ACCORD: Enabling Associativity for Gigascale DRAM Caches by Coordinating Way-Install and Way-Prediction

Vinson Young[†], Chiachen Chou[†], Aamer Jaleel[*‡], and Moinuddin K. Qureshi[†]

†*Georgia Institute of Technology*   ‡*NVIDIA*
{vyoung,cchou34,moin}@gatech.edu, ajaleel@nvidia.com

Abstract—**Stacked-DRAM technology has enabled high bandwidth gigascale DRAM caches. Since DRAM caches require a tag store of several tens of megabytes, commercial DRAM cache designs typically co-locate tag and data within the DRAM array. DRAM caches are organized as a direct-mapped structure so that the tag and data can be streamed out in a single access. While direct-mapped DRAM caches provide low hit-latency, they suffer from low hit-rate due to conflict misses. Ideally, we want the hit-rate of a set-associative DRAM cache, without incurring additional latency and bandwidth costs of increasing associativity. To address this problem, *way prediction* can be applied to a set-associative DRAM cache to achieve the latency and bandwidth of a direct-mapped DRAM cache. Unfortunately, conventional way prediction policies typically require per-set storage, causing multi-megabyte storage overheads for gigascale DRAM caches. If we can obtain accurate way prediction without incurring significant storage overheads, we can efficiently enable set-associativity for DRAM caches.**

This paper proposes *Associativity via Coordinated Way-Install and Way-Prediction (ACCORD)*, a design that steers an incoming line to a "preferred way" based on the line address and uses the preferred way as the default way prediction. We propose two way-steering policies that are effective for 2-way caches. First, *Probabilistic Way-Steering (PWS)*, which steers lines to a preferred way with high probability, while still allowing lines to be installed in an alternate way in case of conflicts. Second, *Ganged Way-Steering (GWS)*, which steers lines of a spatially contiguous region to the way where an earlier line from that region was installed. On a 2-way cache, ACCORD (PWS+GWS) obtains a way prediction accuracy of 90% and retains a hit-rate similar to a baseline 2-way cache while incurring 320 bytes of storage overhead. We extend ACCORD to support highly-associative caches using a *Skewed Way-Steering (SWS)* design that steers a line to at-most two ways in the highly-associative cache. This design retains the low-latency of the 2-way ACCORD while obtaining most of the hit-rate benefits of a highly associative design. Our studies with a 4GB DRAM cache backed by non-volatile memory shows that ACCORD provides an average of 11% speedup (up to 54%) across a wide range of workloads.

Keywords-**Stacked DRAM, HBM, Associativity, Cache**

I. INTRODUCTION

As modern computer systems pack more and more cores on a processor chip, both bandwidth and capacity of memory systems must scale proportionally. Advancements in memory packaging and interconnect technology have enabled stacking several DRAM chips, thereby offering 4-8X bandwidth of conventional DIMM-based DRAM [1]. Meanwhile, emerging non-volatile memory technology, such as phase change memory (PCM) [2], [3], [4] and 3D XPoint [5], provide 4-8X higher capacity than DRAM. Therefore, future memory systems are likely to consist of high-bandwidth stacked DRAM and high-capacity non-volatile memory [6], [7], [8], [9]. An attractive option is to architect stacked DRAM as a hardware-managed cache and place it between on-die caches and memory [10], [11], [12], [13], [14], [15], [16], [17], [18].

Architecting stacked DRAM as a cache has several challenges, including designing and accessing a tag-store of several megabytes. For example, a 4GB cache consists of 64 million 64-byte lines, which requires a tag-store of 128MB (even if each tag is only 2 bytes). Therefore, practical designs co-locate tags with data in the stacked DRAM. For example, alloy cache [11] proposes storing tag next to data at a line granularity in a direct-mapped cache, so that a single access can provide both data and the associated tag, which allows for quick determination of a hit or a miss. Unlike its set-associative counterpart [12], [13], [14], [16], the direct-mapped organization is attractive as it eliminates additional accesses to determine location of lines and optimizes for hit latency. Commercial processors such as Intel Knights Landing (KNL) [10] take a similar approach and architect DRAM cache at 64-byte linesize, direct-mapped, tags-stored-with-data (in unused ECC bits). In this paper, we focus on such practical DRAM caches that place tags with data.

Although direct-mapped DRAM caches minimize bandwidth overhead, they suffer from a low cache hit rate because of conflict misses. While trading off the hit rate for lower hit latency may be acceptable when main memory access latency is similar to DRAM-cache latency (e.g., DDR-based main memory), such a tradeoff may not be suitable for systems that have much higher memory latency than that of DRAM caches (e.g., non-volatile memory, or NVM). Ideally, for NVM-based memory systems, we would like to improve the hit rate and reduce long-latency accesses to main memory. One simple way that improves the cache hit rate is to make DRAM caches set-associative. However, set-associative DRAM caches require efficient mechanisms that (1) determine the matching location (way) on a hit, (2) determine that a line does not exist in any way on a miss, (3) determine the matching way on a writeback operation, and (4) maintain the replacement state.
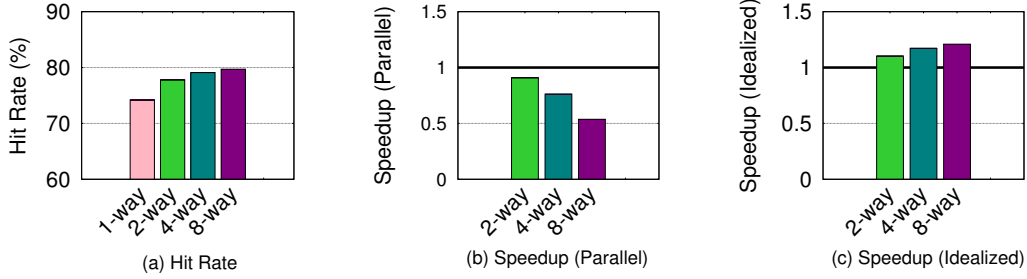

Figure 1. Impact of increasing associativity from 1-way to 8-way on (a) hit-rate (b) performance of parallel lookup design that streams the entire set (c) performance of an idealized set-associative design (BW and latency of 1-way). Speedup is averaged across all workloads w.r.t. a baseline direct-mapped cache. See Section III-A for detailed methodology.

A straightforward way to implement a set-associative cache is to stream out all N ways on each access (a design referred to as *parallel lookup*). While such a design may obtain higher hit rate than a direct-mapped cache, it suffers from high bandwidth overheads, causing performance degradation. Figure 1(a) shows that increasing the set associativity of a 4GB DRAM cache from 1-way to 8-way improves the hit rate from 74% to 80%. Unfortunately, the 8-way cache with parallel lookup degrades performance, shown in Figure 1(b). If we could obtain the hit rate of a set-associative cache and maintain the latency and bandwidth of a direct-mapped cache, performance could improve significantly, 21% with an 8-way cache shown in Figure 1(c). Thus, enabling set associativity for DRAM caches must be done in a bandwidth-efficient manner for performance.

An alternative design, *serial lookup*, checks the ways one-by-one and stops when a tag matches. Serial lookup improves bandwidth consumption but introduces serialization latency as the way lookups are now dependent on each other. The hit latency of this design can improve if we intelligently choose which way to lookup first, using *way prediction*. Unfortunately, conventional way predictors [19], [20], [21], [22] rely on per-set metadata (e.g., tracking the most-recently used line in a set) or per-line metadata (e.g., tracking partial tags of each line in a set). Consequently, these designs are impractical for gigascale caches as they would incur several megabytes of SRAM storage overhead. Ideally, we want accurate way prediction without the storage overheads.

Prior approaches for way prediction need significant storage, as way prediction is independent of way install. If we coordinate way install with way prediction, we could predict ways with high accuracy and low storage overhead. For example, consider an extreme case in a 2-way cache, in which we install all lines with even tags to way-0, and odd tags to way-1. In such a design, based on the line address, we would obtain 100% accuracy for way prediction. Unfortunately, such a design would degenerate into a direct-mapped cache, as a line is always forced to go to exactly one location. However, this example provides an important insight: *by guiding the rules of where lines can go, we can effectively make the cache more predictable without requiring significant storage for way prediction*. Based on this insight, this paper makes the following contributions:

Contribution-1: We propose ACCORD (*Associativity via Coordinated Way Install and Way Prediction*), a framework that allows low-cost way prediction for DRAM caches by coordinating the cache install policy and the way prediction policy. ACCORD steers an incoming line to a "preferred way" based on the line address and uses the preferred way as the default way prediction. ACCORD provides a robust framework for obtaining associativity when associativity is beneficial, but not degrading performance when the system is insensitive to increased cache associativity.

Contribution-2: We propose a *Probabilistic Way-Steering (PWS)* policy that steers lines into the "preferred way" with a high probability (say 85%) while using this preferred way as a *stateless* way prediction. Since PWS allows the line to be installed in another way with some probability (15%), the lines conflicting for the same preferred location will still have a chance to eventually co-reside within the same set. With this steering mechanism, PWS obtains most of the 2-way hit rate but obtains high accuracy for way prediction (83%).

Contribution-3: We propose a *Ganged Way-Steering (GWS)* policy that steers the lines of a spatially contiguous region to the same way where a line from that region was recently installed. By coordinating the install decisions across sets, GWS allows accurate way prediction by simply tracking the way of a few recently accessed regions and predicting that other lines from that region are resident in the same way. We show that tracking 64 recent regions (storage overhead of less than 320 bytes) with GWS is sufficient to provide high way prediction accuracy for workloads with high spatial locality, and combining GWS with PWS provides an accuracy of 90% for a 2-way cache.

In addition, we extend ACCORD to support higher levels of set associativity and show that the main obstacle for a highly set-associative cache is the cost of miss confirmation (the need to check all of the ways in a set).

We perform evaluations using 21 workloads on a 16-core system with a 4GB DRAM cache and non-volatile memory. Our studies show that ACCORD with 2-ways provides 7.3% speedup (up to 40%), which is close to the 10% speedup of an idealized 2-way cache. Furthermore, ACCORD with increased associativity provides average speedup of 11% (up to 54%), while retaining similar bandwidth consumption to that of a direct-mapped cache.
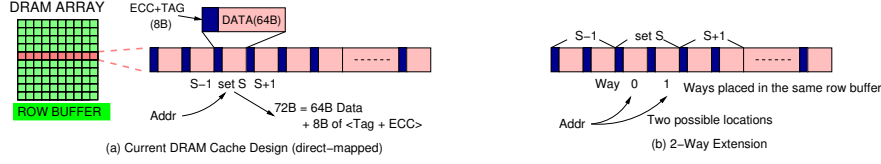
Figure 2. (a) Organization of practical DRAM caches (Intel KNL): Each access indexes a direct-mapped location and transfers 72 bytes that has tag and data. (b) Extending to a 2-way cache. Ways in the same set are placed in the same row buffer. Each memory address has two possible locations (ways).

II. BACKGROUND AND MOTIVATION

We now discuss the trade-offs in designing direct-mapped and set-associative DRAM caches.

A. Organization of Practical DRAM Caches

Recent research work have enabled fine-grained (64B line size) DRAM caches in a low-cost manner [11], [17]. To avoid on-die tag storage overhead, these studies propose co-locating tags with data in a stacked-DRAM array. These proposals optimize for low hit latency, even if it comes at the expense of slight reduction in hit rate [11]. For example, the alloy cache organizes the DRAM cache as a direct-mapped structure and alloys tags and data together, which form a 72-byte unit. Such a unit is streamed out on a cache access. Therefore, one access to a direct-mapped location retrieves one data line and the corresponding tag. Thus, a cache hit or miss is quickly determined with the single tag, and cache hits can be serviced immediately after the access. This direct-mapped tags-with-data design is attractive for latency, as hits and misses can be serviced with just one DRAM access. Commercial designs, like Intel Knights Landing [10], also adopt such a low-latency DRAM cache–a direct-mapped, 64B linesize structure that stores tags as part of line in unused ECC bits,[1] shown in Figure 2(a). In this paper, we consider such practical gigascale DRAM caches, a line-granularity organization that co-locates tag with the data line.

B. Challenges in Set-Associativity

Trading off the cache hit rate for low hit latency may be acceptable when the memory latency is similar to DRAM cache latency. However, this trade-off is unacceptable when the main memory latency is much longer than the DRAM cache latency. For example, when main memory is NVM, memory access latency can be roughly 2X-4X as long as DRAM cache latency. Therefore, it is important to optimize for the DRAM cache hit rate while retaining low DRAM cache hit latency. A straightforward way of improving the cache hit rate is to build a set-associative DRAM cache. For example, we extend the practical DRAM cache design to a 2-way cache, shown in Figure 2(b). The tag of a way is associated with the way, and ways in the same set are placed in one row buffer [12]; in this case, each memory address has

two possible locations. However, we identify the following obstacles in designing set-associative DRAM caches:

1) **Determining Hit Location:** On an access, the line can be present in any way of a set in a set-associative DRAM cache. We may need to check all the ways to obtain the requested line, thus incurring the bandwidth and latency overheads associated with accessing multiple lines. Ideally, we want to implement a set associative DRAM cache while retaining the single lookup of a direct-mapped DRAM cache.

2) **Miss Confirmation:** On miss, we need to ensure that the line is not present in the DRAM cache. This typically requires checking all ways in a set, which incurs bandwidth overhead proportional to DRAM cache set associativity. For set-associative DRAM caches, we want to reduce the bandwidth overheads incurred by miss confirmation.

3) **Writeback Probe:** On a writeback operation to a set-associative DRAM cache, a *writeback probe* may be needed to determine the way in which the line is resident. For a direct-mapped DRAM cache, simply knowing that the line is resident is sufficient to avoid a writeback probe operation (by keeping the DRAM cache presence, or DCP bit, in the L3 cache [17]). However, for a set-associative DRAM cache, we additionally need to know "which way" to write back to. To determine the correct way to write back to on a writeback, we extend the DCP scheme to store way information as well. We note that any associative DRAM cache will require this extension to enable write performance.

4) **Replacement Policy:** A set-associative cache relies on a replacement policy to choose a victim line. For any intelligent replacement policy, an update of the replacement state is performed on hits and insertions (e.g., counter update [23]). Since DRAM caches store the tags with the line, updating the replacement state incurs a DRAM write operation (to the line). Unfortunately, performance overhead because of extra bandwidth for such updates far outweighs performance benefits from the improved hit rate [11]. In fact, using an update-free replacement policy (e.g., random) provides better performance.[2] As such, we use random replacement for set-associative DRAM caches throughout this paper.

C. Options for Implementing Set-Associativity

In an N-way set-associative cache, a line can be resident in any of the N locations. Ideally, we desire a low-latency and low-bandwidth implementation of a set-associative DRAM cache. We now discuss some options for implementing a set-associative DRAM cache.

---

[1]The tags of the KNL design are kept in the ECC space. The stacked DRAM technology used in KNL provides a 16-byte bus for data and two-byte bus for ECC, for a total width of 18 bytes. Fortunately, we need only 9 bits for providing SECDED on a 16-byte entity, which means 7 bits are left unused in each burst. A cache line of 64-byte is transferred over four bursts, therefore there are 28 unused bits in the ECC space, which is sufficient for storing the tag information for the given data line.

[2]For example, using LRU replacement for a 2-way DRAM cache degrades performance by 9% compared to random replacement.
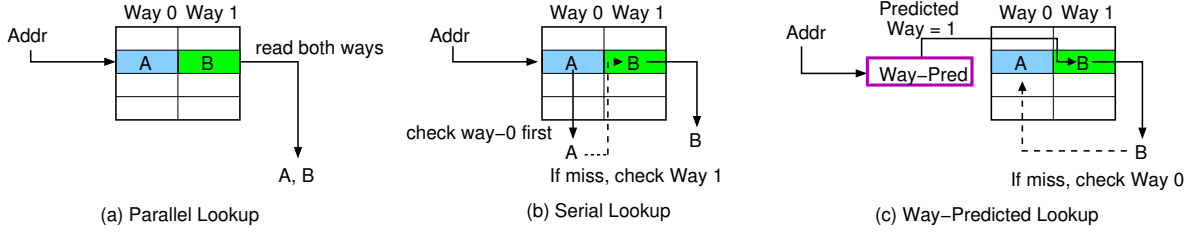
Figure 3. Comparison of accessing lines in a 2-way DRAM cache. (a) Parallel Lookup: One cache request reads all ways in the corresponding set. (b) Serial Lookup: A request first reads way-0; if miss, it checks way-1. If data is in way-1, this dependent way checking incurs serialization penalty. (c) Way-Predicted Lookup: A request first reads a predicted way; if miss, it checks the other way.

1) Parallel Lookup: A straightforward way to locate data in an N-way DRAM cache is to read all N lines of a set on each access, shown in Figure 3(a). We refer to such a design as a *parallel lookup* design. Table I shows the number of line transfers for a cache hit and a cache miss for this design. The number of transfers represents bandwidth overhead that is the number of lines transferred on the bus, and the number of accesses indicates the serialization penalty (i.e., the number of dependent checks). While parallel lookup may be practical for SRAM caches, the bandwidth overhead of streaming N lines on each access becomes prohibitive for DRAM caches.

Table I
THE NUMBER OF CACHE ACCESSES AND LINE TRANSFERS FOR LOOKING UP A N-WAY SET-ASSOCIATIVE CACHE

Cache Organization	Actions on A Hit	Actions on A Miss	Hit Rate
Direct-mapped	1 access 1 transfer	1 access 1 transfer	low
Parallel Lookup (N-way)	1 access N transfers	1 access N transfers	high
Serial Lookup (N-way)	~N/2 accesses ~N/2 transfers	N accesses N transfers	high
Way Predicted (N-way)	~1 access ~1 transfer	N accesses N transfers	high

2) Serial Lookup: Alternatively, we can check the lines sequentially (see Figure 3(b)). We refer to this design as a *serial lookup* design, which reduces the bandwidth for transferring lines on a hit from N to ~(N+1)/2 on average (see Table I). However, it introduces latency as the lookups are now serially dependent on each other (N accesses). If we can predict which way the line is likely to be in, we can reduce hit latency further.

3) Way-Predicted Lookup: We can reduce serialization penalty by intelligently choosing which way to check first using *Way-Prediction* (see Figure 3(c)). We refer to this design as a *Way-Predicted Lookup* design. On a hit, if we can predict the way accurately, we can service hits with just one DRAM access (see Table I). This organization achieves the hit latency of a direct-mapped cache while maintaining the hit rate of an N-way cache. However, achieving high way-prediction accuracy at low storage cost is difficult in practice. Furthermore, the bandwidth overheads of miss confirmation is still N lookups.

D. Conventional Way-Predictors: Challenge in Scaling to Gigascale DRAM Caches

Way Prediction has been proposed for SRAM caches (L1 or LLC) for reducing latency and power. For L1, tracking the most-recently-used way in each set (MRU Pred [19], [20]) provides high accuracy as the access stream of an L1 cache has high temporal locality. Unfortunately, such locality is typically not visible to secondary caches (L2/L3/L4), so MRU prediction tends to be not as effective. Secondary caches can use a partial-tag (e.g., 4-bits) design for each line to avoid cache lookup [21], [22], [24], [25], [26], [27]. Unfortunately, both MRU Pred and Partial-Tag design require per-set or per-line storage. Such schemes would incur storage overhead of several MBs for a 4GB DRAM cache, in Table II.

Table II
ACCURACY AND STORAGE OF WAY PREDICTORS FOR A 4GB CACHE

	Rand Pred	MRU Pred	Partial-Tag
Storage	0B overhead	4MB overhead	32MB overhead
2-way	50.0%	85.7%	97.3%
4-way	25.0%	74.3%	91.6%
8-way	12.5%	63.2%	81.2%

Table II shows that simply predicting a random location has low accuracy (labeled as *Rand Pred*). Additionally, MRU prediction accuracy also degrades significantly at higher associativity. The partial-tag design has good accuracy but incurs an impractical storage overhead of 32MB. If we are to implement way prediction effectively at the size of DRAM caches, we need a way-predictor that is both accurate and requires lower storage overhead than conventional designs.

E. Insight: Way-Steering for Way-Prediction

In a conventional set-associative design, any way in a set can be replaced (determined by the replacement policy), thus complicating the way prediction. We observe that if we *steer* the incoming line to a "preferred way" based on the line address at install time, way prediction can simply use the preferred way as the default prediction at access time. The coordination between way install and way prediction improves the prediction accuracy. Based on this insight, we develop a low-overhead, low-latency, and low-bandwidth way prediction for set-associative DRAM caches. We explain our methodology before describing our proposal.

III. Methodology

A. Framework and Configuration

We use USIMM [28], an x86 simulator with detailed memory system model. We extend USIMM to include a DRAM cache. Table III shows the configuration used in our study. 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 baseline contains a 4GB direct-mapped DRAM cache that places tags with data in the unused ECC bits [10]. The parameters of our DRAM cache is based on HBM technology [1]. The main memory is based on non-volatile memory and assumed to have a latency similar to PCM [2], [3], [4], [6]: the read latency is 2-4X and the write latency is 4X as long as those of DRAM [29].

Table III
SYSTEM CONFIGURATION

Processors	16 cores; 3.0GHz, 2-wide OoO
Last-Level Cache	8MB, 16-way
DRAM Cache	
Capacity	4GB
Bus Frequency	500MHz (DDR 1GHz)
Configuration	8 channel, 128-bit bus
Aggregate Bandwidth	128 GB/s
tCAS-tRCD-tRP-tRAS	13-13-13-30 ns
Main Memory (PCM)	
Capacity	128GB
Bus Frequency	1000MHz (DDR 2GHz)
Configuration	2 channel, 64-bit bus
Aggregate Bandwidth	32 GB/s
tCAS-tRCD-tRP-tRAS-tWR	13-128-8-143-160 ns

B. Workloads

We run benchmark suites of SPEC 2006 [30], GAP [31], and HPC. For SPEC, we perform studies on all 9 benchmarks that have at least 5% speedup potential going from 1-way to 8-way, along with 2 workloads that are less sensitive to set associativity. GAP is graph analytics with real data sets (`twitter`, `web sk-2005`) [32]. The evaluations execute benchmarks in rate mode, where all cores execute the same benchmark. In addition to rate-mode workloads, we evaluate 10 mixed workloads, which are created by choosing 16 of the 16 SPEC workloads that have at least two miss per thousand instructions (MPKI). Table IV shows L3 miss rates, memory footprints, and performance potential with ideal 8-way cache for the rate-mode workloads used 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. For other workloads that are neither memory bound nor sensitive to set associativity of DRAM caches, we present performance of all 46 workloads evaluated (29 SPEC, 10 SPEC-mix, 6 GAP, and 1 HPC) in Section VI-A.

Table IV
WORKLOAD CHARACTERISTICS

Suite	Workload	L3 MPKI	Footprint	8-Way Potential Speedup
SPEC	soplex	29.8	3.6 GB	2.43
	leslie	17.0	1.2 GB	1.63
	libq	25.6	512 MB	1.55
	gcc	5.8	2.9 GB	1.27
	zeusmp	5.3	3.3 GB	1.18
	wrf	7.4	2.2 GB	1.18
	omnet	20.8	2.4 GB	1.17
	xalanc	4.3	3.0 GB	1.09
	mcf	83.1	26.1 GB	1.06
	sphinx	13.7	293 MB	1.01
	milc	25.5	9.0 GB	0.99
GAP	pr twitter	121.7	30.5 GB	1.15
	cc twitter	108.7	18.6 GB	1.15
	bc twitter	81.1	26.9 GB	1.11
	pr web	17.8	30.3 GB	1.07
	cc web	9.2	18.6 GB	1.05
HPC	nekbone	6.4	111 MB	1.04

IV. Design of ACCORD

A. An Overview of ACCORD

Way-prediction can enable a set-associative cache to maintain the hit latency of a direct-mapped cache but the hit-rate of an associative cache. Conventional way predictors are designed independent of the way install policy of the cache and typically incur significant storage overhead.[3] The cache install policy decides the way in which an incoming line is installed. Generally, a line has full flexibility to be installed in any way. There tends to be no correlation between the line address and the install way. If we can instead coordinate way-install with way-prediction, we can obtain high accuracy for way prediction while incurring negligible storage overhead. To this end, we propose *Associativity via Coordinated Way-Install and Way-Prediction (ACCORD)*, a design that steers an incoming line to a "preferred way" based on line address and uses that preferred way as way prediction.
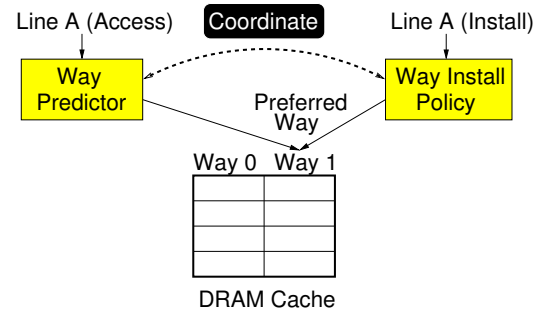

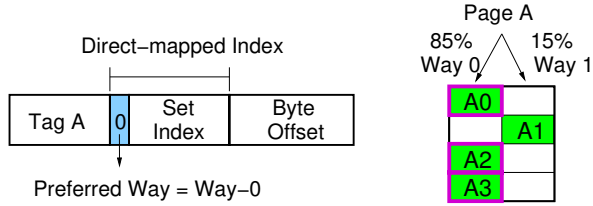Figure 4.   Overview of ACCORD. Coordinating Way-Install and Way-Prediction using Way Steering

Figure 4 shows how ACCORD changes the cache install policy to prefer a particular way based on the line address, in a process which we refer to as *Way Steering*. For way prediction, the preferred way is determined based on the line address, and this way information is used as the prediction.

[3]There have been proposals for storage-free position-based prediction [33], [19], but, these proposals require bandwidth-intensive swaps to maintain accuracy. We evaluate such proposals in Section VII.

The Way Steering policies in ACCORD do not restrict the line to always be installed in the preferred way. The policies are designed such that it is possible, although less likely, that the line can be installed in the non-preferred way. In this section, we propose and evaluate two low-cost and effective policies for way-steering in a 2-way DRAM cache. We then show how to extend ACCORD to support higher levels of set associativity in Section V-A.

B. Probabilistic Way-Steering (PWS)

The install policy in practical set-associative DRAM cache is likely to employ random replacement (to avoid bandwidth for updating replacement state on a hit). Random replacement chooses any way in a set with equal probability. For example, in a 2-way cache, either way will be picked with 50% probability. However, using the line address, we can bias the install policy to select a particular way with a higher probability. Based on this insight, we propose our first way-steering policy, *Probabilistic Way Steering (PWS)*.

Figure 5. Probabilistic Way-Steering (PWS): (a) deciding the preferred way based on tags and (b) installing line in the preferred way based on preferred-way install probability (e.g. 85%).

PWS determines the preferred way of a line based on the tag of the line, as shown in Figure 5(a). If the tag is even, Way-0 is preferred; if the tag is odd, Way-1 is preferred. On an install, PWS chooses the preferred way with a given probability, *Preferred-Way Install Probability (PIP)*. For example, if PIP=85%, then, PWS chooses the preferred way with 85% likelihood, as shown in Figure 5(b). For a two-way cache, PWS with PIP=50% is an unbiased policy that is identical to the baseline random replacement policy, whereas PIP=100% degenerates into a direct-mapped cache. On an access, the way-prediction statically predicts the preferred way based on the tag. As we install in the preferred way 85% of the time, we will find the line in the preferred way 85% of the time. As such, the prediction accuracy of PWS is approximately equal to PIP. Note that a high value of PIP can degrade cache hit-rate as it reduces the flexibility of an associative cache. We analyze the sensitivity to PWS's PIP threshold using a synthetic kernel and our workloads.

1) Analyzing PWS Using Cyclic Reference Model: We analyze the effectiveness of PWS with various PIP values using a cyclic reference model [34], [23] to model hit-rate of two conflicting lines. Let *a* and *b* denote the address of
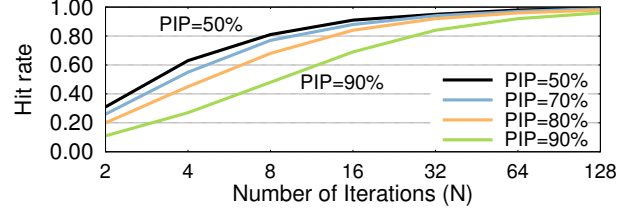

Figure 6. Impact of PIP on hit-rate of a 2-way cache for a cyclic-reference kernel. PIP=80% provides hit-rate near PIP=50% (unbiased random policy).

two cache lines that map to the same set of a 2-way cache. These lines are accessed one after the other. A temporal sequence that repeats for N times is represented as $(a,b)^N$. If N=1, the cache would get zero hits due to compulsory misses. However, when N¿1, we would expect the cache to provide hits in steady state. Note that a direct-mapped cache would always provide 0% hit-rate due to thrashing.

Figure 6 shows the hit-rate of this kernel as N is varied from 2 to 128 using PWS while varying PIP from unbiased (50% for a two-way cache) to 90%. We find that PIP=70% and PIP=80% maintain similar hit-rate compared to the unbiased random replacement policy (PIP=50%) as the two lines are installed in the two separate ways quickly. Interestingly, with enough re-use, *even PIP=90% will eventually learn to use both ways* and provide improved hit-rate. Thus, tuning PIP effectively allows PWS to trade off a small amount of hit-rate (i.e., speed of learning to use both ways) for predictability.

2) Impact of PWS on Hit-Rate and Performance: PWS must balance between the dueling needs of high hit-rate as well as high way-prediction accuracy. Table V shows the hit-rate and performance of PWS as a function of PIP, averaged over all our workloads. At PIP of 80% or below, PWS provides most of the hit-rate of a 2-way cache, and achieves high way-prediction accuracy (similar to PIP). Note that even at PIP=90%, PWS still provides most of the hit-rate benefit of a 2-way design, as lines that constantly thrash eventually use the other way. However, at PIP=100%, the design simply degenerates into a direct-mapped cache. We observe that, PIP between 80% and 85% provides the best trade off between hit rate and way-predictability. Performance of PWS is maximized (5.6% speedup) at PIP=85%. Thus, we use PIP=85% for the rest of this paper. Note that PWS performance improvements incur zero storage overhead.

Table V
AVERAGE HIT-RATE AND SPEEDUP OF PWS

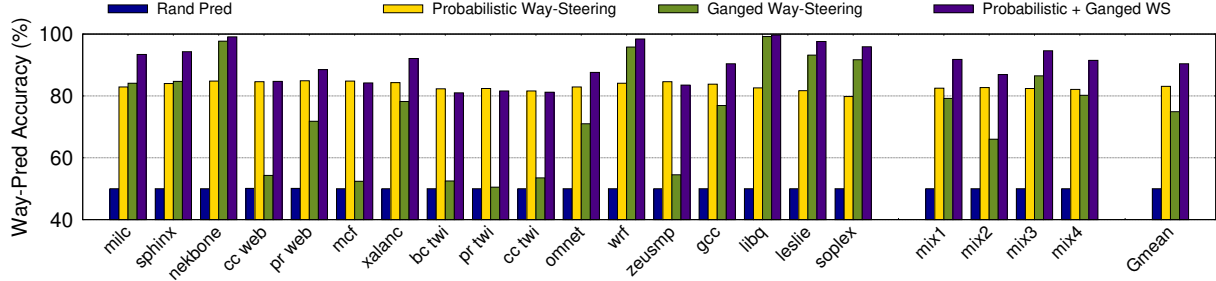| Organization	Hit-Rate	WP Acc.	Speedup
2-way (Unbiased, PIP=50%)	77.5%	50.0%	2.6%
2-way PWS (PIP=60%)	77.5%	59.8%	3.7%
2-way PWS (PIP=70%)	77.5%	69.4%	4.7%
2-way PWS (PIP=80%)	77.3%	78.6%	5.5%
2-way PWS (PIP=85%)	77.2%	83.1%	5.6%
2-way PWS (PIP=90%)	76.9%	87.8%	5.3%
Direct-Mapped (PIP=100%)	74.2%	100.0%	0.0%

Figure 7. Accuracy of way-predictors for a 2-way cache. While probabilistic way-steering (PWS) provides a 83% accuracy, ganged way-steering (GWS) is 75% accurate. Combining GWS with PWS provides 90% accuracy.

C. Ganged Way-Steering (GWS)

PWS guides most of the lines to the preferred way; however, some lines in the region can still go to different ways. This is because PWS makes the install decisions on each miss, and decides if the incoming line should be installed in the preferred way or the non-preferred way. In conventional cache management schemes, the install decisions of one set has no bearing on the install decisions on another set. We develop an insight that if we could coordinate the install decisions across sets, then we can obtain even higher way-prediction accuracy. For example, if multiple lines of a spatially contiguous region miss the cache, then rather than making independent install decisions for each line in the region, we could make the install decision for the first line in the region and install subsequent lines from the region in the same way as the earlier line from that region. Based on this insight, we propose *Ganged Way-Steering (GWS)*.
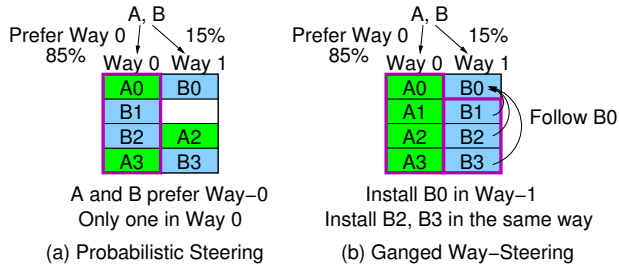

Figure 8. The benefit of coordinating install decisions across sets: (a) PWS install lines of each region (A and B) into either way. (b) GWS steers later lines from the region into the same way earlier line was installed. The purple boxes denote lines that can be way-predicted.

1) GWS: Insight and Proposal: Figure 8 compares PWS and GWS for a spatial pattern from two regions (A and B). PWS can install lines from these regions in either way. However, with GWS, install decisions are made only for the first line in the region, and subsequent lines are steered such that they follow the decision made by the first line. With GWS, the preferred way for a line at install time is the way where recent lines from that region were steered to. And way-prediction is decided based on the way of the last accessed line of that region. For a workload with good spatial locality, GWS can get near-ideal way-prediction accuracy by predicting last-way seen (¿90%). GWS only needs a method to track the install-way of recently missed lines and the way-location of recently accessed lines.

2) GWS: Implementation: Figure 9 shows our implementation of GWS for the install policy. GWS tracks the last-way-installed for recently installed regions in a *Recent Install Table (RIT)*. On an install, GWS first checks the RIT to see if the cache has recently installed lines from that region. If yes (RIT hit), GWS steers subsequent installs to that same way. If no (RIT miss), GWS defaults to a different way-install policy (e.g., unbiased or PWS), and inserts an entry into RIT.
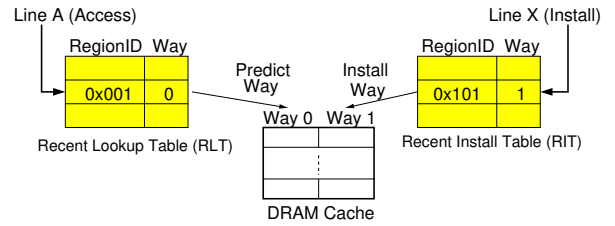

Figure 9. Design of Ganged Way-Steering (GWS)

GWS also tracks last-way-seen for a few recently accessed regions in a *Recent Lookup Table (RLT)*. On a DRAM cache access, GWS checks the RLT to see if it has recently accessed that region. If yes (RLT-hit), GWS predicts last-way-seen for that region. If no (RLT miss), GWS cannot provide a way-prediction, and defaults to a way-prediction policy (e.g., random or PWS). This coordinated Ganged-Install and Ganged-Prediction provides high way-prediction accuracy for workloads with high spatial locality within a region. Each entry of RIT and RLT consists of a RegionID (˜19 bits), and a way information (1 bit for a two-way cache). The region size is 4KB, and we use a 64-entry RIT and 64-entry RLT (total 160B storage overhead) as tracking 64 regions captures most of the benefit of GWS.

3) Impact on Way-Prediction Accuracy: Figure 7 shows the way-prediction accuracy of PWS, GWS, and PWS+GWS. PWS has an accuracy close to 85% (since PIP=85%). GWS has near-ideal accuracy for workloads with high spatial locality (e.g., *nekbone* and *libq* have 99% accuracy). Note that GWS can only way-predict on RLT hits and defaults to random prediction on RLT miss. As such, GWS has limited accuracy when pages are sparsely accessed or the workload footprint is large (e.g., *mcf* and *pr twi*). GWS, combined with the way-prediction policy of PWS, improves way-prediction accuracy to 90%.
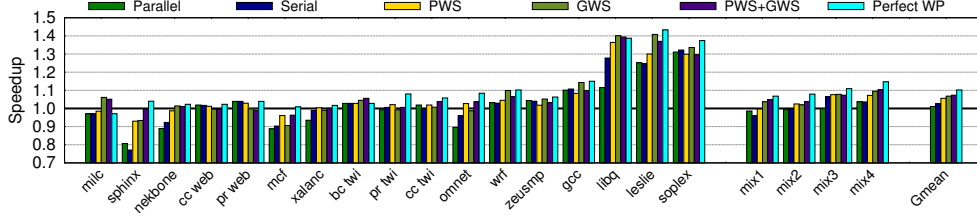
334

Figure 10. Speedup from 2-way DRAM Cache. Parallel Lookup wastes bandwidth and Serial lookup incurs latency. Combining Probabilistic Way-Steering and Ganged Way-Steering provides 7.3% speedup, close to (10%) with perfect.

4) Impact on Cache Hit-Rate: Table VI shows the impact of Way-Steering on hit rate. Increasing associativity to 2 ways increases hit-rate from 74.2% to 77.5%. GWS retains the hit-rate of a 2-way cache, as it simply increases the granularity at which replacement happens. PWS, on the other hand, trades hit-rate for predictability, so there is small hit-rate degradation under PWS+GWS, to 77.3%. Overall PWS+GWS achieves high way-prediction accuracy (>90%), while keeping most of the hit-rate of a 2-way cache (77.3%).

Table VI
SENSITIVITY OF HIT-RATE TO PWS THRESHOLD

	Direct-mapped	2-Way Rand	PWS	GWS	PWS+GWS
Amean	74.2%	77.5%	77.2%	77.7%	77.3%

5) Impact on Performance: Figure 10 shows the speedup for parallel tag-lookup, serial tag-lookup, PWS, GWS, PWS+GWS, and perfect way-prediction. Parallel tag-lookup wastes significant bandwidth to look up both tags on every access. Serial tag-lookup performs slightly better because of the reduction of hit-locating bandwidth. Perfect way-prediction achieves 10.2% speedup servicing hits with one access. Probabilistic Way-Steering biases the cache to act as direct-mapped for 85% of installs. For sparsely-accessed workloads, PWS allows the cache to still predict correctly 85% of the time. PWS provides a good baseline prediction accuracy for the cache. However, accessing lines in unpreferred ways still need 2 accesses. For workloads with high amounts of page-level locality, we can do better with Ganged Way-Steering.

Ganged Way-Steering improves way-predictability for workloads with high spatial locality by steering ganged patterns to install in the same way. For workloads with high spatial locality (e.g. *libq, nekbone*), GWS gets near-ideal prediction accuracy. This translates to near-ideal hit latency, hit-rate (see Table VI), and performance for gang-accessed workloads, for a total of 6.8% speedup. However, GWS underperforms direct-mapped by up to 10% for a few workloads (*mcf, sphinx*) that have limited spatial locality. The combination of PWS and GWS allows the cache to predict ganged patterns nearly perfectly, and sparse accesses with 85% accuracy, for an average way-prediction accuracy of 90%. PWS+GWS achieves 7.3% speedup out of the 10.2% maximum possible speedup (perfect way-prediction) for a 2-way cache, with only one workload experiencing small degradation. The next section shows how these ideas can be used for higher levels of effective associativity.

V. EXTENDING ACCORD FOR HIGHER ASSOCIATIVITY

Thus far, we have analyzed ACCORD for 2-way caches, and found that PWS+GWS is quite effective. Next step is to scale to N-ways. However, there is a major obstacle in making DRAM cache highly set-associative: the prohibitive cost of miss confirmation. For an N-way cache, a cache request needs to do N lookups to confirm a miss. Consider a scenario where an 4-way cache receives 100 requests, and 60 of them hit. Even with perfect way prediction, the 40 misses still incur 160 lookups (40*4) for confirmation, shown in Figure 11(a). The bandwidth for servicing misses far exceeds the bandwidth for servicing hits. We found that extending ACCORD to four ways achieves only 3% speedup and extending it to eight ways causes 6% slowdown. Therefore, to enable high set associativity, we must address the cost of miss confirmation.

A. Skewed Way Steering (SWS)

If each line in a set can be restricted to only two possible locations, the miss confirmation needs to check only two ways, significantly reducing the bandwidth overhead of miss confirmation. Figure 11(b) shows an example. Lines *A*, *B*, and *C* map to the same set and thrash in their preferred ways (Way-1); besides the preferred way, each line now can be in one and only one other location determined by the hash of tags: *A* in 1 or 3, *B* in 1 or 2, and *C* in 1 or 0. On misses, *A*, *B*, and *C* might first try to install in their preferred way. But on subsequent misses, *B* could be installed in its alternate way Way-2, and *C* in Way-0. Therefore, with this restriction, most of lines can still fit in the cache [35], but miss confirmation cost is reduced to two lookups. Based on this insight, we propose *Skewed Way-Steering (SWS)*, which restricts lines to having exactly one *Alternate* location, instead of allowing the line to have (N-1) non-preferred locations in a N-way set-associative cache.
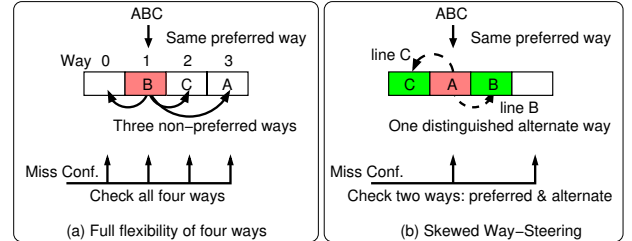

Figure 11. (a) Full flexibility of four ways. Miss confirmation checks all locations. (b) Skewed Way-Steering. Each line in the same set has one distinct alternate location. Miss confirmation checks only two locations.
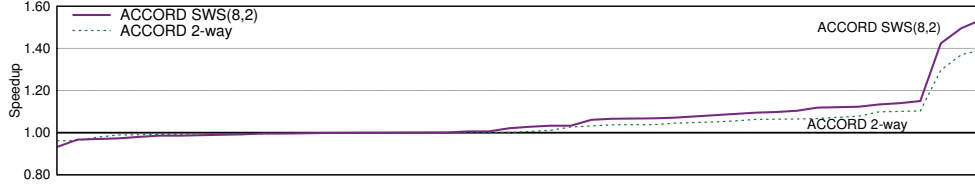
Figure 12. Speedup of ACCORD over 46 workloads (including ones not sensitive to memory or hit rate).

The Alternate way is selected using an intelligent hashing function such that the Alternate location is guaranteed to be different from the Preferred location. For a 4-way cache, the Preferred Way is selected based on the two LSB bits of the tag. To determine the Alternate Way, we look at two-bit groups in the tag starting from the third LSB to the MSB, such that it does not match the Preferred Way and the first such mis-match is regarded as the Alternate Way. In the rare case that all 2-bit values in the tag are identical, the Alternate Way is obtained by inverting the Preferred Way.

For our previous example, where A, B, and C map to the same set, if we have a 4-way cache with SWS, then the cache cannot accommodate A, B, and C if they all map to the same pair of ways (the likelihood of this is 1/36). The idea of SWS can be extended to N ways. For an N way set-associative cache, the N ways are all still organized in the same row buffer to reduce miss lookup cost. While we implement SWS with only one Alternate location, SWS can be extended to support multiple Alternate locations for flexibility, albeit at higher cost of miss-confirmation. We call such organizations with N ways and k hashes as SWS(N,k).

B. Impact on Cache Hit Rate

SWS provides flexibility of where to place lines, while simultaneously removing worst-case tag-lookup bandwidth. We show aggregate hit-rate for SWS(4,2) and SWS(8,2) and compare it with the hit-rate of direct-mapped, 2-way ACCORD (PWS+GWS), and 8-way cache. In Table VII, we can see that SWS(8,2) offers improved hit-rate over 2-way ACCORD, at similar miss-confirmation cost. Note that going from 2-way to 8-way provides little hit-rate improvement, and, SWS provides 1/3rd of the benefit for nearly free.

Table VII
HIT-RATE OF DIFFERENT ACCORD DESIGNS

	Direct-mapped	ACCORD (2-way)	SWS (4,2-way)	SWS (8,2-way)	8-Way
Hit Rate	74.2%	77.3%	77.7%	77.9%	79.7%

C. Impact on Performance

Figure 13 shows the speedup of ACCORD 2-way, AC-CORD with SWS(4,2), and ACCORD with SWS (8,2). ACCORD with SWS(8,2) provides the highest hit rate and speedup. However, it can degrade performance for *sphinx*, because *sphinx* already had 99% hit rate, and any increase in bandwidth consumption or reduction in row buffer hit rate (due to N-way) degrades performance. Overall, ACCORD with SWS(8,2) provides improved hit-rate at direct-mapped bandwidth cost, achieving a total speedup of 10.6%.
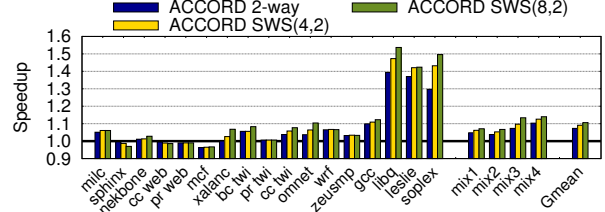

Figure 13. Speedup from extending ACCORD using SWS. SWS(8,2) provides an average speedup of 10.6%.

VI. RESULTS AND ANALYSIS

A. Evaluations Over More Workloads

Figure 12 shows speedup of ACCORD with 2-way and SWS(8,2) across all 46 workloads evaluated, including 29 SPEC, 10 SPEC-mix, 6 GAP, and 1 HPC workloads. On average, ACCORD improves performance by 4% and 6% for the 2-way and SWS(8,2) configurations. And, ACCORD improves the performance of the 10 Mix workloads by 7% and 11% on average. More importantly, Figure 12 shows that ACCORD maintains performance across all workloads, including ones that are not sensitive to increased associativity.

B. Impact of Cache Size

We use a default cache size of 4GB for our studies. Table VIII shows the speedup of ACCORD as the size of the DRAM cache is varied from 1GB to 8GB. ACCORD provides significant speedup across different cache sizes, ranging from 13.6% at 1GB to 8.6% 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 VIII
SENSITIVITY TO CACHE SIZE

Cache Size	Avg. Speedup from ACCORD
1.0GB	13.6%
2.0GB	12.0%
4.0GB	**10.6%**
8.0GB	8.6%

C. Storage Requirements

We analyze the storage overheads of ACCORD in Table IX. Both PWS and SWS do not require any storage overheads. The only storage required is for GWS. Each entry in the Recent Lookup Table entry (RLT) and Recent Install Table (RIT) is 20 bits (1 valid bit + 19-bit tag). With a 64-entry RLT and 64-entry RIT, the total storage overheads would be 320 bytes. Thus, ACCORD enables associativity while incurring a total storage overhead of 320 bytes.
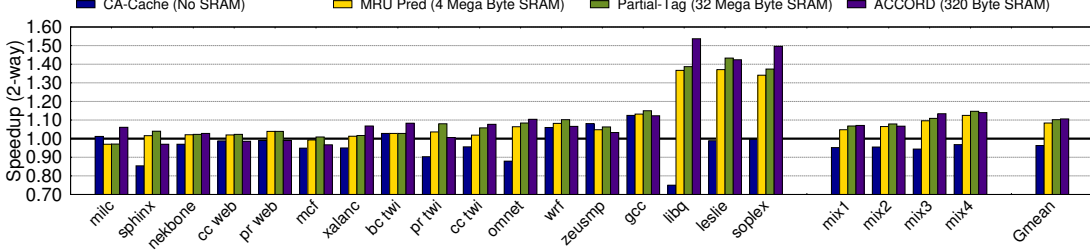
Figure 14. Speedup of way predictors and ACCORD for a 2-way cache. ACCORD obtains high performance while avoiding multi-MB SRAM overheads.

Table IX
STORAGE REQUIREMENTS OF ACCORD

ACCORD Component	Storage
Probabilistic Way-Steering	0 Bytes
Ganged Way-Steering	320 Bytes
Skewed Way-Steering	0 Bytes
ACCORD	320 Bytes

D. Energy of Off-chip Memory System

Figure 15 shows DRAM cache + memory power, energy consumption, and energy-delay-product (EDP) of a system using ACCORD 2-way and ACCORD SWS(8,2), normalized to a baseline direct-mapped cache. We model power and energy for stacked DRAM with [36], [37], and model power and energy for non-volatile memory with [6]. ACCORD provides similar DRAM-cache energy consumption with bandwidth-efficient design, and reduces main memory energy consumption by increasing DRAM cache hit rate. Overall, ACCORD reduces energy use by 3% and EDP by 14%.
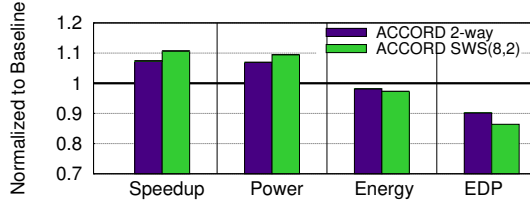

Figure 15. Memory system energy with ACCORD. ACCORD keeps similar DRAM-cache energy consumption but reduces main memory energy.

VII. RELATED WORK

Skewed-Associative Caches. Prior work proposed *skewed associative cache* [35], [38] design that allows a cache line to reside in two possible locations (based on a hash of memory address). If the first hash of two lines collide, it is unlikely the second hash location will also collide. Our extension of SWS for ACCORD can be thought of as a skewed cache with a direct-mapped location and a skew location chosen within a group of 4 (or 8) physically-contiguous lines. However, we tailor our design for DRAM such that the possible skews are resident in the same row buffer, to reduce the latency to access these locations. Furthermore, SWS relies on way-prediction so that most hits can be serviced with one access.

Hash-Rehash Cache. The Hash-Rehash and Column-Associative caches (CA-cache) [33], [39] serially check two indices (preferred and alternate) in a direct-mapped cache

and move the line physically to the preferred index, if a hit happens at the alternate index. Thus, a hit to the preferred index can be serviced in one access, whereas a hit to the alternate index requires not only two accesses but also the bandwidth to swap between the two indices. The probability of CA-cache to obtain a hit with one access is similar to having a two-way cache with MRU-based way predictor (see Table X). However, CA-cache incurs significant bandwidth in swaps, even when the cache is not sensitive to associativity. Figure 14 shows CA-cache degrades performance by 3.7%, due to its increased bandwidth consumption (e.g., *sphinx*). Meanwhile, ACCORD provides robust performance as it does not require additional bandwidth consumption (i.e., swaps) to maintain its prediction accuracy.

Table X
COMPARISON OF DIFFERENT WAY PREDICTORS

	CA-Cache	MRU Pred	Partial-Tag	ACCORD
Storage (2-way)	(0MB)	(4MB)	(32MB)	(320 bytes)
Accuracy (2-way)	85.2%	85.7%	97.3%	90.4%
Accuracy (4-way)	N/A	74.3%	91.6%	90.1%
Accuracy (8-way)	N/A	63.2%	81.2%	90.1%

Way Prediction. There has been research on using way prediction in on-chip caches to reduce hit latency [19], [25] as well as energy consumption [40], [20]. For reducing hit latency, existing proposals, such as PSA Cache [19] utilize a most-recently-used (MRU) bit per set; however, MRU-based approaches are primarily designed for L1 caches and do not scale well to subsequent levels of the cache hierarchy due to filtering of spatial locality. When applied to DRAM caches, the MRU-based predictor has poor accuracy, as shown in Table X. The storage overhead of MRU-based prediction is still significant for gigascale DRAM cache. For example, 4MB SRAM overhead (50% overhead for 8MB LLC) for a 4GB DRAM cache. We compare performance in Figure 14.

Several proposals use partial-tags[21], [22], [24], [25], [26] to reduce power in on-chip last-level caches. The accuracy of partial-tag way predictors does not scale with increasing associativity due to increased false tag-matches. Furthermore, the storage required by partial-tag based predictors increases linearly with the number of lines in the cache, and it becomes prohibitively large for DRAM cache. For example, a predictor with 4-bit partial-tag incurs 32MB SRAM overhead for our 4GB DRAM cache. As shown in Figure 14, ACCORD matches the performance of prior way predictors while avoiding the multi-megabyte storage overhead.
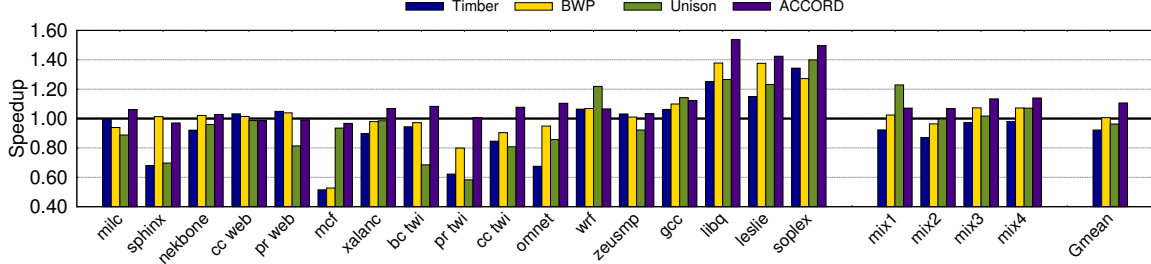
Figure 16.   Speedup from TIMBER, Buffered Way-Predictor (BWP), Unison, and ACCORD.

Selective Direct-Mapping [41], [40] proposes to identify conflicting lines with a small 16-entry victim list, and use associativity for only those conflicting lines. In concept, this can be thought of as similar to our PWS scheme where we use direct-mapped most of the time to improve prediction accuracy. However, their scheme diverges at the level of giga-scale last-level cache, where most of the quickly-conflicting lines are filtered by the associativity of earlier levels and the remaining conflict intervals are at a much larger time scale.

Prior work has also looked at Bloom filters for cache hit prediction[27]. However, Bloom filters require a storage over-head of 240MB for a 4GB cache for obtaining an accuracy similar to partial-tags. ACCORD, on the other hand, obtains performance similar to storage-intensive way predictors while obviating the storage and bandwidth overheads.

Comparison to Other Line-Based DRAM Caches. In our study, we use a practical DRAM cache organization that associates tags with each data line. A different approach enables set associativity by *tag grouping*. For example, Aggressive Tag-Cache [42] groups tags of 15-way set into one cache line and accesses it separately from data, and, Timber [18] uses a small structure (8KB) that keeps recently accessed tag lines. Figure 16 shows that 2-way Timber improves workloads that have good spatial locality and are sensitive to associativity, such as *soplex*, but degrades other workloads by as much as 49% (e.g., *mcf*). The degradation results from the low hit rate of the tag cache for workloads with poor spatial locality. For such workloads, Timber needs two accesses for every cache request (one for tag and the other for data). On average, ACCORD improves performance by 11% while Timber has 8% slowdown.

Buffered Way Predictor. Buffered Way Predictor [43] proposes to maintain the way-location of every line in memory in a region in the DRAM Cache, and cache them in an on-chip buffer. We compare with a 2-way BWP in Figure 16. For our 128GB memory, a 2-way BWP would take 256MB of our 4GB DRAM Cache to store all 1-bit way information, and would cache way information in a 128KB way-cache. For workloads with good spatial locality or small memory footprint, the 128KB way-cache has high hit-rate and is able to provide accurate predictions. Unfortunately, for workloads with large memory footprint and poor spatial locality (e.g., *mcf* or *pr twi*), the way-cache provides poor hit-rate and consequently requires significant extra bandwidth to

obtain and update the way prediction entries in DRAM cache. On average, our proposed ACCORD improves performance by 11% while BWP improves performance by 1%.

Comparison to Page-Based DRAM Caches. The baseline DRAM cache in our paper is organized at a line granularity (64B) with tags as part of line. An alternate approach is use sectored caches and page-granularity caches to reduce the storage required for the tag-store. The recently proposed *Unison Cache*[16] architects the DRAM cache as a 4-way set-associative cache with 1KB lines and 64B sectors. Unison stores the tag and sector bits in the DRAM alongside data. To access the DRAM cache, the Unison Cache controller sends two concurrent DRAM requests: one for the tags and another for the data from the predicted way. And, the page-granularity enables Unison to obtain accurate way prediction.

We compare ACCORD to a 4-way Unison Cache with 512B lines and 64B sectors. Figure 16 shows the performance of ACCORD and Unison relative to the direct-mapped DRAM cache. For SPEC, Unison outperforms a direct-mapped design due to its increased associativity. However, for workloads with large memory footprint and low spatial locality (e.g., GAP workloads of *cc*, *bc*, and *pr*), the large linesize of Unison prevents large amounts of the cache from being used. Accounting for such worst-case workloads, the performance of Unison breaks even with direct-mapped design. In comparison, ACCORD achieves the improved hit-rate of an associative cache, while maintaining the bandwidth-efficiency and high utilization of line-based designs.

VIII. CONCLUSIONS

This paper addresses the challenge of enabling set associativity for line-granularity DRAM caches at low bandwidth and low storage cost. We propose ACCORD (*Associativity via Coordinated Way Install and Way Prediction*), a framework that enables low-cost way prediction by steering lines to "preferred" ways on install time and predicting the preferred way on access time. We propose two policies: *probabilistic way-steering*, which steers cache line installs to a preferred way based on its tag and predicts the preferred way on an access, and *ganged way-steering*, which steers subsequent installs in a region to the same way and predicts accesses in that region with last observed way. These policies provide a 90% way-prediction accuracy at negligible storage overhead (320 bytes). To obtain higher levels of set associativity (e.g., 8-way) at reduced overheads of miss confirmation, we propose

skewed way-steering policy that steers lines to at most two locations in an N-way set-associative cache. Our evaluations on a 4GB DRAM cache show that ACCORD outperforms direct-mapped organization by 11% on average. We develop ACCORD over the existing DRAM cache design used in industry, and we believe the simplicity of our solution will make it appealing for industrial adoption.

Acknowledgements

References

[1] J. Standard, "High bandwidth memory (hbm) dram," *JESD235*, 2013.

[2] M.K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, pp. 1–134, 2011.

[3] Y.C. et al., "A 20nm 1.8v 8gb pram with 40mb/s program bandwidth," in *ISSCC '12*, Feb 2012, pp. 46–48.

[4] H.S.P. Wong, S. Raoux, S. Kim, J. Liang, J.P. Reifenberg, B. Rajendran, M. Asheghi, and K.E. Goodson, "Phase change memory," *IEEE*, vol. 98, pp. 2201–2227, Dec 2010.

[5] Intel and Micron, "A revolutionary breakthrough in memory technology," 2015.

[6] 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: ACM, 2009, pp. 2–13.

[7] M.K. Qureshi, V. Srinivasan, and J.A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA '09*. New York, NY, USA: ACM, 2009, pp. 24–33.

[8] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: A hybrid pram and dram main memory system," in *DAC '09*, July 2009, pp. 664–669.

[9] A. Bivens, P. Dube, M. Franceschini, J. Karidis, L. Lastras, and M. Tsao, "Architectural design for next generation heterogeneous memory systems," in *Memory Workshop (IMW), 2010 IEEE International*. IEEE, 2010, pp. 1–4.

[10] 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.

[11] 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*. IEEE Computer Society, 2012, pp. 235–246.

[12] 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: ACM, 2011, pp. 454–464.

[13] J. Sim, G.H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean dram cache for effective hit speculation and self-balancing dispatch," in *MICRO '12*. IEEE, 2012, pp. 247–257.

[14] 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: ACM, 2013, pp. 404–415.

[15] S. Franey and M. Lipasti, "Tag tables," in *HPCA 2015*.

[16] D. Jevdjic, G.H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *MICRO 2014*.

[17] 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: ACM, 2015, pp. 198–210.

[18] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE Computer Architecture Letters*, vol. 11, pp. 61–64, July 2012.

[19] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *HPCA 1996*.

[20] H.C. Chen and J.S. Chiang, "Low-power way-predicting cache using valid-bit pre-decision for parallel architectures," in *(AINA'05)*, vol. 2, March 2005, pp. 203–206 vol.2.

[21] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," in *ISLPED '04*, Aug 2004, pp. 126–131.

[22] F.M. Sleiman, R.G. Dreslinski, and T.F. Wenisch, "Embedded way prediction for last-level caches," in *ICCSD 2012*.

[23] 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: ACM, 2010, pp. 60–71.

[24] J.J. Valls, J. Sahuquillo, A. Ros, and M.E. Gmez, "The tag filter cache: An energy-efficient approach," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, March 2015, pp. 182–189.

[25] D.H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *MICRO '99*. IEEE, 1999, pp. 248–259.

[26] J.J. Valls, A. Ros, J. Sahuquillo, and M.E. Gomez, "Ps-cache: An energy-efficient cache design for chip multiprocessors," *J. Supercomput.*, vol. 71, pp. 67–86, Jan. 2015.

[27] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.H.S. Lee, "Way guard: a segmented counting bloom filter approach to reducing energy for set-associative caches," in *ISLPED '09*. ACM, 2009, pp. 165–170.

[28] 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.

[29] *DDR4 SPEC (JESD79-4)*, JEDEC, 2013.

[30] J.L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, Sep. 2006.

[31] S. Beamer, K. Asanovic, and D.A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015.

[32] T.A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, Dec. 2011.

[33] A. Agarwal, J. Hennessy, and M. Horowitz, "Cache performance of operating system and multiprogramming workloads," *ACM Trans. Comput. Syst.*, vol. 6, pp. 393–431, Nov. 1988.

[34] S. McFarling, "Cache replacement with dynamic exclusion," in *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, 1992, pp. 191–200.

[35] A. Seznec, "A case for two-way skewed-associative caches," in *ISCA '93*. New York, NY, USA: ACM, 1993, pp. 169–178.

[36] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens, "System and circuit level power modeling of energy-efficient 3d-stacked wide i/o drams," in *DATE '13*. San Jose, CA, USA: EDA Consortium, 2013, pp. 236–241.

[37] K.T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B.C. Lee, and M. Horowitz, "Rethinking dram power modes for energy proportionality," in *MICRO '12*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 131–142.

[38] D. Sanchez and C. Kozyrakis, "The zcache: Decoupling ways and associativity," in *MICRO '10*. IEEE, 2010, pp. 187–198.

[39] A. Agarwal and S.D. Pudar, *Column-associative caches: A technique for reducing the miss rate of direct-mapped caches*. ACM, 1993, vol. 21, no. 2.

[40] M.D. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *MICRO '01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 54–65.

[41] B. Batson and T.N. Vijaykumar, "Reactive-associative caches," in *PACT '01*, 2001, pp. 49–60.

[42] C.C. Huang and V. Nagarajan, "Atcache: reducing dram cache latency via a small sram tag cache," in *PACT '14*. ACM, 2014, pp. 51–60.

[43] 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*, Oct 2016, pp. 109–117.