IEEE TRANSACTIONS ON
GAMES
A PUBLICATION OF THE IEEE COMPUTATIONAL INTELLIGENCE SOCIETY, THE IEEE SENSORS COUNCIL, AND THE IEEE CONSUMER ELECTRONICS SOCIETY
http://cis.ieee.org/ieee-transactions-on-games.html

# Optimizing DRAM Cache by a Trade-off between Hit Rate and Hit Latency

Pai Chen, Jianhui Yue, Xiaofei Liao, and Hai Jin

**Abstract**—Due to the large storage capacity, high bandwidth and low latency, 3D DRAM is proposed to be the last level cache, referred to as DRAM cache. The hit rate and hit latency are two conflicting optimization goals for DRAM cache. To address this issue, we design a new DRAM organization that trades the lower hit rate for shorter hit latency by way-locator cache and novel cache set layout. We have designed a novel DRAM cache organization to simultaneously achieve a good hit rate and shorter latency, referred to as SODA-cache. The SODA-cache adapts 2-way set associate cache motivated by the observation that 2-way set associative cache provides the most hit rate improvement from the direct-mapped cache to highly associative cache. The proposed way-locator cache and a novel set layout effectively reduce the cache-hit latency. We use SPEC 2006 CPU benchmark to evaluate our design and two stat-of-art DRAM cache designs. Experimental results show that SODA-cache can improve hit rate by 8.1% compared with Alloy-cache and reduce average access latency by 23.1%, 13.2% and 8.6% compared with LH-cache, Alloy-cache and ATCache respectively on average. Accordingly, SODA-cache outperforms over LH-cache, Alloy-cache and ATCache by on average 17%, 12.8% and 8.4% respectively in term of weighted speedup.

**Index Terms**—DRAM Cache, 3D Stacked DRAM, Cache, Memory Hierarchy, Computer Architecture.

## 1 Introduction

The 3D die-stacking DRAM provides high bandwidth, low latency and large capacity, mitigating memory wall. Due to its gigascale capacity, the 3D die-stacking DRAM could not replace off-chip DRAM and has been proposed to be architected as the last level cache, referred to as DRAM cache [1], [2], [3]. The DRAM caches tag storage overhead caused by the large capacity of 3D DRAM makes it challenging to design high performance DRAM cache. For example, 256 MB DRAM caches tag lines storage overhead is about 12MB with 64B cache line and it is less beneficial to place 12MB tags to the fast SRAM than storing cache lines in 12MB SRAM cache. In order to address this issue, prior research work proposes two solutions: 1) reducing tag storage overhead with large granularity of cache line and 2) storing tags in DRAM cache with small granularity of cache line. Storing tags in different storage devices, they are referred to as tag-in-SRAM DRAM cache and tag-in-DRAM DRAM cache respectively. However, they still have limitations. With tag-in-SRAM, design, large size cache line suffers from large bandwidth overhead and under-utilization of space. With tag-in-DRAM, co-locating data lines and tag lines in DRAM cache serializes the tag access and data accesses, increasing access latency. This paper focuses on the DRAM cache design with tag-in-DRAM.

With tag-in-DRAM, hit rate and cache hit latency are two major optimization goals in DRAM cache design. Loh et al. [1], [2] pioneers the research of set-associative DRAM cache to optimize hit rate at the cost of high hit latency incurred by tag-then-data access serialization [4]. Even though caching tag in on-chip SRAM [5], [6] has been proposed to speed up the tag lookup, the tag-then-data access serialization cant be completely removed from data access path, resulting suboptimal performance. To minimize hit latency, Alloy Cache [3] has been proposed to organize DRAM cache as direct-mapped cache and access data and tag simultaneously, sacrificing hit rate. These two-extreme tag-in-DRAM designs optimize one optimization goal at the cost of the other goal, failing to achieve the optimal performance.

Its a challenging to optimize hit rate and hit latency simultaneously. The high set associativity brings the higher hit rate and requires more tag lines stored in DRAM. On serving a request, the DRAM cache controller needs to retrieve all tag lines belonged to a set by issuing a CAS DRAM command and multiple bus bursts before determining the location of the requested data line. This tag access latency increases the hit latency. In order to minimize hit latency, the Alloy-cache merges a data line with its tag in tag-and-data unit(TAD) and performs tag look up by issuing a CAS command to stream out a TAD. TAD restricts cache organization to be direct-mapped cache and suffers from lower hit rate. Hit rate and hit latency are two conflicting design goals in DRAM cache.

In this paper, we propose a design to Simultaneously Optimizing hit rate and latency for DRAM cache, referred to as SODA-cache. We make an observation on the relationship between hit rate and set associativity, suggesting that high associative degree brings negligible hit rate improvement and the 2-way associativity provides the biggest increase of hit rate. Motivated by this observation, we propose to organize DRAM cache as 2-way set associative cache, with

• *P. Chen, X. Liao and H. Jin are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China.*

• *J. Yue with the department of Computer Science, Michigan Technological University,Houghton, Michigan. E-mail: jyue@mtu.edu*

*Manuscript received Dec. 7, 2017.*

Digital Object Identifier 10.1109/TETC.2018.2800721

good hit rate and reducing tag space overhead. The smaller tag space can translate to less time to access tags. In order to reduce hit latency, we propose the WayLocator cache in SRAM to quickly determine the requested data location in DRAM cache without accessing tag in DRAM cache. To further decrease hit latency, we propose a novel set layout and issue one CSA command to stream out data and tags together.

The main contributions are summarized as follows.

- We observe that 2-way set associative cache provides the largest improvement of hit rate with associativity varying from direct-mapped to 58-way. This observation motivated us to choose 2-way associativity to organize DRAM cache, with good hit rate.
- We propose the way locator cache stored in SRAM to directly stream the requested tag and data together from DRAM cache, without incurring tag-then-data sterilization. To further decrease the hit latency, a novel set layout is proposed that one CAS DRAM command is required to retrieve tags and data.
- Our SODA-Cache design effectively exploits the best aspects of LH-Cache and Ally-Cache avoiding their limitations and achieves the optimal average access latency and weighted IPC.
- We evaluate our proposed SODA-Cache by simulating an 8-core system under SPEC CPU 2006 benchmark suite. Experimental results under 10 multi-programmed workloads show that SODA-Cache can improve hit rate by 8.1% compared with Alloy-cache and reduce average access latency by 23.1%, 13.2% and 8.6% compared with LH-cache, Alloy-cache and ATCache respectively on average. Accordingly, SODA-cache outperforms over LH-cache, Alloy-cache and ATCache by on average 17%, 12.8% and 8.4% respectively in term of weighted speedup.

The rest of this paper is organized as follows. Backgrounds and motivation are given in Section 2. Section 3 introduces our design. Experimental methodology and results are given in Section 4. Related works are summarized in Section 5. Section 6 concludes this paper.

## 2 Background and Motivation

Die-stacked DRAM's high bandwidth and low access latency have great potential to mitigates the memory wall issue. The relatively small storage capacity of Die-stacked DRAM makes it be architected as the cache, referred to as DRAM cache, which is transparent to the software. There exit two classes of DRAM cache designs: block-based DRAM caches and page-based DRAM caches. While the cache block size of a block- based DRAM is same as the last level SRAM cache, the size of cache block in a page-based DRAM cache is 2KB or 4KB and is not larger than DRAM page size. The larger granularity of page-based DRAM cache requires smaller tag storage space, which makes tags fit in the SRAM. It also brings higher hit rate by exploiting the spatial locality. However, the larger capacity of Die-stacked DRAM makes SRAM-based tag storage less salable. In addition, the fetching page-sized cache block not only wastes bandwidth but also results in under-utilization of DRAM cache capacity. Therefore, this paper focuses on the block-based DRAM cache design. We are going to briefly discuss two block-based DRAM cache designs.

### 2.1 Hit Rate Optimization DRAM Cache

Similar to the conventional SRAM cache, block-based DRAM cache has tag and data for each data block. Since the large DRAM cache makes it impractical to accommodate the correspondingly large tag in SRAM, LH-cache proposes to store the tag and data in the DRAM cache. LH-cache architects DRAM cache as a highly-associated cache by storing tag and data of a set in one DRAM row. For example, 58 data lines and tag are placed in the one DRAM row with 4KB, being a 58-way cache. As shown in Figure 1, tag and data lines reside in beginning of a row and the remaining of the row respectively. This high association degree can reduce the conflict misses and benefit system performance.

To service a request, the DRAM cache controller checks the tag and then reads the data line according to the outcome of tag query. In order to reduce the access latency, the compound access scheduling algorithm issues activation and read command to read tag and then reads the data line before servicing the other request to the same bank, without re-activating the row. Further to reduce the access latency, the missMap is proposed to direct the DRAM cache miss requests to off-chip DRAM, without wasting time to look up tag in the DRAM cache. Even the LH-cache effectively solves the tag store issue and mitigates the DRAM cache access latency, its hit latency is large and degrades performance.

Serving a request involves one activation command, two CAS commands with I/O bursts for tags and a data line, which makes the hit latency high. In addition, high associativity requires reading a large number of tag lines and worsen the hit latency. We will show that the over high associative degree design is not necessary for the optimal performance.

### 2.2 Hit Latency Optimization DRAM Cache

In LH-cache, the aforementioned serialization of tag access and data line access increases the hit latency in DRAM cache, resulting in suboptimal performance. In order to address this issue, the Alloy Cache proposes to trade the low hit latency for the low hit rate. The Alloy cache organizes the DRAM cache as a direct-mapped cache and combines a data line with its tag line, referred to as tag-and-data units(TAD). This merging of data line and tag line removes the searching correct way from the data access path and directly accesses the TAD to avoid the serialization of tag and data access. In case of cache miss, DRAM cache is accessed before the off-chip memory, increasing the access latency. Further to reduce the access latency, Alloy Cache issues the commands to access DRAM cache and off-chip cache simultaneously if the requested data is predicated to be cache miss exploiting either memory access history or instruction causing memory access. While Alloy Cache efficiently reduces the DRAM cache hit latency, it suffers from the low hit rate because of the direct-mapped cache organization.

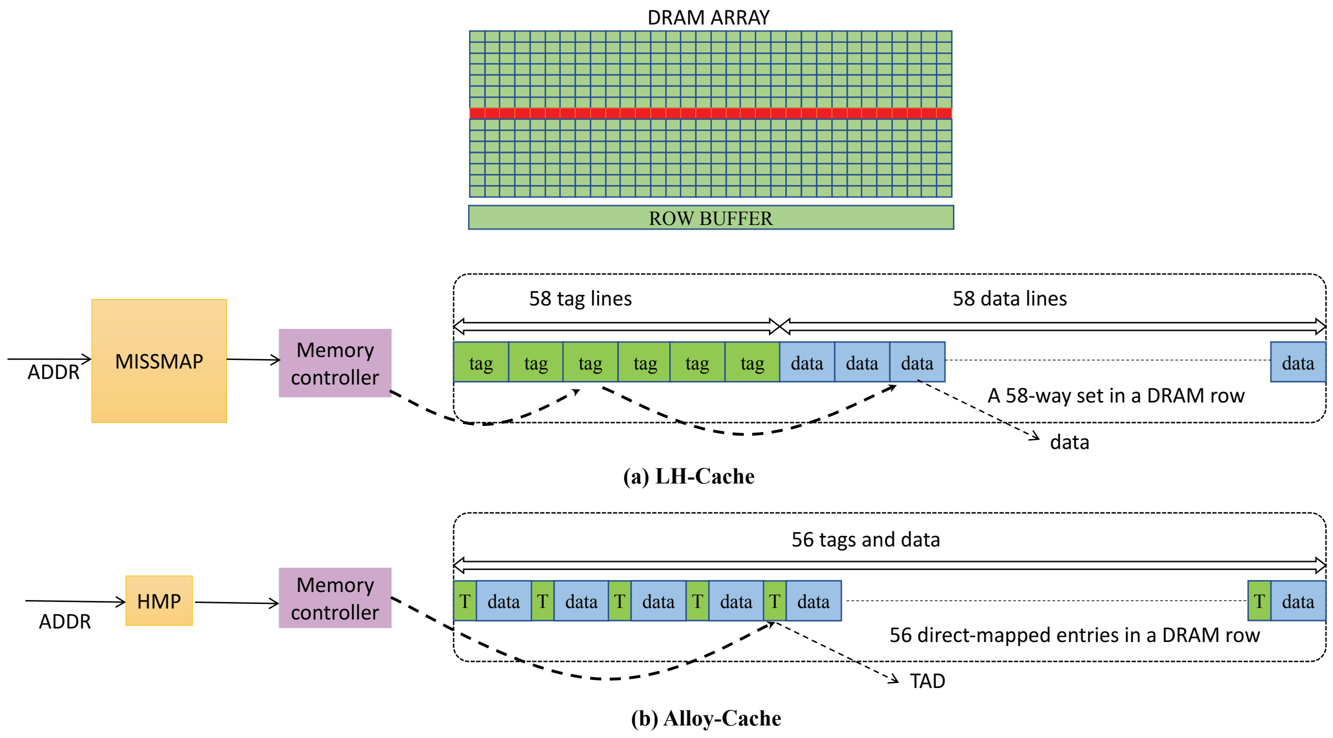


Fig. 1: Block-based DRAM cache organizations

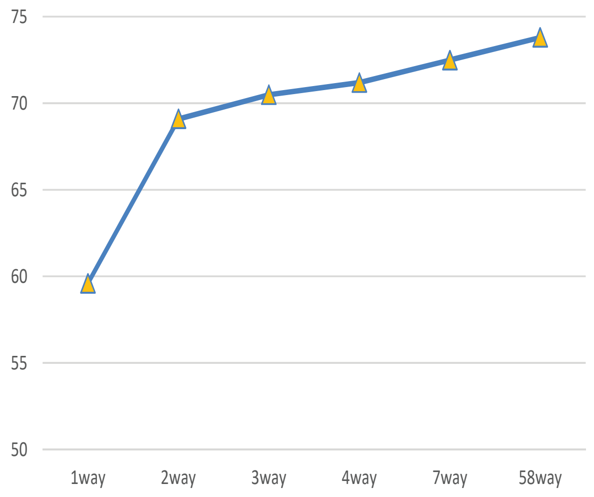


Fig. 2: Hit Rate and Associativty

## 2.3 Motivation

The LH Cache and Alloy Cache take two extreme approaches to organize DRAM cache with high associative cache and direct-mapped cache respectively. In reality, one DRAM cache row can accommodate multiple sets with some number of ways. Assuming each row is 4KB and data line is 64B, a row can implement one set with 58 ways, two sets with 29 ways, three sets with 19 ways, 4 sets with 14 ways and so on, until 58 sets with 1 way. Note that some cache organizations waste row storage capacity. For example, a row only can provide 11 sets of 5-way cache and 9 sets of 6 ways cache, wasting 4 and 5 data lines storage spaces respectively. In order to fairly compare different cache organizations, the above two cache organizations with low space utilization are ruled out. The Fig. 2 shows the how the cache organization impact the DRAM cache hit rate(experiment details will be discussed in the Section 4). The Fig. 2 indicates that the low associative cache can significantly improve hit rate compared with direct-mapped cache, achieving the maximal performance gain and there is relative smaller hit rate benefit when the associative degree become larger. For example, when the associativity is changed from one to two, the hit rate is increased by 9.5%, and 2-way set-associative cache is only 4.7% worse than 58-way set-associative cache. Therefore, the 2-way set associative cache organization achieves the most benefit in terms of hit rate variation.

The associative degree also affects the tag lookup latency in DRAM cache, which is in the critical path of data access. The higher associative degree cache has more tag lines and incurs more memory cycles to stream out the tag stores from DRAM cache before reading the requested data line stored in DRAM cache. For example, 58-way cache has six tag lines with 64B taking 12 memory cycles to stream out and 29-way cache needs to burst three tag lines with 64B incurring 6 memory cycles to burst. When it comes to the 2-way cache, its tag streaming latency is 1 memory cycle, with the smallest tag streaming latency.

An ideal DRAM cache design should achieve higher hit rate and lower access latency. However, these goals are conflicting. For example, while the LH Cache achieves higher hit rate but has longer access latency caused by the tag and data access serialization, Alloy Cache realizes lower access latency at the cost of lower hit rate. So, an efficient DRAM cache design could make a good tradeoff between the access latency and hit rate. Motivated by the above two observations, we believe that 2-way DRAM cache organization is a good design tradeoff between the hit rate and access latency, achieving sufficiently high hit rate and the closest to the minimal access latency. We will discuss the efficient 2-way DRAM design at the Section 3.

## 3 Design

### 3.1 Design Overview

The main idea of SODA Cache is that the data line is directly retrieved from DRAM cache together with tags by exploiting the proposed WayLocator cache and a novel set layout. As shown in Fig. 3, the address of incoming request is consulted with WayLocator cache and is responded with its corresponding way identification(ID) on WayLocator cache hit. In addition, we use the memory access predicators(MAP)to predicate the presence of the request with its address. In the case of both WayLocator cache hit and a predication of presence, the way ID provided by WayLocator is used to stream the requested data line and tags. In this way, we can break tag-then-data sterilization, reducing hit latency. On WayLocator cache miss, all data lines and tags per set are needed to be streamed out from 3D DRAM. If the request is predicated to be DRAM cache miss, both off-chip DRAM and 3D DRAM are accessed.

Since we choose 2-way set associative cache organization, there are 29 sets in a row of 3D DRAM and one set has two data lines and two tags. On accessing one request, two tags and the data line indicated by the way ID, which is the response of WayLocator cache, are read with a single CAS command and a number of bursts, reducing hit latency. Note that our proposed set layout can facilitate streaming two tags with one CAS command. Tags are needed to be checked before returning data line to processor. In case tag is mismatched, off-chip DRAM access is involved, occurring with small possibility.

### 3.2 WayLocator Cache

As discussed in the Section 2, tag lines stored in DRAM cache are probed before the data line is streamed out from the DRAM row with tags indication of the target data line location at the DRAM row. This serialization of tag and data access is one part of hit latency overhead.

Tag-in-SRAM is one option to avoid tag-then-data serialization. However, the large tag storage overhead excludes the possibility of tag-store in SRAM. A traditional cache tag line has two major pieces of information: the location of data in a set and data residency, with 6 bytes storage overhead [1]. The location of data indicates the column address of data stored in the DRAM row. In the LH-Cache, the DRAM cache controller first streams tag lines out and extracts the data location from them to read the data line, increasing hit latency.

In order to optimize the hit latency, we propose to store data location information in the SRAM. In case of 2-way set associative cache, the storage overhead of a way number is one bit. With the way ID, the DRAM cache can locate the requested data line in DRAM cache before reading it out. Such relative smaller storage requirement of data location makes it possible to put way ID information in SRAM.

Similar to the missMap, we organize data location information in SRAM, which is referred to as WayLocator cache. As show in Fig. 4, each WayLocator cache entry tracks data lines in a contiguous memory region in off-chip DRAM, such as a page, and an entry includes a tag for the corresponding region address and a location bit vector with one bit representing the location of a data line. The location bit with the value of 0 indicates the corresponding data line is located at the 0 way. Otherwise, the data line resides in 1 way. Due to the limited storage budget, the WayLocator cache organizes its entries with a set-associative cache. With WayLocator cache, we can directly retrieve the requested data line from the row, reducing access latency. Note that data residence information is retrieved together with data line and DRAM cache controller still checks the residence to guarantee the correctness.

Representing a data line, a location bit only specifies the location of the corresponding data line and cannot indicate the data residence. Accordingly, it is a challenge for WayLocator cache to sever the purpose of data residency. Although data residency can be queried by the tag lines stored in the DRAM row, the query latency incurred by the tag access hurts performance. In order to address this issue, we propose to integrate the memory access predicators(MAP) with the WayLocator cache shown in Fig. 3. In previous research work, MAPs were proposed to predicate whether the data line is DRAM cache. The cache controller sends the memory access to DRAM cache and off-chip memory simultaneously if the memory access is predicated to be DRAM cache miss. This parallel memory access model can eliminate the serialization of the cache miss checking latency from the memory access path. Besides access latency reduction, the MAP can quickly facilitate cache-miss detection for WayLocator cache, without accessing tag line stored in DRAM cache.

For a WayLocator entry, each location bit is initialized to be 0. When a cache line is installed, the location bit is set to indicate its location in DRAM cache. For example, setting the bit to be 1 means the cache line is located at the 1-way. If MAP makes a misprediction of cache-hit and the location bit is initialized to be 0 , 0-way data and all tags lines are read and tags checking can detect the address tag mismatch, without reading wrong data. So WayLocator and MAP can together ensure the correctness of responded data.

The SODA cache controller consults both MAP and WayLocator cache, then takes different actions to stream out a data line from DRAM cache as show in Fig.3. The WayLocator cache produces either WayLocator-hit or WayLocator-miss. There are four combinations of outcome of MAP and WayLocator cache sketched as follows:

- **Predicated to be cache-hit and WayLocator cache hit:** The controller fetches the data line according to the way ID provided by WayLocator entry.
- **Predicated to be cache-hit and WayLocator cache miss:** All ways are retrieved upon WayLocator cache-miss.
- **Predicated to be cache-miss and WayLocator cache hit:** The controller streams the data line according to the way ID indicated by the WayLocator entry.
- **Predicated to be cache-miss and WayLocator cache**

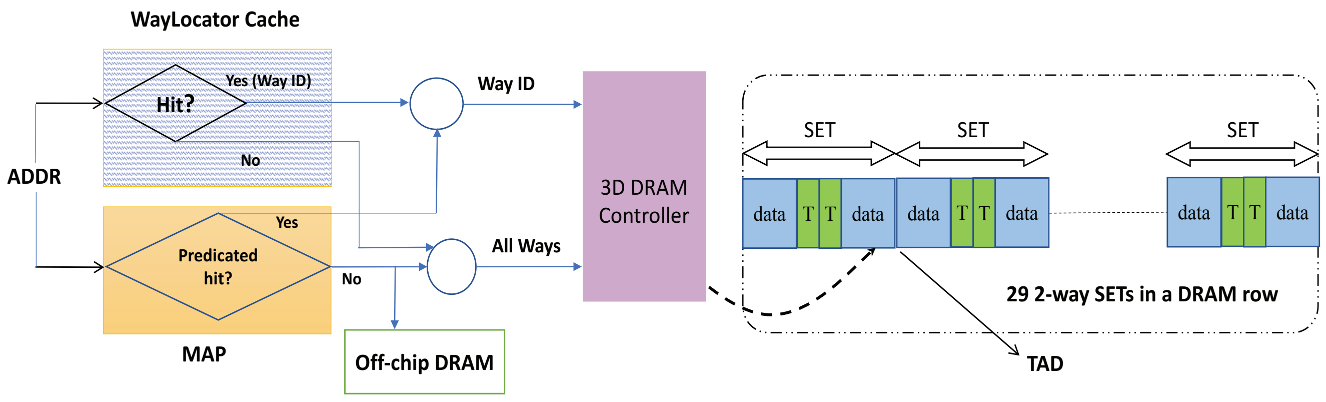


Fig. 3: SODA Cache Achitecture

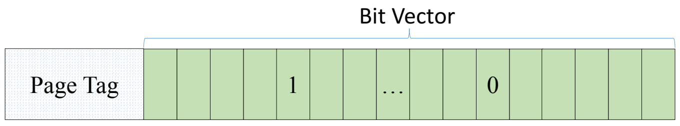


Fig. 4: A WayLocator Cache Entry

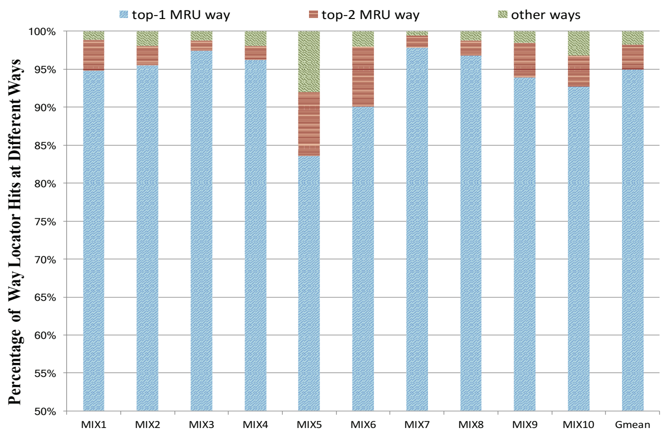


Fig. 5: Temporal Locality of Way Locator Cache

**miss:** The controller reads data from the off-chip cache and streams all ways simultaneously.

The efficient MAP design incurs negligible latency. The WayLocator cache provides the data line location without responding to residency query.

The large storage capacity of DRAM cache poses a challenge for the WayLocator implementation, being realized as set associative cache. Assuming that physical address and page size are 48 bits and 4KB respectively, each WayLocator entry has 36 bits (page tag) + 64 bits (bit vector) = 12.5 bytes. Further, our experiments show that the 16-way set associative cache with 8000 sets provides good coverage for the 256MB DRAM cache. This WayLocator organization requires 8000*16*12.5 B = 1.52 MB, making it impractical to deploy WayLocator in SRAM. The Giga-scale DRAM cache worsens the storage issue for the WayLocator.

In order to address the storage issue, we propose to store WayLocators the most recently used(MRU) entry of one set in SRAM and leave the remaining entries in the DRAM cache. Prior research work show that DRAM cache accesses exhibit high spatial locality. The large contiguous memory region, such as a page of 4KB, associated with an entry can benefit from this access pattern. The Figure5 shows the percentages of DRAM access traffic served by the MRU entry, MRU-2 entries which including MRU entry and the entry next to the MRU and reminding entries under different workloads. As you can see, MRU entry absorbs 93.7% DRAM cache accesses on average and MRU-2 entries serve 97% DRAM cache accesses. Our experimental results show the MRU entry absorbs 93.7% DRAM cache accesses and MRU-2 entries serve 97% DRAM cache accesses on average. Therefore, MRU entries in the SRAM can obtain the data line location in DRAM cache with low latency for 93.7% requests and only 6.3% requests involve with DRAM cache to get data line location. In other words, WayLocator caches hit rate can reach 93.7% with MRU entry. The MRU storage overhead is 1/16 of the original one and SRAM space overhead is 1.52 MB/16 = 97.6 KB in the aforementioned example, which is acceptable in most designs. In summary, we exploit the high spatial locality in a page and significantly reduce WayLocator storage overhead, without sacrificing performance much.

## 3.3 Set Layout in DRAM Cache

We introduce a novel set layout in DRAM cache to reduce tag access latency. Although the WayLocator has the way identification number, the residency examination is still facilitated by the address tag stored in the DRAM cache where a set consists of tag stores and data lines. In the LH-Cache, the tag stores are followed by data lines for a set stored in DRAM cache and a controller issues an extra DRAM command to fetch tags before reading a data line. Unlike the set layout of LH-Cache, we propose to store tag lines between data lines as shown Fig.6. Specifically, the way-0s data line is followed by its tag and the way-1s tag is followed by its data line.

This novel set layout can reduce the access latency. For example, assuming a processor is going to access the way-0, the controller issues one CAS command and a number of I/O bursts to stream both the data line and two tags, and

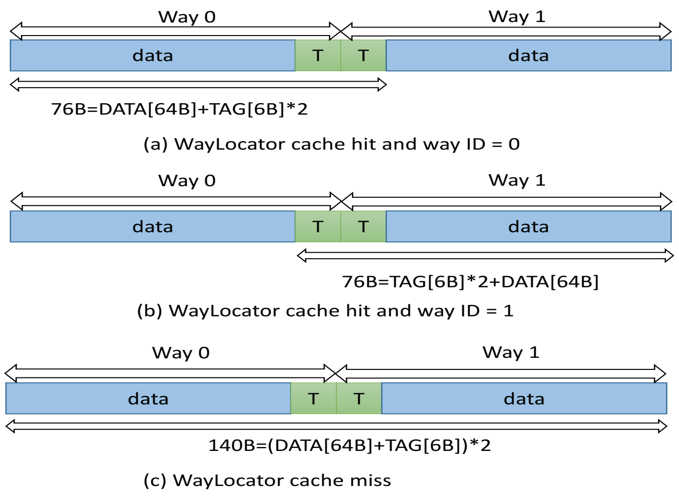

Way 0
Way 1
data
T
T
data
76B=DATA[64B]+TAG[6B]*2
(a) WayLocator cache hit and way ID = 0
76B=TAG[6B]*2+DATA[64B]
(b) WayLocator cache hit and way ID = 1
140B=(DATA[64B]+TAG[6B])*2
(c) WayLocator cache miss

Fig. 6: Set Layout

updates LRU counters in these two tags shown in Fig. 6(a). When the way-1 is to be accessed, the Fig. 6(b) shows that its data line and two tags are streamed out with one CAS command and a number of I/O bursts, updating LRU counters in tags. In case of WayLocator cache miss, the controller does not know which way to be accessed and therefore all tags and data lines are fetched, incurring extra overhead indicated in Fig. 6(c). However, our experimental results show WayLocator cache hit rate is 93.7% on average and the WayLocator cache misses negligibly impact performance.

The Fig.7 compares SODA-Cache with LH-Cache and Alloy-Cache, and demonstrate its performance advantage. LH-Cache access includes the large amounts of tag bursts caused by the high associativity and two CAS commands required by reading the tags and data line sequentially, leading to the worst access latency. The Alloy-Cache issues a CAS and a smaller number of bursts to stream a TAD out, achieving the best latency. In the case of WayLocator cache hit, one data and tag are read out, which is same as Alloy-Cache. On SODA-Cache miss, extra bursts are needed to retrieve an extra data line. Since the WayLocator cache hit rate is as high as 93.7%, our SODA-Cache average access latency is very close to Alloy-Cache and better than LH-Cache. The Fig.7 does not show the time overheads for the WayLocator cache and MAP since the CACTI [7] estimates the WayLoator cache latency to be 5 CPU cycles and MAP latency is reported to be 1 CPU cycle [1], and they are done when being queued. Note that processor compares the fetched tag line against the address tag and proceeds to access off-chip DRAM if a tag mismatch is found.

## 4 Performance Evaluation

We evaluate our deign by using a Pin-based [8] x86 simulator to model processor cores and cache, with an enhanced cycle-accurate NVMain [9] to simulate DRAM Cache and off-chip main memory. Table 1 shows the parameters of simulated processor, DRAM cache and off-chip main memory.

We construct 10 multi-programmed workloads by combining 8 applications selected from the SPEC CPU 2006 [10] benchmark suit. Table 2 summarizes key characteristics of

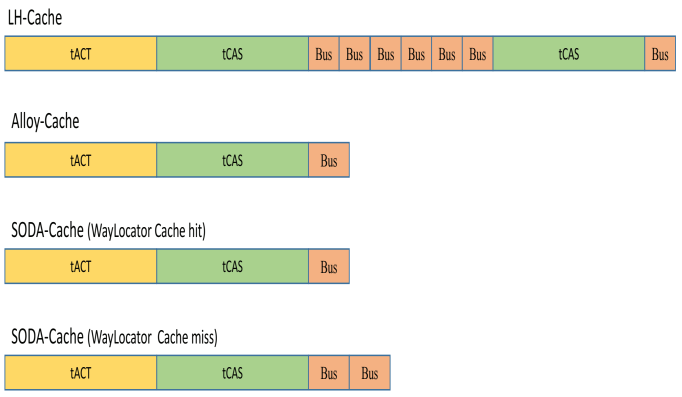

LH-Cache
tACT
tCAS
Bus
Alloy-Cache
SODA-Cache (WayLocator Cache hit)
SODA-Cache (WayLocator Cache miss)

Fig. 7: DRAM caches' access latency comparison

| Parameter | Value |
|---|---|
| Cores | 8-core CMP, 3.2 GHz, 128-entry instruction window |
| Shared L3 Cache | 24 cycles, 8MB, 16-way, 64B cache line |
| DRAM Cache | 256MB, bus frequency 1.6GHz, 4 channles, 1 rank per channel,16 banks per rank, 128 bits per channel, tCAS-tRCD-tRP-tRAS 10-10-10-25*cycles* |
| Main Memory | bus frequence: 800 MHz, 2 channles, 1 rank per channel, 8 banks per rank, 64 bits per channel, tCAS-tRCD-tRP-tRAS 10-10-10-25*cycles;* |

TABLE 1: Simulation Parameters

these workloads, including L3 Memory Per Kilo Instructions(MPKI). With representative phases, all applications in each workload run in parallel until each core executes at least 1 billion instructions.

We measure performance improvement with the weighted speedup defined as:

$$weightedspeedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}. \quad (1)$$

In addition, we report the geometric mean for these ten multi-programmed workloads. The LH-Cache is used as baseline DRAM cache, and Alloy-Cache is also evaluated. Since and ATCache [5] uses the small SRAM tag cache to mitigate the tag-then-data serialization, we also compare our SODA-Cache with ATCache. In our evaluation ,the ATCaches tag cache is set to be 95 KB.

### 4.1 Overall Preformance

Fig. 8 shows the weighted IPC improvement that SODA-Cache achieves over the LH-Cache and Alloy-Cache in 8-core workloads. As we can see, on average SODA-Cache improves the weighted IPC by 17%, 12.8% and 8.4% compared with LH-Cache, Alloy-Cache and ATCache respectively. For the workloads such as MIX4, MIX5 and MIX8, the Alloy-Cache works better than LH-Cache, due to its shorter hit latency. However, for the workload MIX6, Alloy-Caches IPC is worse than LH-Cache. This is because the LH-Caches higher hit rate can effectively reduce the memory traffic to the off-chip DRAM, achieving the performance superior to

| Workload | Description | L3 MPKI |
|---|---|---|
| MIX1 | milc, leslie, soplex, lbm, milc, mcf, leslie, soplex | 34.9 |
| MIX2 | astar, soplex, libq, Gems, astar, soplex, libq, Gems | 24.1 |
| MIX3 | libq, Gems, soplex, lbm, libq, Gems, soplex, lbm | 41.1 |
| MIX4 | leslie, leslie, soplex, sphinx3, leslie, leslie, soplex, sphinx3 | 22.7 |
| MIX5 | sphinx3, xalan, mcf, soplex, sphinx3, xalan, mcf, soplex | 32.9 |
| MIX6 | mcf, mcf, milc, milc, libq, libq, lbm, lbm | 44.4 |
| MIX7 | leslie, Gems, lbm, sphinx3, leslie, Gems, lbm, sphinx3 | 30 |
| MIX8 | leslie, libq, lbm, astar, leslie, libq, lbm, astar | 26.2 |
| MIX9 | mcf, leslie, milc, soplex, Gems, libq, lbm, sphinx3 | 34.6 |
| MIX10 | leslie, soplex, Gems, libq, lbm, astar, sphinx3, xalan | 25 |

TABLE 2: Characteristics of 10 workloads for an eight-core system

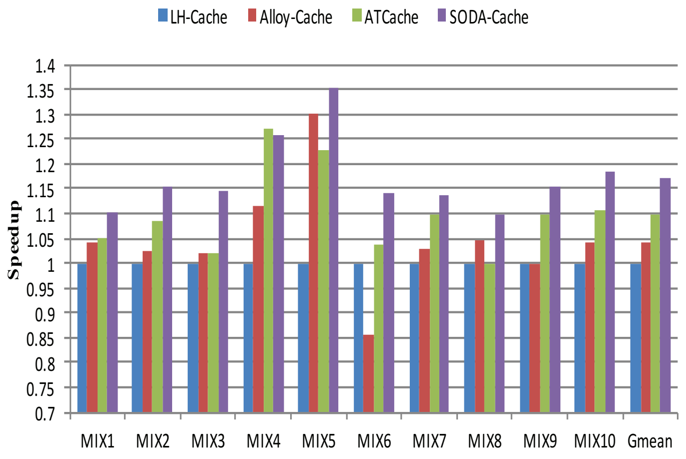


Fig. 8: Speedup

Alloy-Cache. Both hit rate and hit latency play important roles in overall performance. Optimization of either hit rate or hit latency cannot obtain the optimal performance. Our SODA-Cache effectively reduces the hit latency, does not sacrifice hit rate too much and therefore consistently outperforms over both LH-Cache and Alloy-Cache. Particularly, the SODA-Caches hit rate and hit latency are close to LH-Cache and Alloy-Cache respectively, achieving the optimal performance.

## 4.2 Hit Rate

The Fig. 9 presents DRAM cache hit rate for LH-Cache, Alloy-Cache and SODA-Cache. First, it is clear that the SODA-Caches hit rate is consistently higher than Alloy-Cache since Alloy-Cache is a directed-mapped but SODA-Cache is 2-way set associative cache. For example, SODAs hit rate on average 8.1% better than Alloy-Cache. Secondly, as expected, the SODA-Cache hit is inferior to LH-Cache for most workloads. This is because LH-Cache is a 58-way set associative cache, leading less conflict misses. On average, the SODA-Cache hit rate is 2.4% lower than LH-Cache. It is

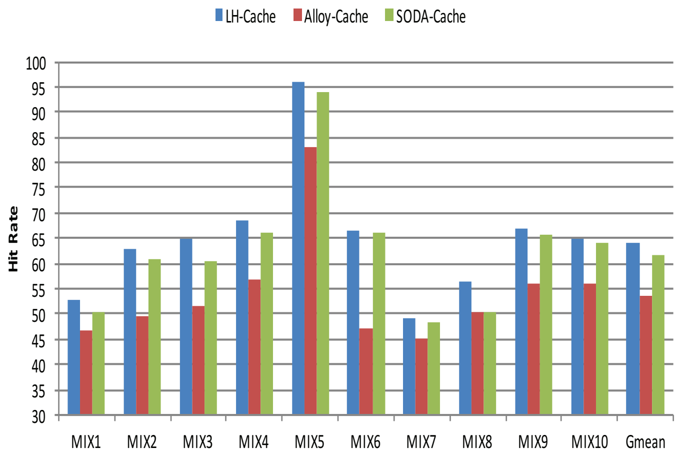


Fig. 9: DRAM Cache Hit Rate

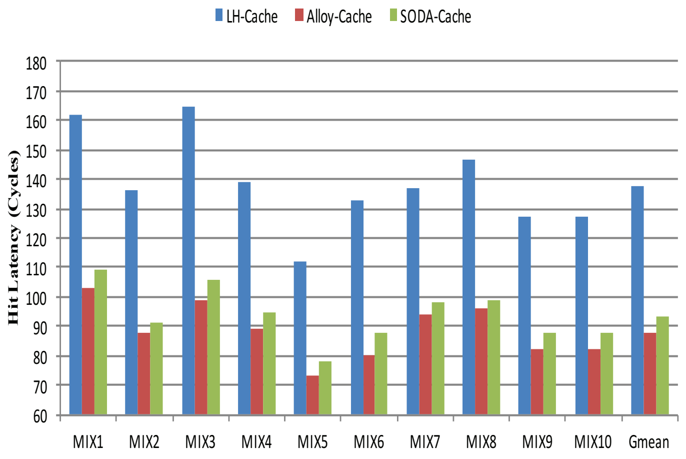


Fig. 10: DRAM Cache Hit Latency

concluded that the 2-way associative degree of SODA-Cache does not hurt hit rate too much.

## 4.3 Latency

The Fig. 10 compares the hit latency of SODA-Cache with LH-Cache, Alloy-Cache. First, SODA-Caches hit latency is 47.1% faster than LH-Cache on average. This hit latency reduction comes from our WayLocator cache and set layout. As discussed early, the WayLocator cache can avoid most tag-data serializations and all data lines accessed are required on WayLocator misses. Our experimental results show that WayLocator cache hit rate is 93.7% on average for these ten workloads. Further, the proposed set layout only requires one CAS command to stream both data line and tag line on WayLocator cache hit, without issuing two CAS commands in LH-Cache. Secondly, SODA-Caches hit latency is 5 cycles longer than Alloy-Cache on average. Attempting to avoiding tag-data access serialization, SODA-Cache still has to burst the second data line in case of WayLocator cache miss, resulting longer access latency than Alloy-Cache. Fortunately, the strong locality of WayLocator cache leads to 93.7% hit rate and makes WayLocator misses infrequent. Overall, the SODA-Cache significantly reduces hit latency and closely approaches to Alloy-Cache.

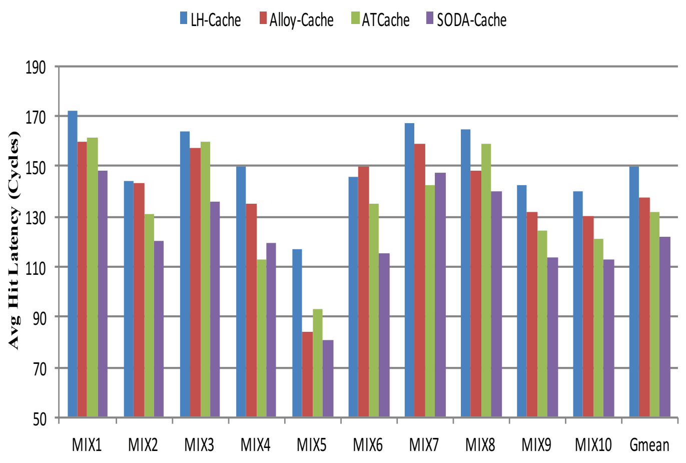


Fig. 11: Average DRAM Cache Access Latency

The Fig. 11 presents the average DRAM cache access latency for LH-Cache, Alloy-Cache, ATCache and SODA-Cache. It indicates that our SODA-Cache consistently outperform both LH-Cache, Alloy-Cache and ATCache. For example, SODA-Cache on average reduces the access latency by 23.1%, 13.2% and 8.6% compared with LH-Cache, Alloy-Cache and ATCache respectively. Our experimental results have vividly demonstrated that SODA-Cache successfully minimizes hit access latency and improves hit rate. Therefore, SODA-Cache achieves shorter average access latency. It is worth to note that SODA-Cache achieves neither the highest hit rate nor the minimal hit latency but obtains the optimal average access latency. This is because that the average DRAM cache access latency is the function of hit rate and hit access latency. The ATCache exploits the tag locality and cache the most recently accessed tags in a small cache in SRAM, reducing tag-data serialization. However, ATCache has to incur tag-data serialization on tag cache misses and causes the same latency overhead as LH-Cache. Compared with ATCache, our WayLocator cache achieves higher hit rate and the WayLocator cache miss latency is much shorter. It is concluded that our SODA-Cache design makes a good tradeoff between the hit rate and hit access latency to implement the optimal performance.

## 5 Related Work

### 5.1 Block-Based DRAM Cache

The tag storage overhead of block-based DRAM cache design make it impractical to store tags in SRAM. To efficiently implement storing tags in DRAM cache, Loh-Hill et al. [1], [2] co-locates tags with data in the same row, and schedule the data request right after the tag read, improving row buffer efficiency. They organize each 2KB DRAM row as a 29-way cache set, providing good hit rate. They However, it suffers from tag access latency.

Alloy Cache [3] breaks the tag-and-data serialization by statically mapping each data block to the DRAM cache. It also combines each tag with its corresponding data into a single entity to eliminate the standalone tag read command, with introducing one additional burst latency. CAMEO [4] develops this design by eliminating the data copy between the direct-mapped cache and the main memory. The in-memory data that mapped to the same cache line, as well as the data that currently occupies the line, are identical to each other, and contend the cache line via swapping, instead of inserting the data copy.

Based on direct-mapped structure, BEAR Cache [11] proposes a series of techniques to reduce the bandwidth waste due to secondary operations related to cache functions. These techniques include the bandwidth-aware bypass for miss fill, DRAM cache presence bit for writeback probe, and neighboring tag cache for miss probe. Mostly-Clean Cache [12] balances the bandwidth of DRAM cache and off-chip memory. It utilizes the idle off-chip bandwidth when the DRAM cache is servicing a burst of cache hits. In addition, the bandwidth optimization is not the focus of this paper and the SODA-cache does not increase the bandwidth overhead compared with the Alloy Cache. Therefore the recently proposed bandwidth reduction scheme, such as BEAR [11], which is based on Alloy-cache, can be applied to the SODA-cache.

ATCache [5] accelerates the probe procedure via introducing a small SRAM tag cache. It exploits spatial locality by prefetching tags from nearby cache sets, and limits the tag cache pollution with an on-demand tag prefetching scheme. Our performance evaluations have shown that our SODA-cache outperforms ATCache.

### 5.2 Page-based DRAM Cache

There exit a plenty of spatial locality in page-based DRAM cache, providing good hit rate. In addition, the small tag storage overhead makes it possible to implement tag-in-SRAM, reducing tag access latency. However, page-based DRAM cache [13], [14] suffers from data over-fetch, wasting bandwidth because fetched pages contains unused data. To address this issue, Footprint Cache [15] was proposed to track the workload's access footprints [16], [17], and only prefetch the lines that will likely be touched during the page's presence in cache. F-TDC [18] integrates the footprint history table into the page table, and hide the cache probes into the TLB lookup procedures.

### 5.3 Hybrid DRAM Cache

To exploit the best aspects of block-based cache and page-based cache, Unison Cache [19] and Bi-Modal Cache [20] increase the cache line granularity to exploit the spatial locality, and they rely on the set-associative structure to relieve the introduced cache conflicts. In addition, the Unison cache [20] is a page-based cache targeting the workload with strong spatial locality and our SODA-cache is a block-based cache targeting the workload without strong spatial locality.

## 6 Conclusion

The hit rate and hit latency are two conflicting optimization goals in DRAM cache design. While the higher hit rate implemented by high set associativity entails large tag access latency overhead, the optimal hit latency mandates direct-mapped cache and suffers from lower hit rate. In this paper, we presented SODA-Cache, a DRAM cache architecture to simultaneously optimize hit rate and hit latency. SODA-Cache chooses 2-way set associative cache based on the observation of hit rate and set associativity, achieves good hit rate and requires smaller tag storage, facilitating the minimization of tag access latency. SODA-cache exploits the WayLocator cache to directly retrieve a data line from DRAM cache, before accessing tag and therefore removing the data locating latency from data access path. Further to decrease the hit latency, we propose a novel set layout that can stream out tag and data together from DRAM cache by issuing a CSA DRAM command.

Experimental results under 10 multi-programmed workloads show that SODA-Cache can improve hit rate by 8.1% compared with Alloy-cache and reduce average access latency by 23.1% and 13.2% and 8.4% compared with LH-cache, Alloy-cache and 8.6% respectively on average. Accordingly, SODA-cache outperforms over LH-cache, Alloy-cache and ATCache by on average 17%, 12.8% and 8.4% respectively in term of weighted speedup.

## Acknowledgments

This work was supported by the National High-Tech Research and Development Program (863 Program) under grant No. 2015AA015303, National Natural Science Foundation of China (NSFC) under grants No. 61672251, 61628204, 61433019 , the US NSF under Grants NSF-CCF-1618194.

## References

[1] 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*, 2011.

[2] G. Loh and M. D. Hill, "Supporting very large dram caches with compound-access scheduling and missmap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, 2012.

[3] 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 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[4] C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[5] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, 2014.

[6] F. Hameed, L. Bauer, and J. Henkel, "Reducing latency in an sram/dram cache hierarchy via a novel tag-cache architecture," in *Proceedings of the 51st Annual Design Automation Conference*, 2014.

[7] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Architecting efficient interconnects for large caches with cacti 6.0," *IEEE Micro*, vol. 28, no. 1, pp. 69–79, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1109/MM.2008.2

[8] Intel. Pin - a dynamic binary instrumentation tool. [Online]. Available: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[9] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI*, ser. ISVLSI '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 392–397. [Online]. Available: http://dx.doi.org/10.1109/ISVLSI.2012.82

[10] SPEC. Spec cpu2006 benchmark suite. [Online]. Available: http://www.spec.org/cpu2006/

[11] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," in *Proceedings of the 42th International Symposium on Computer Architecture*, 2015.

[12] 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 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[13] 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 *Proceedings of the16th International Symposium on High Performance Computer Architecture*, 2010.

[14] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Adaptive filter-based dram caching for cmp server platforms," in *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, 2010.

[15] 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*, 2013.

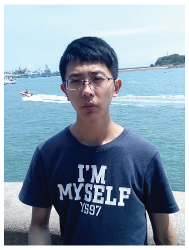
[16] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, 2006.

[17] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.

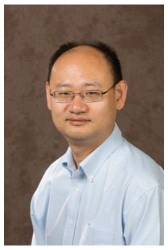
[18] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *Proceedings of the 22nd IEEE International Symposium on High Performance Computer Architecture*, 2016.

[19] 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*, 2014.

[20] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-modal dram cache: Improving hit rate, hit latency and bandwidth," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.



**Pai Chen** Pai Chen is a master student of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Pai received his B.Sc.'s Degree in Northeastern university, china, in 2015. His research interests mainly include computer architecture, machine learning and deep learning.



**Jianhui Yue** Jianhui Yue received the PhD degree from the University of Maine, Orono, in 2012. He is an assistant professor of Computer Science department, Michigan Technological University, Michigan. Before joining Michigan Tech. University, he was a visiting assistant professor with Miami University, Ohio. His research interests include computer architecture and systems. He served as the program committee of international conferences, including ICPP and CCGrid. He received the Best Paper Award at IEEE CLUSTER 07 and was the Best Paper Award candidate at HPCA 13.

**Xiaofei Liao** Xiaofei Liao received a Ph.D. degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2005. He is now a Professor in school of Computer Science and Technology at HUST. His research interests are in the areas of system virtualization, system software, and big data processing.

**Hai Jin** Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. Jin is a Fellow of CCF, senior member of the IEEE and a member of the ACM. He has co-authored 22 books and published over 800 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.