Building a Low Latency, Highly Associative DRAM Cache with the Buffered Way Predictor

Zhe Wang[†] Daniel A. Jiménez[‡] Tao Zhang[♯] Gabriel H. Loh[♮] Yuan Xie[§]
[†]Intel Labs [‡]Texas A&M University [♯]Pennsylvania State University
[♮]Advanced Micro Devices, Inc. [§]University of California, Santa Barbara
[†]zhe2.wang@intel.com [‡]djimenez@cse.tamu.edu [♯]tao.zhang.0924@gmail.com
[♮]gabriel.loh@amd.com [§]yuanxie@ece.ucsb.edu

***Abstract*—The emerging die-stacked DRAM technology allows computer architects to design a last-level cache (LLC) with high memory bandwidth and large capacity. There are four key requirements for DRAM cache design: minimizing on-chip tag storage overhead, optimizing access latency, improving hit rate, and reducing off-chip traffic. These requirements seem mutually incompatible. For example, to reduce the tag storage overhead, the recent proposed LH-cache co-locates tags and data in the same DRAM cache row, and the Alloy Cache proposed to alloy data and tags in the same cache line in a direct-mapped design. However, these ideas either require significant tag lookup latency or sacrifice hit rate for hit latency.**

To optimize all four key requirements, we propose the *Buffered Way Predictor* (BWP). The BWP predicts the way ID of a DRAM cache request with high accuracy and coverage, allowing data and tag to be fetched back to back. Thus, the read latency for the data can be completely hidden so that DRAM cache hitting requests have low access latency. The BWP technique is designed for highly associative block-based DRAM caches and achieves a low miss rate and low off-chip traffic. Our evaluation with multi-programmed workloads and a 128MB DRAM cache shows that a 128KB BWP achieves a 76.2% hit rate. The BWP improves performance by 8.8% and 12.3% compared to LH-cache and Alloy Cache, respectively.

I. Introduction

Die-stacked memory [13], [18], [21], [8], [12], [7] has been envisioned as a solution for future chip-multiprocessor (CMP) design to address the challenges of the "memory wall." Commercial die-stacked DRAM products, such as Hybrid Memory Cube (HMC) [17] and High-bandwidth Memory (HBM) [10], [11], have provided opportunities to re-architect modern microprocessor design. Such die-stacked DRAM technology allows designers to integrate a large memory varying from a few tens of megabytes (MB) up to several gigabytes (GB) [1], [13], [23], [14] into the processor package, providing a last-level cache design with high memory bandwidth and large capacity. We identify four competing optimization requirements: minimizing on-chip storage overhead, optimizing access latency, reducing miss rate and reducing off-chip traffic. Optimizing one requirement usually comes at the cost of another requirement.

This work was done while Zhe Wang were with Texas A&M University.

Page-based DRAM caches [9], [8] mitigate the tag overhead of DRAM caches. However, fetching a page of data from main memory leads to high contention for memory resources, increasing the access latency of memory requests on the critical path. Moreover, the page-based DRAM cache design has a fragmentation problem that wastes bandwidth.

Block-based DRAM caches [2] eliminate the off-chip bandwidth waste. However, the size of the tag array can be tens or hundreds of megabytes for a large-capacity cache. Storing tags on the processor die requires an impractically large tag space. The alternative is to place the LLC tags in the on-chip DRAM. However, doing so requires two DRAM accesses upon each cache hit: one to look up the tag and the other to return the data. Each cache hit could take approximately twice the delay of a single access to the on-package DRAM.

Previous proposals mitigate the large tag overhead by placing the tag and data in the same row [13], co-locating data and tag in the same block using direct map DRAM cache design [18] or building an on-chip tag table [3]. However, these ideas still require significant tag lookup latency [13], sacrifice hit rate for hit latency [18], and require significant on-chip storage capacity [3].

In this paper, we propose the *Buffered Way Predictor* (BWP) technique for a highly associative block-based DRAM cache. The technique optimizes all four key requirements of DRAM cache design. The BWP technique keeps a small on-chip way predictor to predict the way IDs of DRAM cache requests. A DRAM cache block can now be accessed according to the way ID as predicted by the BWP. The BWP is able to achieve a low miss rate by exploiting the spatial locality of DRAM cache requests. As a result, the technique provides fast DRAM cache access with high associativity and a lower miss rate than a direct-mapped cache. The low miss rate and block-based DRAM cache design eliminate off-chip bandwidth waste. Our evaluation with multi-programmed workloads and a 128MB DRAM cache shows that a 128KB BWP achieves a 76.2% hit rate. The BWP improves performance by 8.8% and 12.3% compared to LH-cache and Alloy Cache, respectively.

II. Background and Related Work

To effectively architecture DRAM caches, there are still several challenges that must be considered, including reducing the overhead of supporting tags, improving hit rate, optimizing access latency, and reducing off-chip traffic. DRAM cache organization can be categorized into two classes: *page-based* and *block-based* designs.

A. Page-based DRAM Cache Design

A page-based DRAM cache [9], [8], [2], [7] allocates cache lines at a coarse granularity. It mitigates the tag overhead of DRAM caches, but it has a fragmentation problem that wastes bandwidth. Moreover, fetching a page of data from main memory can lead to high contention for memory resources, increasing the access latency of memory requests on the critical path.

Jevdjic *et al.* [8] propose the Footprint Cache that adopts a footprint predictor to identify blocks within a page that will be reused during the page's lifetime. To eliminate wasted bandwidth, it only fetches the reused blocks within a page into the DRAM cache. However, the Footprint Cache does not scale well with increasing DRAM cache size. On the other hand, the Unison Cache [7] improves on the Footprint Cache by incorporating tag metadata directly into the stacked DRAM to enable scalability. It proposes a way prediction technique that detects the way ID before obtaining the tag block from DRAM cache. However, their way prediction technique does not work well for a highly associative block-based DRAM cache.

B. Block-based DRAM Cache Design

A block-based approach eliminates the memory bandwidth waste. However, to efficiently design a large block-based DRAM cache, one of the key challenges is mitigating the large overhead of tags. Placing DRAM cache tags into an on-chip SRAM array incurs an intolerable on-chip storage overhead. To reduce the tag store and access overhead, Recent work proposes various optimization techniques that are discussed below.

Co-locate Data and Tag within the Same DRAM Cache Row. Previous work [13], [21] proposes co-locating tags and data in the same DRAM cache row to eliminate the large SRAM tag array. A 2KB DRAM cache row contains 29 data blocks co-located with 3 tag blocks for a DRAM cache set. Correspondingly, Loh and Hill [13] propose *Compound Access Scheduling* to speed up the cache access. On a DRAM cache hit, the row is opened to obtain the tag first, then the data can be accessed with a row-buffer hitting operation. To serve misses faster, the LH-cache uses the MissMap structure to keep track of whether a block resides in the DRAM cache. Thus, a DRAM cache miss bypasses the DRAM cache and goes directly to main memory. To eliminate the hardware overhead of MissMap structure, Sim *et al.* [21] propose the Mostly Clean Cache technique.It uses a Hit-Miss predictor to predict whether the requested data resides in the DRAM cache. However, in this work the data returned from main memory must wait until a tag mismatch is verified in the DRAM cache, still incurring a significant latency overhead. Note that the problems of previous work [18], [21] with set-associative cache design are similar: the tag and data have serial accesses. In other words, only after obtaining the tag and finding out the location of the data can the DRAM cache data be accessed.

Cache Tags on Chip. Huang *et al.* [5] propose using a tag cache to cache the DRAM cache tags. The technique prefetches tags from nearby sets in the hope that subsequent requests hit in the tag cache thus reducing the tag access latency. However, in this technique, each data burst can only deliver the tags of one DRAM cache set. Prefetching tags with a large granularity consumes energy and bandwidth. On the other hand, prefetching tags at small granularity can result in a low tag cache hit rate causing large DRAM cache tag access latency.

Alloy DRAM Cache Structure. Exploring the trade-off between hit latency and miss rate of DRAM caches, Qureshi and Loh [18] propose the direct-mapped Alloy Cache. Instead of associatively searching for the tag, the putative position of the data can be directly obtained from the physical address. The Alloy Cache places the data and tag together so that they can be streamed out in two back-to-back data bursts[1]. A Memory Access Predictor (MAP) is proposed to predict whether the data presents in the DRAM cache, allowing main memory to service the request quickly without incurring significant memory traffic. Thus, the Alloy Cache achieves fast tag lookup, reducing DRAM cache hit latency. However, by forgoing associativity, that work sacrifices DRAM cache hit rate for hit latency.

Even though a set-associative DRAM cache design can have a low miss rate, the serial tag-data accesses in the previous proposal [13], [21] leads to significant latency overhead. We propose the Buffered Way Predictor (BWP) technique to predict the way ID of a request. A data block can be fetched according to the predicted way ID without the aid of the tag. Thus, a correctly predicted request achieves similar access latency to the Alloy Cache while the set associativity can further improve the hit rate.

III. The Buffered Way Predictor

In this section, we introduce the Buffered Way Predictor (BWP). The BWP is used to cache the way IDs of DRAM cache data blocks. The way blocks (WB) and associated partial tags are stored in the BWP. Each 64-byte WB contains way IDs of 128 DRAM cache data blocks. As way IDs are packed densely in a WB, it can achieve high spatial

[1]We assume one data burst delivers 64 bytes.

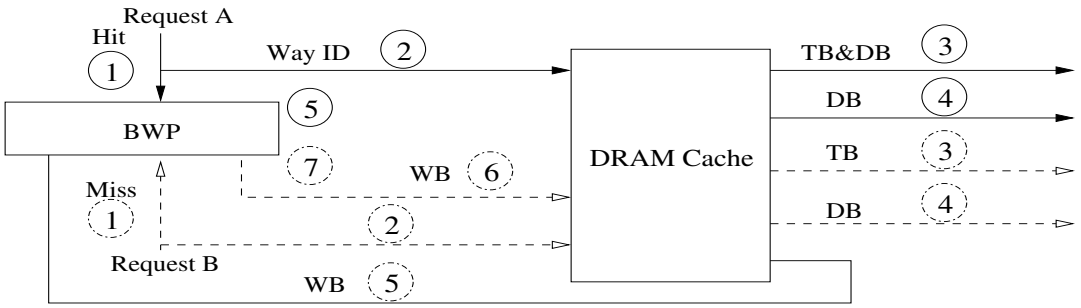

Figure 1. An example of the mechanism of the BWP

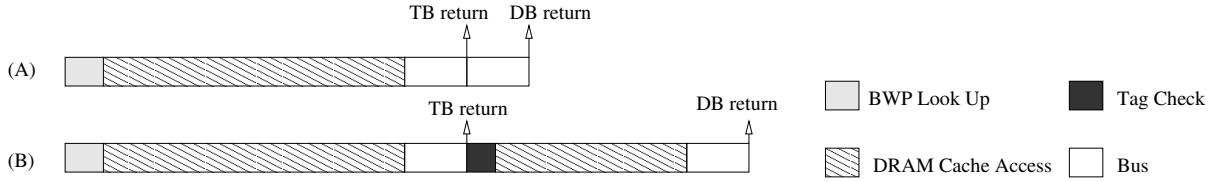

Figure 2. Latency breakdown for DRAM cache hit (A) BWP hit with correct way prediction (B) BWP hit with incorrect way prediction or BWP miss

locality. When a way ID for a request is present in the BWP (a BWP *hit*), the BWP provides the way ID for the request, allowing the data and tag together to be directly read from the DRAM cache. Note that a BWP hit might still be a misprediction if the buffered way ID is incorrect. Compared to LH-Cache, the BWP effectively reduces the access latency due to the elimination of the discrete tag-data accesses.

Data and tags are co-located in the same DRAM cache row. Each DRAM cache row reserves two WBs for storing the way IDs. When a way ID for a request is not present in the BWP (a BWP *miss*), the corresponding WB will be fetched from the DRAM cache and placed in the BWP. Modified WBs are written back to the corresponding DRAM cache row on eviction from the BWP.

Figure 1 shows an example of the BWP. For each DRAM cache hit, there are two possibilities: hitting in the BWP or missing in the BWP.

Possibility 1: Assuming request *A* hits in the BWP (step 1), the BWP predicts the way ID of request *A* stored in the DRAM cache. Then the request *A* is sent to the DRAM cache for service (step 2). The tag block and data block specified by the predicted way ID are fetched in parallel (step 3). If the tags match, data will be sent to processor and/or lower level memory. Otherwise, the correct data block will be fetched from the DRAM cache (step 4), and then the way ID of the request will be updated in the BWP (step 5) and the corresponding WB will be marked as modified.

Figure 2 (A) shows the latency breakdown of BWP hit request with correct way prediction. It avoids the tag and data serialization timing, resulting in a fast DRAM cache hit. Although accessing the BWP consumes extra cycles, the BWP is a small structure with a small latency overhead. Figure 2 (B) shows the latency breakdown of a BWP hit with an incorrect way prediction. In this case, Compound Access Scheduling adds extra tag access latency on the critical path.

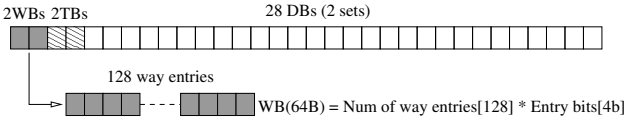

Figure 3. DRAM cache row organization.

Possibility 2: Assuming request *B* misses in the BWP (step 1, dotted), request *B* is sent to DRAM cache (step 2, dotted). The tag block will be brought from the DRAM cache (step 3, dotted). After searching the tag block to find the way ID, the data block will be brought from the DRAM cache and sent to the processor and/or lower level cache (step 4, dotted). Then the corresponding WB of the request will be fetched from the DRAM cache (step 5, dotted). The WB will be allocated in the BWP (step 6, dotted). If the actual way ID of the request differs from the way ID provided by the WB, the way ID of the request will be updated in the BWP and the corresponding WB will be marked as modified. If the evicted WB is marked as modified, the WB must be written back to the DRAM cache (step 7, dotted). For the BWP miss request, the DRAM cache access latency is the access latency of Compound Access Scheduling.

The BWP provides the following advantages:

1) Low on-chip storage overhead. The BWP is a small structure. We restrict the size of the BWP to be no larger than the size of last-level SRAM cache tag array.
2) Low hit latency. Due to its small size, the BWP can easily achieve low access latency. Thus, most requests that hit in the BWP can quickly obtain the way ID. The

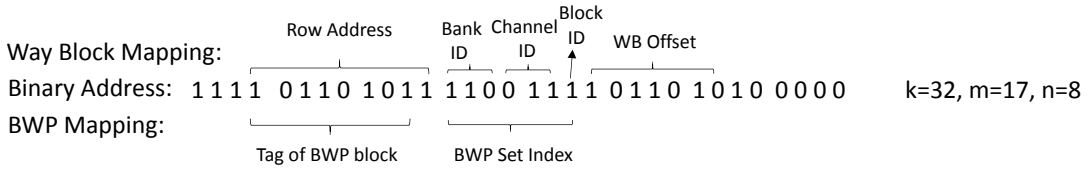

Figure 4. An example of way block mapping scheme and BWP mapping scheme in DRAM cache.

BWP can effectively reduce the overall hit latency by eliminating the discrete tag lookup and data fetching.

3) High DRAM cache hit rate. A set-associative cache with an intelligent cache replacement policy can significantly reduce miss rate when compared to a direct-mapped cache. Equipped with the BWP, a set-associative DRAM cache can achieve a high hit rate together with low latency.
4) Low off-chip traffic. The block-based DRAM cache achieves low off-chip bandwidth overhead compared to page-based DRAM cache. The set-associative cache organization reduces miss rate compared to direct-mapped organization. Way prediction efficiently enables a block-based and set-associative DRAM cache that achieves low off-chip traffic.

A. DRAM Cache Organization

The DRAM cache row organization is shown in Figure 3. Data and tags are co-located in the same DRAM row. Each row also reserves two WBs for storing the way IDs of data blocks. For a 32-block DRAM cache row (2KB), each row allocates 28 data blocks (DB), 2 tag blocks (TB) and 2 WBs. The tag blocks store the full tags and the metadata (e.g. replacement state, valid bit etc.) of the data blocks.

Each DRAM cache set is composed of 14 ways, thus each row co-locates the tags and data for two sets. Each WB in the DRAM cache row has 128 entries where each entry has 4 bits. Thus, each WB stores way information for 2 pages assuming a 4KB page. The value in the 4 bit-entry represents the way ID of the corresponding data block while value '0xE' represents an invalid entry.

Assuming an eight-channel and eight-bank per channel DRAM cache, a k-bit physical address, 64B block size, and totally 2^m WBs in DRAM cache, then the physical address mapping is as follows. Bit[5:0] is the offset within a cache block; Bit[12:6] is the index of way entry within a WB; Bit[13] is mapped to the block ID of DRAM cache; Bit[16:14] is interpreted as the channel ID while Bit[19:17] is the bank ID. Finally, Bit[(m+12):20] is the row address of DRAM cache. Given the DRAM cache capacity of N blocks, two WBs allocated in each DRAM cache row allow the DRAM cache to store way information for $8 \times N$ blocks. Such large coverage ensures that most of the way information can be stored in the DRAM cache.

Figure 4 shows an example of the address mapping scheme of a 32-bit physical address. The way block mapping scheme in DRAM cache is showing above the binary address. Assuming a 128MB DRAM cache capacity, there are 128K way blocks storing in the DRAM cache considering each row stores two WBs. In this case, k=32, m=17. So the way number of this physical address stored in DRAM cache is predicted to reside in channel 3, bank 6, row 363, block 1 and way block offset 90.

B. The BWP

The BWP is used to cache way blocks in the DRAM cache. It adopts a set-associative structure.For each block in BWP, there is a valid bit, T_{valid}, to indicate whether this block is valid. We also add one dirty bit for representing whether this block is modified during its lifetime in the BWP. If the dirty bit is set, the block needs to be written back to DRAM cache when it is evicted.

Each WB stores way IDs for 128 DRAM cache data blocks. Given a k-bit physical address, 2^n sets in the BWP, the physical address Bit[(n+12):12] is the index into the BWP. When 2^m way blocks are stored in the DRAM cache, Bit[(m+12):(n+12)] is used as the tag of a BWP block. Note that the BWP only stores a partial tag of the request. Thus, it is possible that two requests with different physical addresses but the same Bit[(m+12):12] are mapped to the same way block entry. However, this address conflict does not affect correctness as eventually the full tag will be checked before the data can be used. We have found out that allowing multiple requests to share a way entry is beneficial to performance because it reduces the fragmentation problem when using a large way entry vector in the way block.

Figure 4 also illustrates the BWP mapping scheme of the physical address. The BWP mapping of the binary address is showing below the binary address. Assuming a 256-set BWP which is n=8, the BWP set index of the physical address is 231, the tag is 181.

Read Request Operation. If a read request hits in the BWP, the corresponding way block entry provides a way ID. Two requests are issued to the DRAM cache: one for fetching the data according to the way ID and the other for the tag. Note that since the data and tag reside in the same DRAM row, they can burst out back-to-back. Once the tag block is available, if it matches the given block tag, it is a real hit and the data will be sent to the core and/or next closest level of the memory hierarchy. If it is a mismatch, the tags stored in the tag block which belong to the same DRAM cache set

Processors	
Processors	3.2GHz, 8-core CMP, out of order, 128 entry reorder buffer, 4 width issue
Caches	L1 I-cache: private, 32KB, 2-cycle L1 D-cache: private, 32KB, 2-cycle L2: Shared, 8MB, DRRIP, 24-cycle
Off-Chip DRAM	
Bus Frequency	800MHz, DDR3(1.6GHz)
Channels	2
Ranks	1 rank per channel
Banks	8 banks per rank
Row buffer size	4KB
Bus Width	64 bits per channel
tCAS-tRCD-tRP	11-11-11
Stacked DRAM	
Bus Frequency	1600MHz, DDR3(3.2GHz)
Ranks	1 rank per channel
Banks	8 banks per rank
Row buffer size	2KB
Bus Width	128 bits per channel
tCAS-tRCD-tRP	8-8-8

Table I
SYSTEM CONFIGURATION

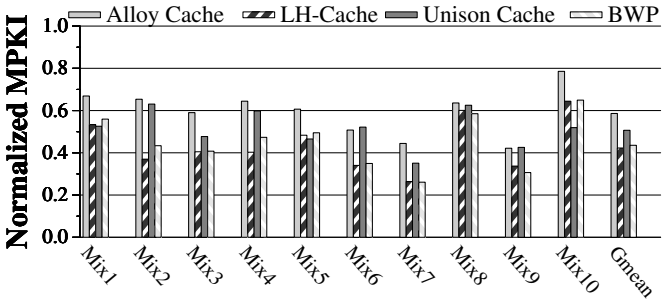

Figure 5. **Reduction in DRAM cache misses for various techniques normalized to no DRAM cache**

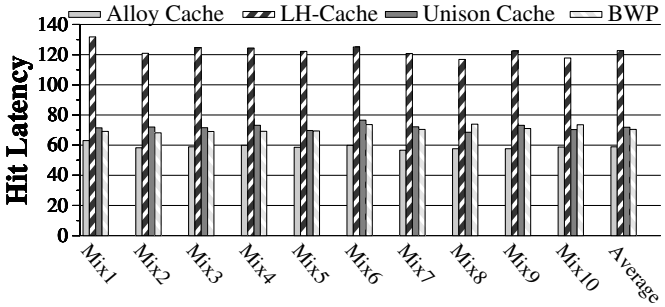

Figure 6. **DRAM cache hit latency for various techniques**

will be searched for a matching tag. If a match is found, the corresponding data will be fetched from the DRAM cache. Otherwise, it is a miss in the DRAM cache, so the request will be serviced by main memory.

In the technique, the memory access predictor (MAP) [18] is used to guide the BWP miss requests. The MAP predicts whether the request will hit in the DRAM cache. If a miss is predicted, the tag block in the DRAM cache will be accessed. At the same time the request will be sent to main memory for service. Once the tag block is obtained, if it is a DRAM cache hit request, data will be fetched from DRAM cache using Compound Access Scheduling. Otherwise, the data will be returned from main memory. If the MAP predicts a hit in the DRAM cache, the request will be serviced by Compound Access Scheduling. The tag block will be accessed first. If a matched tag is found, the data block in the DRAM cache will be accessed. Otherwise, the request will be sent to main memory for service.

Group M	MPKI	Group H	MPKI
gcc	1.86	mcf	56.47
soplex	4.63	lbm	20.04
milc	15.66	GemsFDTD	18.86
omnetpp	11.32	libquantum	25.49
cactusADM	2.78	astar	39.61

Table II
L2 MISSES PER THOUSAND INSTRUCTIONS (MPKI)

Write Request Operation. Since a write request is not on the critical path for the progress of programs, it is treated differently from a read request. The goal of handling write requests is to avoid unnecessary DRAM accesses. The write access to DRAM cache uses Compound Access Scheduling. That is, the tag block in the DRAM row will be accessed first. After finding a full tag match, the DRAM cache data will be overwritten. Otherwise, the data will be written into main memory.

BWP Writeback. A dirty bit is assigned to each BWP block. The dirty bit is set to 1 once the BWP block is updated. The BWP block will be updated in two cases : (1) On a DRAM cache hit, if the way ID stored in the DRAM cache is different from the way ID provided by the BWP, the way ID in the corresponding BWP block entry will be updated; (2) On a DRAM cache miss, the data is brought from the main memory and written into the DRAM cache and the corresponding BWP block entry will be updated.

When a block is evicted from the BWP, if the dirty bit is set, the block must be written back to the DRAM cache row. The writeback address in the DRAM row can be derived by combining the partial tag and the BWP set index of the eviction block.

IV. EVALUATION

In this section, we first describe our evaluation methodology and benchmark for evaluation, and then present the experimental results for our technique, as well as comparison to prior work.

Name	Benchmarks	Group
Mix 1	$8 \times libquantum$	$4 \times H(rate)$
Mix 2	$4 \times (mcf - libquantum)$	$4 \times H$-$4 \times H$
Mix 3	$4 \times (mcf - milc)$	$4 \times H$-$4 \times M$
Mix 4	$2 \times (astar - lbm - libquantum - gcc)$	$2 \times H$-$2 \times H$- $2 \times H$-$2 \times M$
Mix 5	$2 \times (mcf - lbm - GemsFDTD - cactusADM)$	$2 \times H$-$2 \times H$- $2 \times H$-$2 \times M$
Mix 6	$2 \times (mcf - libquantum - GemsFDTD - cactusADM)$	$2 \times H$-$2 \times H$- $2 \times H$-$2 \times M$
Mix 7	$2 \times (mcf - astar - soplex - omnetpp)$	$2 \times H$-$2 \times M$- $2 \times M$-$2 \times M$
Mix8	$2 \times (astar - soplex - omnetpp - cactusADM)$	$2 \times M$-$2 \times M$- $2 \times M$-$2 \times M$
Mix 9	$2 \times (soplex - gcc - astar - omnetpp)$	$2 \times M$-$2 \times M$- $2 \times M$-$2 \times M$
Mix 10	$lbm - mcf - GemsFDTD - gcc - cactusADM - milc - soplex - libquantum$	$H - H - H - H - M - M - M - M$

Table III
MULTI-CORE WORKLOADS

Name	Speedup	Hit Rate	Hit Latency	Miss Latency	Average Latency
No DRAM Cache	0%	0%	n/a	336	336
Alloy Cache	32.5%	39.4%	59	303	207
LH-Cache	38.0%	57.3%	123	230	168
Unison Cache	38.6%	48.7%	72	309	194
BWP	46.8%	54.2%	71	265	159

Table IV
SPEEDUP, HIT RATE, HIT LATENCY, MISS LATENCY AND AVERAGE LATENCY OF TECHNIQUES

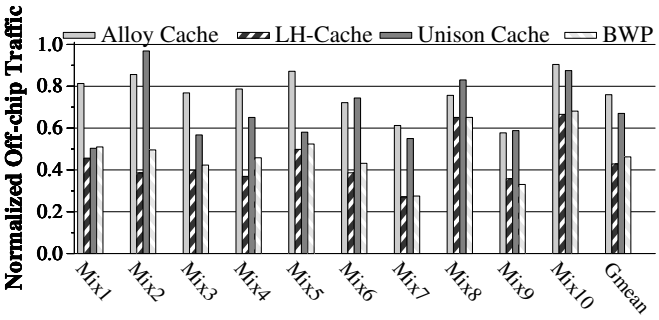

Figure 7. **Memory traffic for various techniques normalized to no DRAM cache**

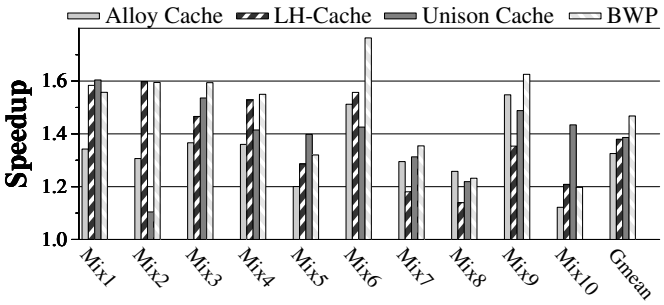

Figure 8. **Performance improvement normalized to no DRAM cache**

A. Methodology

Configuration. We use MARSSx86 [16], a cycle-accurate simulator for the x86_64 instruction set. The DRAMSim2 [19]is integrated into MARSSx86 to simulate the DRAM system. Table I lists the system configuration. We model an eight-core, out-of-order processor. It has two-level SRAM caches with an L3 DRAM cache. We model a per-core L2 cache middle-of-the-road stream prefetcher with 32 streams for each [22]. The prefetcher looks up the stream table at each L2 request for issuing eligible prefetch requests. If the prefetch requests miss in the DRAM cache, they will be serviced by the main memory and allocated in the DRAM cache.

We evaluate five techniques: no DRAM cache, LH-Cache, Alloy Cache, Unison Cache and Buffered Way Predictor (BWP). We use DRRIP [6] cache replacement policy for LH-Cache, Unison Cache and BWP techniques. We perform a detailed evaluation for a 128MB DRAM cache. We also show a DRAM cache size sensitive study ranging from 64MB to 512MB in Section IV-B. We use a 64KB BWP capacity for the 64MB DRAM cache study, 128KB BWP capacity for the 128MB DRAM cache study, and 256KB BWP for 256MB and 512MB DRAM cache study. According to CACTI 6.5 [15], we add 3 cycles access latency for accessing the 64KB and 128KB BWP, and 4 cycles access latency for accessing the 256KB BWP.

We also evaluate the performance and hit rate impact of BWP size ranging from 16KB to 256KB.

Workloads. The SPEC® CPU 2006 [4] benchmarks are used for evaluation. Of the 29 benchmarks, 22 can be compiled and run correctly with our infrastructure. For each benchmark, we run 400 million instructions from a typical phase identified by SimPoint [20]. Then we classify the

benchmarks into two groups: Group H (High intensity) and Group M (Mid intensity). If the L2 Misses Per 1000 Instructions (MPKI) of the benchmark is greater than 20, the benchmark is classified in Group H. Benchmarks with MPKI greater than 1 are classified in Group M. Table II lists the L2 MPKI for each of those benchmarks.

The eight-core workloads are selected as shown in Table III. For each mix, we run the experiment with 4 billion instructions total for all eight cores starting from the typical phase. Each benchmark runs simultaneously with others.

B. Experimental Results

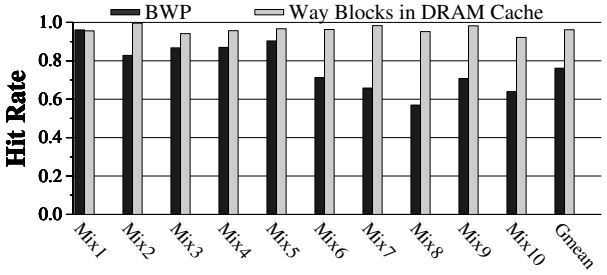

Figure 9. **The hit rates of BWP and way blocks in DRAM cache**

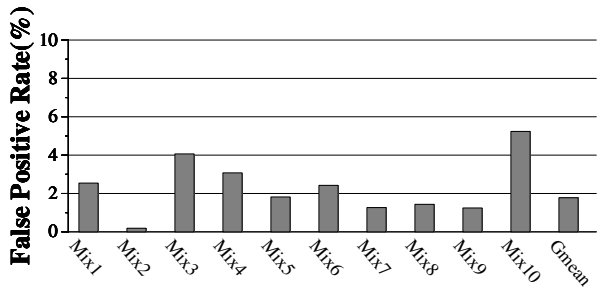

Figure 10. **The false positive rate of BWP**

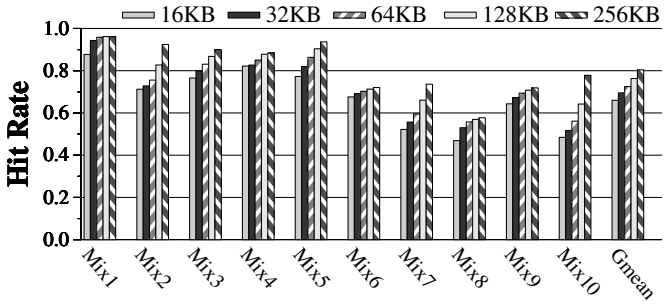

Figure 11. **Hit rate sensitivity across various BWP sizes**

DRAM Cache Read Misses. Figure 5 shows the DRAM cache read misses normalized to all the DRAM cache read requests for various techniques. The Alloy Cache is a direct-mapped design. The average normalized misses of Alloy Cache is 58.6%. The LH-cache has 30-way set associativity in our implementation. It achieves 42.3% normalized misses. In the Unison Cache, each row co-locates four large cache lines consisting of one cache set. A footprint predictor is used to predict the data that will be accessed soon within a large cacheline and prefetch them into the DRAM cache. The footprint predictor is a PC based predictor. It uses the PC and the first block offset within a large cacheline to index the access pattern. However, the prefetch requests issued by L2 cache do not have PC information which can not be used to recognize the access pattern. Moreover, the originally proposed Unison Cache technique does not enable a last-level SRAM cache prefetcher. In the presence of the L2 prefetcher that prefetches data into both last-level SRAM cache and DRAM cache, the benefit of the Unison Cache technique are diminished. The average normalized misses of Unison Cache is 50.6% of baseline. The BWP technique uses a 14-way set associativity. The normalized misses of the BWP technique are 43.6% of baseline.

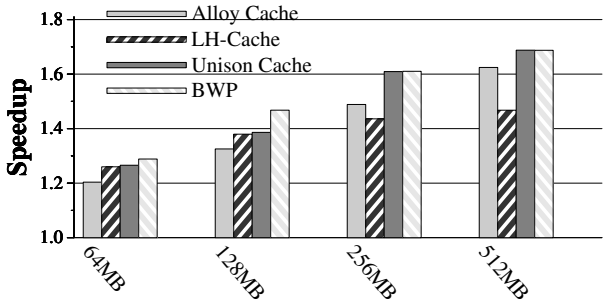

Figure 12. **Performance sensitivity across various BWP sizes (normalized to no DRAM cache)**

DRAM Cache Hit Latency.

Figure 6 shows DRAM cache hit latency for various techniques. The Alloy Cache technique streams out tags and data together, achieving the lowest hit latency which is 59 CPU cycles on average. The LH-cache serializes the tag and data accesses, adding extra tag access latency compared to Alloy Cache. Accessing the MissMap structure in the LH-cache also takes an extra 24 cycles. The average hit latency of DRAM cache in LH-cache is 123 CPU cycles. Both Unison Cache and BWP techniques predict the way ID of the request. For the correct way prediction, the tag and data of the request can be accessed in parallel. For an incorrect way prediction, the tag and data are accessed in serial. The Unison Cache technique uses a 4K-entry region address-based way predictor for a 4-way set associativity DRAM cache. The average access latency of Unison Cache is 72 cycles. However, the region address-based way predictor does not work well for high associativity block-based cache. The BWP technique speculates the way IDs of requests for high associativity block-based cache with high accuracy. The average hit latency for BWP technique is 71 cycles.

Off-chip Traffic. We evaluate the off-chip traffic and access latency for various techniques. Figure 7 shows the off-chip traffic evaluation results for various techniques normalized to No DRAM Cache. The LH-Cache achieves lowest DRAM cache miss rate which results in lowest off-chip traffic that is 42.9% of baseline. The normalized off-chip traffic of BWP technique is 46.2% than no DRAM Cache. The Alloy Cache technique has higher off-chip traffic than other techniques due to the high DRAM cache miss rate. Reducing the off-chip memory traffic can reduce the main memory access latency.

Performance Evaluation. Figure 8 shows the speedup for various techniques normalized to no DRAM cache. The Alloy Cache delivers a geometric mean speedup of 32.5%. LH-Cache improves the performance by 38.0% on average. Although the average hit latency of Alloy Cache is lower than the LH-Cache, the LH-Cache achieves higher hit rate and lower off-chip access latency compared to Alloy Cache. Thus LH-Cache performs better than Alloy Cache. The average speedup of Unison Cache is 38.6%. The BWP technique balances the hit latency and miss rate for an overall performance benefit. It yields a geometric mean speedup of 46.8%.

Performance Impact of Hit Rate and Access Latency. Table IV gives the average speedup, DRAM cache hit rate, DRAM cache hit latency, DRAM cache miss latency and average access latency of various techniques. The average access latency is calculated as:

$$\begin{aligned} AverageAccessLatency = HitRatio \times HitLatency \\ + MissRatio \times MissLatency \end{aligned} \quad (1)$$

In general, the lower average access latency results in better performance. Although LH-Cache has lower average access latency than Unison Cache, the speedup of Unison Cache is higher than LH-Cache because it takes 2MB L2 cache capacity to store the MissMap structure. So LH-Cache has a higher L2 cache miss rate.

BWP and Way Blocks Hit Rate. We evaluate the BWP and way block hit rate. If the way ID of the request can be found in the BWP, it is a BWP hit. If the way ID of the request can be found in the way block in the DRAM cache row, it is a way block hit. Figure 9 shows the BWP and way block hit rate in our technique. The 128K BWP achieves 76.2% BWP hit rate. Allocating two way blocks in the 32-block DRAM cache row yields 96.2% way block hit rate.

We also evaluate the false-positive rate of BWP. A false positive occurs when the BWP provides an incorrect way ID. False positives are harmful since tag and data accesses of the request will be serviced in serial. Figure 10 shows the false-positive rate of BWP. The average false-positive rate of BWP is only 1.8%. This low false-positive rate does not cause significant performance and bandwidth degradation.

BWP Size Sensitivity Study. We evaluate the impact of the size of the BWP on its hit rate. Figure 11 shows the hit rate sensitivity across various BWP size ranges from 32KB to 256KB. Doubling the BWP size from 128KB to 256KB, the cache hit rate slightly increases from 76.3% to 80.4%. This result indicates that most of the BWP hits benefit from spatial locality rather than the temporal locality of the requests. To improve the BWP hit rate, it is more helpful to intelligently explore the spatial locality of requests.

DRAM Cache Size Sensitivity Study. Figure 12 shows the average performance evaluation results for various techniques with the DRAM cache size ranges from 64MB to 512MB. For a smaller capacity DRAM cache (*e.g.*, 64MB) with a higher miss rate, the DRAM cache design benefit more from improving DRAM cache hit rate. The BWP technique improves DRAM cache hit rate by enabling high associativity DRAM cache design. For a larger capacity DRAM cache (*e.g.*, 512MB) which has lower miss rate, reducing the DRAM cache hit latency is critical to improve the DRAM cache efficiency. The BWP technique optimizes the DRAM cache hit latency by reducing the tag access latency.

V. Conclusion

In this work, we propose the Buffer Way Predictor (BWP) for set associative die-stacked DRAM caches. The way IDs of DRAM cache data blocks are stored in the DRAM cache. A small capacity BWP structure is proposed to cache the way IDs of DRAM cache data blocks. For each DRAM cache hit request that hits in the BWP, the BWP speculates the way ID of the request. Thus, the tag and data of the DRAM cache request can be accessed in parallel. Each way block in the BWP stores way IDs for 128 DRAM cache data blocks with high spatial locality which achieves high BWP hit rate. Our proposed technique can achieve low DRAM cache hit latency, high DRAM cache hit rate, and low off-chip traffic.

Acknowledgment

Daniel A. Jiménez is supported by NSF grant CCF-1332598. Yuan Xie was supported in part by DOE DE-SC0013553 and NSF 1500848. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. SPEC is a registered trademark of the Standard Performance Evaluation Corporation (SPEC).

References

[1] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die stacking (3d) microarchitecture," in *Proceedings of the 39th Annual*

IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 469–479. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2006.18

[2] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, Nov 2010, pp. 1–11.

[3] S. Franey and M. Lipasti, "Tag tables," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 514–525.

[4] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, September 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[5] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 2014 International Conference on Parallel Architectures and Compiler Techniques (PACT)*, September 2014.

[6] A. Jaleel, K. Theobald, S. S. Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA-37)*, June 2010.

[7] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 47. Cambridge, UK: IEEE Computer Society, 2014.

[8] 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 *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 404–415. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485957

[9] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Integrating dram caches for cmp server platforms," *IEEE Micro*, vol. 31, no. 1, pp. 99–108, 2011.

[10] J. Kim and Y. Kim, "HBM: Memory solution for bandwidth-hungry processors," in *HotChips 26*, 2014.

[11] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (HBM) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 432–433.

[12] G. H. Loh, "Extending the effectiveness of 3d-stacked dram caches with an adaptive multi-queue policy," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/1669112.1669139

[13] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 454–464. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155673

[14] G. Loh, "3d-stacked memory architectures for multi-core processors," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, June 2008, pp. 453–464.

[15] N. Muralimanohar and R. Balasubramonian, "Cacti 6.0: A tool to model large caches."

[16] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," in *Proceedings of the 2011 Design Automation Conference*, June 2011.

[17] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *HotChips 23*, 2011.

[18] 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 *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 235–246. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.30

[19] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. PP, no. 99, p. 1, 2011.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[21] 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 *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 247–257. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.31

[22] S. Srinath, O. Mutlu, H. Kim, and Y. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, 2007, pp. 63–74.

[23] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, "An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth," in *High Performance Computer Architecture, 2010. HPCA 2010. IEEE 16th International Symposium on*, 2010, pp. 429–440.