting a Representative-Line in the region (2) Keeping accurate RRPV for *only* the Representative-Line, and (3) Using the representative's RRPV to infer coresident lines' RRPV to make bypass decisions.

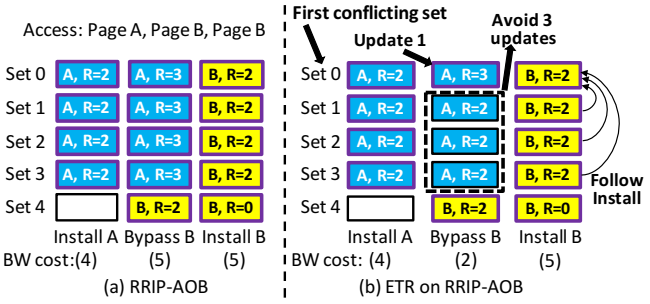


Fig. 10. ETR's representative-update and bypass-decision following enables similar RRIP-AOB install, at reduced update bandwidth (dashed box = benefit)

To implement representative-update, we first need to pick a stable representative line. The first access to a region is generally consistent [31]. If we maintain state for just the first conflicting set in a region, we can maintain good reuse information for the rest of the region without incurring extra bandwidth costs. Figure 10 shows an example of how ETR's RRPV-inference (i.e., representative-update and bypass-decision following) can be used to obtain similar install-policy and hit-rate at reduced update cost.

If we first access 4 lines from region A at time 0, we install region A with RRPV=2. If 5 lines from region B are then accessed, Figure 10(b) shows that we can save bandwidth and *demote* only the state of the first conflicting set (being set 0). On second access to region B, set 0 with its RRPV of 3 will inform us that that region A was not used recently. This means that lines corresponding to region A have low reuse and should be evicted in favor of installing region B. We can then follow the region B install-decision for the rest of the lines. Such a policy will end up installing all of region B and result in an install policy similar to if we had maintained each state individually in Figure 10(a). As such, we can keep similar install policy and save update bandwidth with representative state update and bypass-decision following.

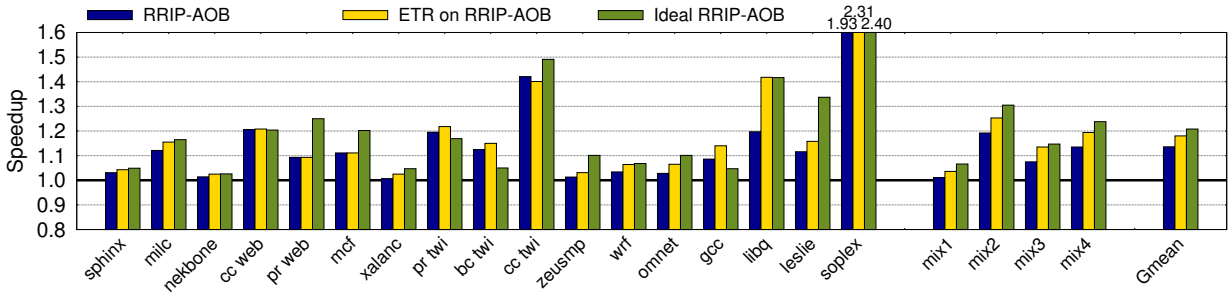


Fig. 11. Performance of RRIP-AOB, ETR on RRIP-AOB, and an Ideal RRIP-AOB with no state update costs. Coordinating bypass decisions with ETR reduces state update needs, and enables RRIP-AOB to obtain 18% speedup.

Structures for ETR: To implement representative state update and bypass-decision following, ETR maintains a *Recent-Bypass Table (RBT)*, in Figure 12. RBT tracks recently seen regions (*Region-ID*) and the bypass decision made for them (*Last-Bypass-Decision*). RBT enables us to find the representative first-conflicting-set in a region (as the first conflicting set would have miss in RBT), keep just that set's RRPV up-to-date, and remember the first-conflicting-set's bypass decision to inform bypass decision for the other lines in the region (as the follower sets would hit in RBT and see previous decision made). We use a 128-entry RBT, which requires <512B of SRAM (performance is relatively insensitive to RBT sizing).

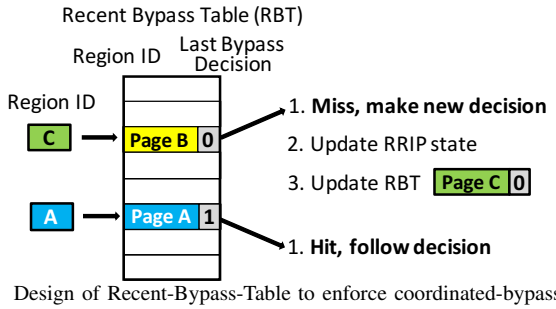


Fig. 12. Design of Recent-Bypass-Table to enforce coordinated-bypass and coordinated-state-update. Demotions only occur on first miss to a region.

Operation of ETR: On cache miss, we index into RBT with Region-ID. If there is an RBT miss, we are currently accessing the representative first-conflicting-set in a region. In this case, we should make a bypass decision based on its RRPV, spend bandwidth to demote state if bypass was chosen, and update the RBT so later accesses can make an informed bypass decision. Otherwise, if there is an RBT hit, the region has been recently accessed and already had a bypass decision made, so we should follow the *Last Bypass Decision* to keep similar install policy and save on demotion bandwidth.

C. Impact on Bandwidth

ETR tries to reduce the bandwidth used for replacement state updates (RRPV promotion and demotion) to improve performance. To understand effectiveness of ETR, we divide the bandwidth used for cache replacement into three parts: installs, promotions, and demotions, and we normalize this consumption to the baseline design that uses bandwidth only for installs. Figure 13 shows the replacement bandwidth usage of base RRIP-AOB and ETR on RRIP-AOB, normalized to Always-Install. ETR saves 70% of replacement state update bandwidth. These bandwidth savings result in speedup.

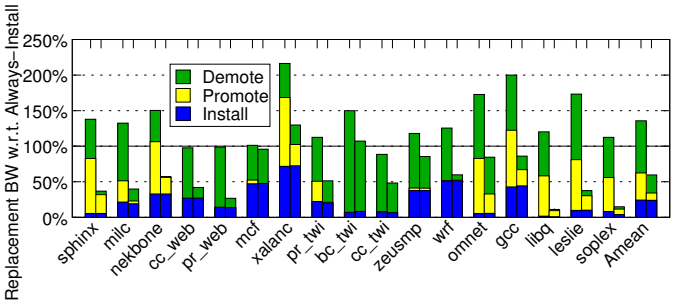


Fig. 13. Replacement and Install bandwidth consumption of base RRIP-AOB [left] and ETR on RRIP-AOB [right], normalized to Always-Install. ETR reduces 70% of the bandwidth consumed in state update.

D. Impact on Performance

Figure 11 shows the performance of RRIP-AOB, ETR on RRIP-AOB, and Ideal RRIP-AOB with no state update costs. ETR on RRIP-AOB bridges 70% of the performance gap between RRIP-AOB and Ideal to achieve 18% speedup, while needing <1KB SRAM storage. Thus, our AOB and ETR make reuse-based policies applicable and effective on DRAM caches.

VI. Results and Discussion

A. Storage Requirements

We analyze the SRAM storage overheads of ETR. ETR requires only a 128-entry 4B-per-entry Recent-Bypass Table, which needs 512B. Thus, our proposal can be easily built with negligible overheads within the memory controller.

For DRAM storage overheads of RRIP-AOB, we use the fact that the DRAM cache has 28 unused bits in the ECC, which can be used for tag and metadata (see Figure 2). Baseline uses 8-10 bits for tag, valid, and dirty bit. RRIP requires just 2 bits for RRPV. Tag-entry becomes 12 bits, which fit in 28 free bits.

B. Impact of Cache Size

Table III shows the speedup of ETR as the size of the DRAM cache is varied from 1GB to 8GB. ETR on RRIP-AOB continues to provide significant speedup across different cache sizes, ranging from 16.4% at 1GB to 13.5% at 8GB. As expected, when the cache size is increased, larger portions of the workload fit in, and there is reduced scope for improvement.

TABLE III
ETR Sensitivity to Cache Size

Cache Size	Avg. Speedup from ETR
1.0GB	16.4%
2.0GB	**18.0%**
4.0GB	17.4%
8.0GB	13.5%

C. Impact of Memory Type

We use a non-volatile main memory for our studies, but our benefits are not limited to NVM-backed systems only. We compare BEAR's Bandwidth Aware Bypass [6] with proposed ETR on DRAM-backed main memory in Table IV. ETR outperforms BEAR by intelligently bypassing lines and achieving better hit-rate and substantial bandwidth benefits.

TABLE IV
ETR ON DRAM-BACKED MEMORY

	Bandwidth Aware Bypass	ETR
SPEC RATE	+7.4%	+17.0%
SPEC MIX	+1.7%	+16.0%
GAP	+4.8%	+26.6%
GMEAN26	+5.7%	+19.0%

D. Impact on 2-Way Designs

Table V shows the impact on average L4 misses when using different replacement policies for 1-way and 2-way caches, normalized to baseline 1-way always-install policy. Proposed RRIP-AOB achieves the highest miss reduction for 1-way caches, as it enables intelligent reuse-based replacement for 1-way caches. RRIP-AOB also achieves the highest miss reduction for 2-way caches, as RRIP-AOB can intelligently bypass when a cache set is storing multiple useful lines.

TABLE V
RRIP-AOB IMPACT ON MISSES FOR 1-2 WAY L4

Replacement Policy	Impact on Avg. L4 Misses
1-way Always-Install	-0.0%
1-way Probabilistic Bypass [6]	-1.6%
1-way RRIP-AOB	**-10.4%**
2-way Random	-14.6%
2-way LRU	-15.5%
2-way RRIP	-19.7%
2-way RRIP-AOB	**-26.6%**

VII. RELATED WORK

A. Replacement / Bypassing policies

Recency-based replacement policies [16], [32], [33] install lines at highest priority, which degenerate into always-install baseline. *Probabilistic* replacement policies [17], [34], become probabilistic bypass [8] in Figure 5. *Frequency*-based replacement [18], [19], [35], [36], [37] or *Reuse*-based replacement [7], [8], [9], [38], [39] try to predict and keep most-frequently used lines in the cache. We design a bypassing version of RRIP, RRIP-AOB, and implement ETR on our RRIP-AOB as an example of this class of policies, but our ETR scheme can be easily used to reduce update-cost of other frequency and reuse-based replacement algorithms [10], [11], [12], [40], [41].

B. Line-based DRAM Caches

In our study, we use the DRAM cache organization used in Intel's Knights-Landing [4] that is direct-mapped and stores each tag next to its data as our baseline. This organization is the commercial implementation of many research efforts that store Tag-With-Data [5], [6], [43], [44], [45] to improve latency and reduce bandwidth consumption. We compare with recent enhancements in Figure 5 (90%-Bypass and BAB [6]). A recent enhancement ACCORD [44] enables associativity via way prediction. We note that ETR on RRIP-AOB has higher potential as it can improve both hit-rate *and* bandwidth (ETR 18% speedup vs. ACCORD 10% speedup). Nonetheless, the ideas can be combined, via tiered ACCORD way-selection then RRIP-AOB bypass-decision, for greater effect (20% speedup).

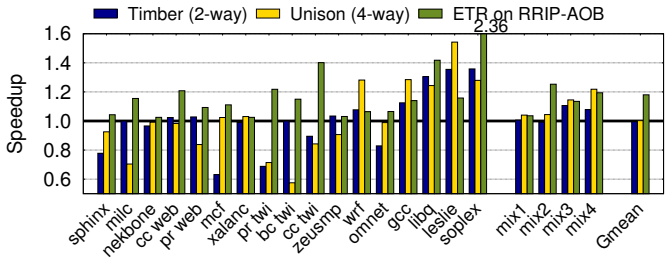


Fig. 14. Speedup of line-based [42] and page-based [14] set-associative DRAM caches, rel. to baseline direct-mapped DRAM cache [4], [5]. Proposed ETR on RRIP-AOB enables direct-mapped DRAM caches to obtain hit-rate benefits of intelligent replacement, without bandwidth cost of set-associative designs

Alternative designs such as Sim et al. [15] store tags via *tag grouping*. For such caches, a tag-only line is placed along with data in the same row buffer [15], [42], [46], [47], [48], and this tag is accessed separately from data. Timber is an enhancement that mitigates tag lookup by employing a tag-cache and exploiting spatial locality [42]. We compare with Timber in Figure 14. Such approaches enable associativity, but pay bandwidth to access and update tags when the tag cache has poor hit-rate (e.g., *mcf* and *pr twi*). Our ETR on RRIP-AOB, on the other hand, enables intelligent replacement without this separate tag access, and outperforms such approaches.

C. Page-based DRAM Caches

One can also use large granularity caches to reduce tag and metadata storage costs [14], [49], [50], [51], [52], [53] and enable set-associative designs. We compare with Unison cache [14] (hardware-managed, 4-way, sectored cache with LRU, separate tag lookup) as a representative of page-based designs, in Figure 14. The associativity and replacement Unison offers enable it to often outperform the baseline; however, the large linesize and separate tag lookup can waste cache capacity and bandwidth to cause slowdown (e.g., *pr twi* and *bc twi*). Meanwhile, ETR on RRIP-AOB enables direct-mapped DRAM caches to get *intelligent replacement* without sacrificing *cache-utilization* or *bandwidth-efficiency*, to outperform such designs.

VIII. CONCLUSION

This paper investigates improving hit-rate for direct-mapped DRAM caches by utilizing reuse-based replacement polices. To make reuse-based policies applicable to direct-mapped caches, we propose a bypass version of RRIP called *RRIP Age-On-Bypass (RRIP-AOB)*, which protects reused lines via bypassing and evicts stale lines via aging state on bypass. However, RRIP-AOB needs per-line reuse counters, and maintaining such state in DRAM costs bandwidth. To tackle state update bandwidth costs, we propose *Efficient Tracking of Reuse (ETR)*. ETR maintains accurate reuse state for just a representative line within a region, and uses that representative's state to inform bypass decisions for surrounding lines. Our evaluations with a 2GB DRAM cache, show that ETR on RRIP-AOB provides a speedup of 18.0% while incurring an SRAM cost of <1KB.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our colleagues from the Memory Systems Lab for their critique and suggestions. This work was supported by a grant from the Semiconductor Research Center (SRC).

REFERENCES

[1] J. Standard, "High bandwidth memory (hbm) dram," *JESD235*, 2013.
[2] JEDEC, *DDR4 SPEC (JESD79-4)*, 2013.
[3] Intel and Micron, "A revolutionary breakthrough in memory technology," 2015.
[4] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, pp. 34–46, Mar 2016.
[5] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *MICRO '12*, pp. 235–246, Dec 2012.
[6] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," in *ISCA '15*, (New York, NY, USA), pp. 198–210, ACM, 2015.
[7] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Trans. Comput.*, vol. 57, pp. 433–447, Apr. 2008.
[8] H. Gao and C. Wilkerson, "A dueling segmented lru replacement algorithm with adaptive bypassing," in *JWAC 2010-1st JILP Worshop on Computer Architecture Competitions: Cache Replacement Championship*, 2010.
[9] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ISCA '10*, (New York, NY, USA), pp. 60–71, ACM, 2010.
[10] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, Jr., and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *MICRO '11*, (New York, NY, USA), pp. 430–441, ACM.
[11] V. Young, C.-C. Chou, A. Jaleel, and M. Qureshi, "Ship++: Enhancing signature-based hit predictor for improved cache performance," in *2nd Cache Replacement Championship (CRC-2 Workshop in ISCA '17)*, 2017.
[12] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *ISCA '16*, pp. 78–89, June 2016.
[13] D. A. Jiménez and E. Teran, "Multiperspective reuse prediction," in *MICRO '17*, (New York, NY, USA), pp. 436–448, ACM, 2017.
[14] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *MICRO '14*, pp. 25–37, IEEE, 2014.
[15] J. Sim, G. H. Loh, V. Sridharan, and M. O'Connor, "Resilient die-stacked dram caches," in *ISCA '13*, (New York, NY, USA), pp. 416–427, ACM, 2013.
[16] W. A. Wong and J.-L. Baer, "Modified lru policies for improving second-level cache behavior," in *HPCA '00*, pp. 49–60, IEEE, 2000.
[17] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *ISCA '07*, (New York, NY, USA), pp. 381–391, ACM, 2007.
[18] J. T. Robinson and M. V. Devarakonda, "Data cache management using frequency-based replacement," in *SIGMETRICS '90*, (New York, NY, USA), pp. 134–142, ACM, 1990.
[19] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The v-way cache: demand-based associativity via global replacement," in *ISCA '05*, pp. 544–555, IEEE, 2005.
[20] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "Usimm: the utah simulated memory module," *University of Utah, Tech. Rep*, 2012.
[21] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, pp. 34–46, Mar 2016.
[22] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the intel optane DC persistent memory module," *CoRR*, vol. abs/1903.05714, 2019.
[23] Intel, "Fact sheet: New intel architectures and technologies target expanded market opportunities," 2018. Accessed: 2019-03-20.
[24] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.
[25] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M.-G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20nm 1.8v 8gb pram with 40mb/s program bandwidth," in *ISSCC '12*, pp. 46–48, Feb 2012.
[26] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, pp. 2201–2227, Dec 2010.
[27] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ISCA '09*, (New York, NY, USA), pp. 2–13, ACM, 2009.
[28] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi, "Pinpointing representative portions of large intel itanium programs with dynamic instrumentation," in *MICRO '04*, pp. 81–92, Dec 2004.
[29] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, Sept. 2006.
[30] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015.
[31] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *ISCA '06*, (Washington, DC, USA), pp. 252–263, IEEE Computer Society, 2006.
[32] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *MICRO '13*, pp. 284–296, ACM, 2013.
[33] Y. Smaragdakis, S. Kaplan, and P. Wilson, "Eelru: simple and effective adaptive page replacement," in *SIGMETRICS '99*, vol. 27, pp. 122–133, ACM, 1999.
[34] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr., and J. Emer, "Adaptive insertion policies for managing shared caches," in *PACT '08*, (New York, NY, USA), pp. 208–219, ACM, 2008.
[35] E. G. Hallnor and S. K. Reinhardt, "A fully associative software-managed cache design," in *ISCA '00*, (New York, NY, USA), pp. 107–116, ACM, 2000.
[36] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," in *SIGMOD '93*, (New York, NY, USA), pp. 297–306, ACM, 1993.
[37] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Trans. Comput.*, vol. 50, pp. 1352–1361, Dec. 2001.
[38] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *MICRO '12*, pp. 389–400, IEEE, 2012.
[39] G. Keramidas, P. Petoumenos, and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in *ICCD '07*, pp. 245–250, IEEE, 2007.
[40] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *MICRO '43*, (Washington, DC, USA), pp. 175–186, IEEE, 2010.
[41] A. Jain and C. Lin, "Rethinking belady's algorithm to accommodate prefetching," in *ISCA '18*, June 2018.
[42] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE CAL*, vol. 11, pp. 61–64, July 2012.
[43] C. Chou, A. Jaleel, and M. K. Qureshi, "Candy: Enabling coherent dram caches for multi-node systems," in *MICRO '16*, pp. 1–13, Oct 2016.
[44] V. Young, C. Chou, A. Jaleel, and M. K. Qureshi, "Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction," in *ISCA '18*, pp. 328–339, June 2018.
[45] V. Young, P. J. Nair, and M. K. Qureshi, "Dice: Compressing dram caches for bandwidth and capacity," in *ISCA '17*, (New York, NY, USA), pp. 627–638, ACM, 2017.
[46] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *MICRO '11*, (New York, NY, USA), pp. 454–464, ACM, 2011.
[47] C.-C. Huang and V. Nagarajan, "Atcache: reducing dram cache latency via a small sram tag cache," in *PACT '14*, pp. 51–60, ACM, 2014.
[48] Z. Wang, D. A. Jimnez, T. Zhang, G. H. Loh, and Y. Xie, "Building a low latency, highly associative dram cache with the buffered way predictor," in *SBAC-PAD '16*, pp. 109–117, Oct 2016.

[49] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in *ISCA '13*, (New York, NY, USA), pp. 404–415, ACM, 2013.
[50] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in *ISCA '15*, (New York, NY, USA), pp. 211–222, ACM, 2015.
[51] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *HPCA '16*, pp. 237–248, IEEE, 2016.
[52] G. H Loh, N. Jayasena, J. Chung, S. K Reinhardt, M. O'Connor, and K. McGrath, "Challenges in heterogeneous die-stacked and off-chip memory systems," in *3rd Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW-3)*, 02 2012.
[53] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," in *MICRO '17*, (New York, NY, USA), pp. 1–14, ACM, 2017.