Chi Y, Guo RT, Liao XF *et al.* P3DC: Reducing DRAM cache hit latency by hybrid mappings. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(6): 1341-1360 Nov. 2024. DOI: 10.1007/s11390-023-2561-y

P3DC: Reducing DRAM Cache Hit Latency by Hybrid Mappings

Ye Chi[1, 2, 3, 4, 5] (池　也), Ren-Tong Guo[1, 2, 3, 4] (郭人通)
Xiao-Fei Liao[1, 2, 3, 4, *] (廖小飞), *Distinguised Member, CCF*
Hai-Kun Liu[1, 2, 3, 4] (刘海坤), *Senior Member, CCF, Member, IEEE*, and Jianhui Yue[6] (岳建辉)

[1] *National Engineering Research Center for Big Data Technology and System, Wuhan 430074, China*
[2] *Services Computing Technology and System Laboratory, Wuhan 430074, China*
[3] *Cluster and Grid Computing Laboratory, Wuhan 430074, China*
[4] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[5] *School of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China*
[6] *Department of Computer Science, Michigan Technological University, Houghton 49931-1295, U.S.A.*

E-mail: ychi@hust.edu.cn; rtguo@hust.edu.cn; xfliao@hust.edu.cn; hkliu@hust.edu.cn; jyue@mtu.edu

Received June 4, 2022; accepted September 28, 2023.

Abstract Die-stacked dynamic random access memory (DRAM) caches are increasingly advocated to bridge the performance gap between the on-chip cache and the main memory. To fully realize their potential, it is essential to improve DRAM cache hit rate and lower its cache hit latency. In order to take advantage of the high hit-rate of set-association and the low hit latency of direct-mapping at the same time, we propose a partial direct-mapped die-stacked DRAM cache called P3DC. This design is motivated by a key observation, i.e., applying a unified mapping policy to different types of blocks cannot achieve a high cache hit rate and low hit latency simultaneously. To address this problem, P3DC classifies data blocks into leading blocks and following blocks, and places them at static positions and dynamic positions, respectively, in a unified set-associative structure. We also propose a replacement policy to balance the miss penalty and the temporal locality of different blocks. In addition, P3DC provides a policy to mitigate cache thrashing due to block type variations. Experimental results demonstrate that P3DC can reduce the cache hit latency by 20.5% while achieving a similar cache hit rate compared with typical set-associative caches. P3DC improves the instructions per cycle (IPC) by up to 66% (12% on average) compared with the state-of-the-art direct-mapped cache—BEAR, and by up to 19% (6% on average) compared with the tag-data decoupled set-associative cache—DEC-A8.

Keywords die-stacked dynamic random access memory (DRAM), cache, set-associative, direct-mapped, hit latency

1 Introduction

In the post-Moore's Law era, the memory wall problem has become a critical topic of interests for both academic and industrial communities. 3D die-stacked dynamic random access memory (DRAM) such as High Bandwidth Memory (HBM)[1] and Hybrid Memory Cube (HMC)[2] provides high bandwidth as well as high energy efficiency. It has been widely exploited to address the memory wall problem in high-performance servers[3, 4]. For example, the next-generation Intel Sapphire Rapids Xeon scalable processors offer optional on-chip HBM memory as large as 64 GB. It can be used as the last-level cache or the main memory to improve the system performance[5, 6].

Although 3D die-stacked DRAM can achieve gigabyte-scale memory capacity, it is not large enough to replace the off-chip DRAM as the main memory. Thus, it is often utilized as the last-level cache, namely DRAM cache or near memory[7–18]. Because of the relatively large capacity of 3D DRAM, the DRAM

Regular Paper

This work was supported jointly by the National Key Research and Development Program of China under Grant No. 2022YFB45-00303, and the National Natural Science Foundation of China under Grant Nos. 62072198, 61825202, and 61929103.

*Corresponding Author

cache's tag storage overhead is very high. This is a significant challenge to design a high-performance DRAM cache. For instance, for a 512 MB DRAM cache with 64B cache lines, its tags occupy about 24 MB storage. To reduce the storage overhead, prior studies propose two solutions: 1) storing tags in the DRAM cache with a small granularity of cache lines, and 2) storing all tags in the static random access memory (SRAM) with a large granularity of cache lines. However, they both have limitations. The co-location of data lines and tag lines serializes the tag access and data access from the DRAM cache, increasing the cache hit latency. The large-sized cache lines suffer from high DRAM bandwidth consumption, high read/write amplification, and low capacity scalability of DRAM cache. In this paper, we mainly focus on DRAM cache designs with small cache lines.

Hit latency and cache hit rate are the two important performance metrics in DRAM cache designs. Alloy Cache[9] was proposed to merge a data line with its tag in a tag-and-data (TAD) unit and implements tag lookup by issuing a Compare-and-Swap (CAS) command. In this way, the tag-data access serialization has been eliminated, reducing hit latency. However, TAD can only be applied to direct-mapping caches, thereby reducing hit rates. LH Cache[7, 8] architects the DRAM cache as a set-associative cache by co-locating the tags with data blocks in the same row, to achieve high hit rates. Upon a request, the DRAM cache controller has to search all tag lines in a set by issuing a CAS DRAM command before ascertaining the location of the requested data line. The cumbersome tag access latency increases the data hit latency. Some studies[19, 20] propose to cache the tags in a small on-chip SRAM to accelerate the tag lookup for the set-associative DRAM cache. Upon a tag-cache hit, the data block can be fetched from the DRAM cache without accessing the tag in the DRAM cache. However, upon a tag cache miss, the tag should be fetched into the tag cache before the data block is accessed. Therefore, the TAD access serialization cannot be completely avoided from the data access path, resulting in sub-optimal performance. These studies organize the DRAM cache as a direct-mapped cache or a set-associative cache exclusively, and cannot optimize the hit latency and hit rate simultaneously.

The tag cache using SRAM is an effective way to reduce the DRAM hit latency[19]. We also have some interesting observations on the set associative cache with a tag cache. On the tag cache miss, a batch of tags will be fetched from the DRAM cache into the SRAM tag cache before the requested data is accessed in the DRAM cache. The target data block in the first access of this set is referred to the leading block, and the remaining blocks are referred to the following blocks. Only the leading block incurs the tag fetching overhead, while the following blocks can benefit from the fast tag lookup in SRAM.

Motivated by the above key observations, we propose P3DC, a partial direct-mapped die-stacked DRAM cache that exploits different access overheads of leading blocks and following blocks to achieve both low hit latency and high cache hit rates. Specifically, P3DC maps leading blocks to the fix position in a set (referred to as static mapping) to reduce cache hit latency, and maps following blocks to the remaining positions in a set (referred to as dynamic mapping) to improve cache hit rates. However, P3DC still faces two challenges. Firstly, we find that leading blocks and following blocks have different miss penalties in terms of latency and bandwidth. Secondly, the frequent block type transitions should be handled efficiently to mitigate performance degradation. To address these challenges, we propose a new cache replacement policy and a high-frequent block variation filter accordingly. Overall, the major contributions of this paper are summarized as follows.

- We find that two different block types, i.e., leading and following blocks, have distinct impacts on the hit latency and hit rate of DRAM cache mapping policies. These findings challenge the single mapping policy on all blocks proposed by conventional designs.
- We propose a partial direct-mapped die-stacked DRAM cache, called P3DC, to achieve both low hit latency and high cache hit rates simultaneously. Specifically, it applies static and dynamic mapping to leading and following blocks, respectively. Apart from the novel mapping scheme, we propose a cache replacement policy called Range-Variable CLOCK (RV-CLOCK) to further improve the cache performance, considering the miss penalties of data blocks with different block types. Furthermore, we propose a high-frequent variation filter to handle the frequent block type transitions.
- Through extensive evaluations, we demonstrate that P3DC can reduce the cache hit latency by 20.5% while achieving a comparable hit rate compared with set-associative caches. P3DC improves the IPC by up to 66% and 19% compared with BEAR[21] and DEC-A8[22], respectively.

The rest of this paper is organized as follows. We present the background in Section 2 and motivations

in Section 3. Section 4 introduces the system design. Experimental methodologies are given in Section 5, followed by evaluations in Section 6. We present the related work in Section 7 and conclude the paper in Section 8.

2 Background

2.1 DRAM Cache Organizations

The stacked DRAM[23–27] can be organized as either a direct-mapped cache[9, 21] or a set-associative cache[7, 28], as shown in Fig.1. The set-associative cache organizes each DRAM row as a set. Upon insertions, data blocks can be dynamically placed to any associated cache lines. This offers good replacement flexibility at the expense of high lookup latency because all tags in the set must be checked in order to locate the requested data block. To address this problem, the set-associative cache combines several tags as one batch to reduce the tag access requests. In contrast, the direct-mapped cache integrates tags and data into a single entity to reduce the hit latency. Upon an L3 miss, tags and data can be directly located by a static mapping function. The whole entity is read by a single request, rather than by serialized tags and data accesses.

Storing tags in stacked DRAM show better scalability for large DRAM caches, but may introduce non-trivial probe latency. A widely-adopted solution is to use a portion of on-die SRAM as a tag cache[10, 21, 28]. In the set-associative cache, tags are fetched in a batch. These tags are then cached in the on-chip SRAM after the tag lookup, to effectively exploit spacial locality. The direct-mapped cache organizes tags and data in an interleaving manner. Upon each DRAM cache access, the tag of the next block can be fetched along with the currently requested data. The prefetched adjacent tag is then cached in the SRAM for future tag lookup.

2.2 Access Latency Breakdown

Fig.2 illustrates the access latency of the two cache structures. The direct-mapped cache can offer the lowest hit latency by fetching tags and data in a single request when the DRAM cache is hit (cases A_1 and B_1). However, if there is a simultaneous tag cache miss and DRAM cache miss (case D_1), an extra cache probe is needed before the main memory is accessed. If the tag cache misses, i.e., the data block does not exist in the DRAM cache (case C_1), the request will be sent to the main memory.

Unlike the direct-mapped cache, the set-associative cache would suffer from the cache probe latency even if the request hits the DRAM cache (case B_2). For the other cases (cases A_2, C_2, and D_2), the set-associative cache behaves in the same way as the direct-mapped cache, resulting in similar access latency.

From Fig.2, we find that the benefits of the direct-mapped structure depend on whether the DRAM cache hits or not. It provides good hit latency, but its performance is worse in cache misses. In addition, the tag fetching procedure introduces non-trivial latency in the set-associative cache. If a request involves tag fetching, the overall latency can be close to a main memory access, even if the DRAM cache is hit.

3 Motivation

We first present several key definitions that are used in the rest of this paper. A section is defined as a continuous logical address region mapped to a single cache set and hence data blocks in the same region are mapped to the same cache set. This mapping enables us to preserve the spatial locality exhibited in

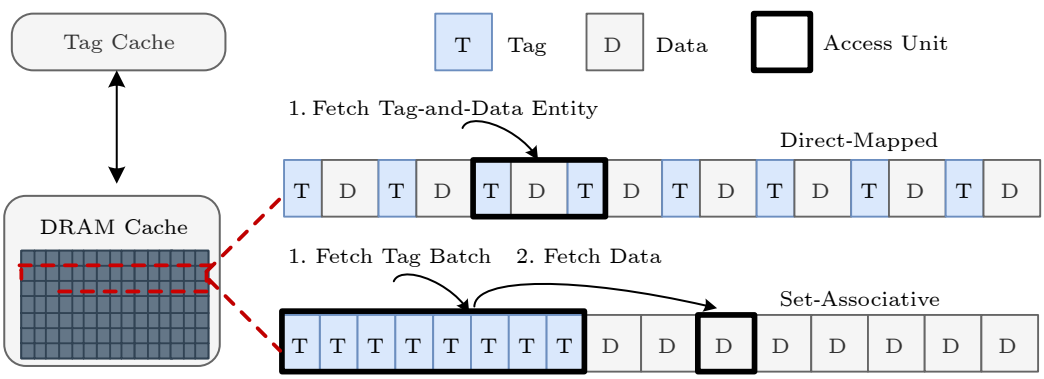


Fig.1. Basic organizations of a direct-mapped cache and a set-associative cache.

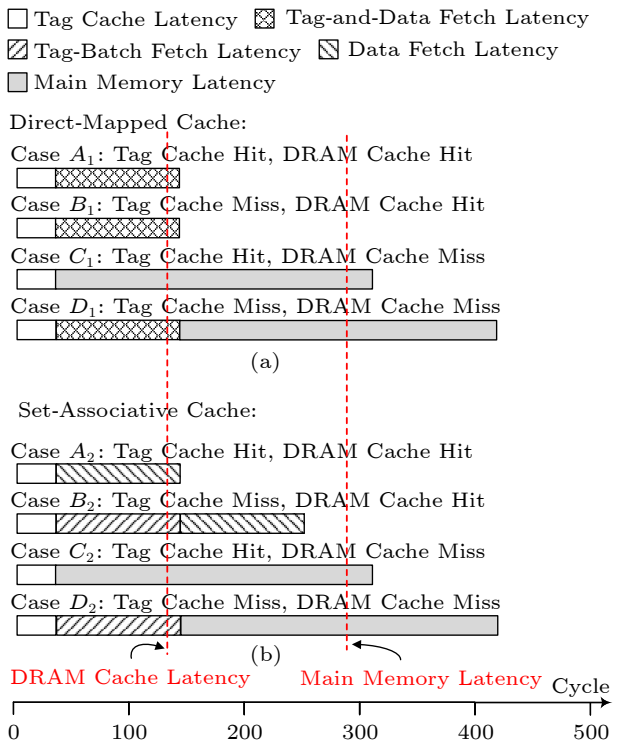


Fig.2. Latency breakdown. (a) Direct-mapped cache. (b) Set-associative cache.

workloads. A section is called active if any of its tags is cached in the tag cache, or is currently being fetched. Otherwise, it is inactive. We also define two block types. A leading block is the first accessed block within an individual section when the section becomes active, and the following blocks are the remaining ones accessed subsequently until the section is no longer active. There could be multiple sections mapped to the same set at the same time. For example, data blocks in section A and data blocks in section C reside in the same set, as shown in Fig.3. Although this mapping seems to be worse than the conventional mapping in high-level caches, our experimental results show it achieves a hit rate close to the conventional mapping in a DRAM cache due to the poor locality. For each L3 cache miss, the tag cache should be checked before accessing the DRAM cache. On a tag cache miss, all tags in the target DRAM cache set, referred to as a tag batch, are fetched to the tag cache in SRAM. A section is active if any of its tags is cached in the tag cache, or is currently being fetched. Otherwise, it is inactive. A leading block changes the corresponding section from an inactive state to an active state, and subsequent accesses are following blocks before the section becomes inactive, which is caused by tag cache replacement. In other words, a leading block is the data block experiencing

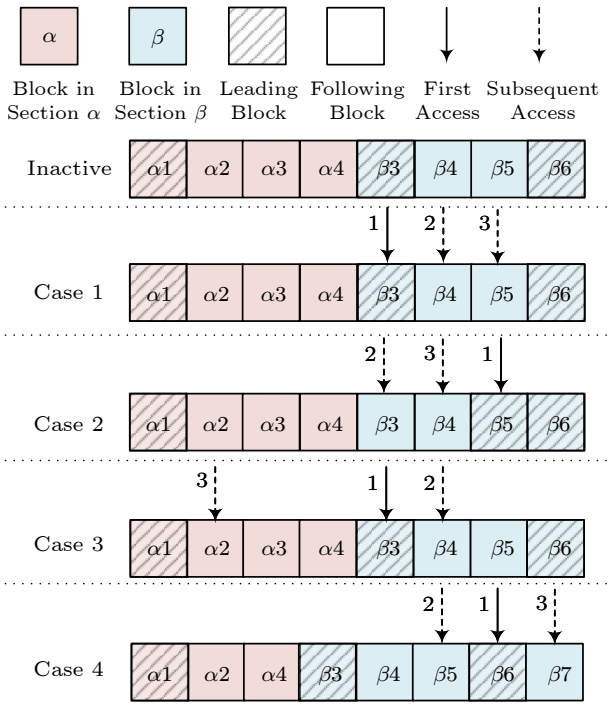


Fig.3. Leading and following data blocks in a set.

tag cache miss and the subsequently-accessed blocks in the same section are following blocks when their tags are in the tag cache.

We use several simple examples to elaborate the leading blocks and following blocks, as shown in Fig.3. Assume that there are eight data blocks from section α and section β in the same DRAM cache set, and their tags are not in the tag cache. Accordingly, these two sections are in inactive state. We assume that during the last time the cache set was replaced from the tag cache, section α has one leading block $\alpha1$ and section β has two leading blocks $\beta3$ and $\beta6$. In case 1 with the access sequence of $\beta3$, $\beta4$ and $\beta5$, $\beta3$ becomes a leading block, because it misses the tag cache. After $\beta3$ has been served, the set's tag has been fetched to the tag cache, and the subsequent accesses to $\beta4$ and $\beta5$ hit the tag cache. They become following blocks. Case 2 shows new leading block, $\beta5$, in section β because its access misses the tag cache, denoting the existence of multiple leading blocks for a section. In case 3, $\alpha2$ becomes a following block because when $\beta3$ is accessed, the whole cache set is uploaded to the tag cache which not only activates section β, but also activates section α. Case 4 presents that $\beta7$ is also a following block even its tag cache misses because section β has been activated by $\beta6$.

In this section, we would ask the following four questions that motivate the P3DC design.

• How much reduction in hit latency can we achieve from leading blocks?

• What improvement in hit rate can we achieve by following blocks?

• Should we consider the different miss penalties of leading blocks and following blocks in the data replacement policy?

• Do leading blocks and following blocks frequently shift to each other?

In order to answer the above questions, we experimentally study two state-of-the-art cache designs, namely BEAR cache (direct-mapped cache)[21] and DEC-A8 cache (set-associative cache)[22]. The experimental methodology is detailed in Section 5.

3.1 Hit Latency Reduction from Leading Blocks

A set-associative cache offers higher hit rates, but incurs higher hit latency. In the DRAM cache, the large associativity necessitates the tags co-located with the data blocks in the same row, and tags should be fetched before data blocks. Such a serialization of tag and data blocks is the root cause for high hit latency. Our experimental results show that the serialized tags and data accesses will significantly increase the cache's hit latency by 1.7x∼2.3x.

Tag cache is an effective way to reduce the hit latency in the DRAM cache. Upon a tag cache miss, all tags of the set are fetched from the DRAM cache and then are stored in SRAM to accelerate the tag lookup for following accesses. Due to spatial locality, most of following accesses benefit from the quick tag lookup in SRAM, avoiding tag fetching from the DRAM cache.

In Fig.4, our experiments show the breakdown of tag fetching caused by the leading blocks and the following blocks for 18 workloads (detailed in Subsection 5.2). We find that on average 89% tag fetches are triggered by the leading blocks, which experience the serializations of data and tag accesses. Therefore, removing these serializations could effectively reduce the hit latency. This observation motivates us to apply the direct mapping to the leading blocks. Without tag probation latency, the direct mapping can remove 89% tag and data serializations, reducing the hit latency.

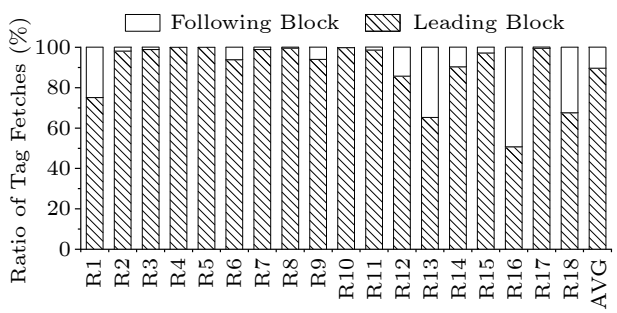


Fig.4. Distributions of tag fetches.

3.2 Hit Rate Improvement from Following Blocks

Different from leading blocks, most following blocks involve only data fetching since their tags are cached in SRAM. Our experimental results show that 97% of the following block hit latency is contributed by the data fetching. Applying static mapping to following blocks can hardly further reduce the hit latency. Moreover, static mapping limits the replacement flexibility and can decrease the hit rate greatly. Above observations motivate us to apply dynamic mapping for following blocks to increase hit rates while maintaining similar hit latency with static mapping.

The proportion of following blocks affects the hit rate improvement in dynamic mapping. In a corner case with no following blocks, the DRAM cache degenerates into a direct-mapped cache with a low hit rate. In contrast, if the proportion of following blocks is 100%, the DRAM cache becomes a set-associative cache with high hit latency. In practice, this proportion is determined by workload behaviors. Our experiments with 18 typical workloads show that the average proportion of following blocks is 88%, which is large enough to approximate the set-associative cache.

3.3 Block Miss Penalty

Fig.5 shows the miss penalty of a leading block and a following block. Fig.5(a) compares the DRAM cache hit with the DRAM cache miss for the leading block. Since the leading block is statically mapped to the DRAM cache, the DRAM cache controller directly fetches the requested leading block from the DRAM cache, requiring only one DRAM cache access. After reading the leading block from the DRAM cache, the cache controller detects the cache miss, and fetches the tag batch from the DRAM cache and the requested data block from off-chip DRAM simultaneously, as shown in Fig.5(b).

Fig.5 shows that the miss penalty of the leading block is attributed to a data block access from the off-chip DRAM in terms of latency and a tag batch transfer over the DRAM cache in terms of bandwidth.

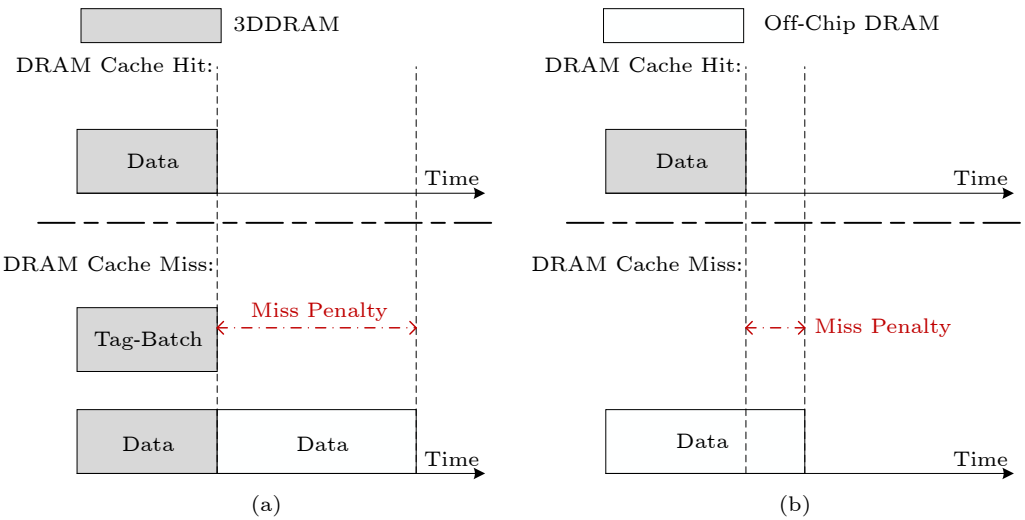


Fig.5. Miss penalty comparison of a leading block and a following block. (a) Accessing a leading block. (b) Accessing a following block.

Fig.5(a) shows the miss penalty of the following block, which is the difference in latency between the DRAM cache and the off-chip DRAM. For the following block, the cache controller can quickly determine the cache miss/hit with the help of the tag batch in SRAM. Upon a DRAM cache hit, the requested block is fetched from the DRAM cache. Otherwise, one off-chip DRAM access is involved. Therefore, the leading block has larger miss penalties than the following block in terms of both latency and bandwidth consumption.

Table 1 shows the average latency and bandwidth consumption of 28 workloads for different cases. It demonstrates that the leading blocks incur larger miss penalty than the following blocks in terms of latency and bandwidth. Upon a leading block miss, the cache controller should first access the DRAM cache to request both tags and data, and then the data is fetched from the main memory. Compared with the case of the leading block hit, the leading block miss incurs extra latency of 273 cycles and occupies additional memory bandwidth of 64 bytes. For following blocks, their miss penalty is much lower because block locations can be given by the tag cache. Upon a DRAM cache miss, the following block is read directly from the main memory without accessing the DRAM cache, and the latency penalty is only 112 cycles. Furthermore, the following block miss uses 64-byte memory bandwidth, which is the same as the hit case. The different miss penalty of leading and following blocks motivates us to minimize leading block misses in our data replacement policy.

Table 1. Miss Penalty of Leading Blocks and Following Blocks

Type	Parameter	Latency (Cycle)	Bandwidth (Byte)
Leading blocks	Hit	185	Cache: 128 (tag+data)
	Miss	458	Cache: 128 (tag+data)
			Memory: 64 (data)
	Penalty	273	Cache: 64 (data)
Following blocks	Hit	147	Cache: 64 (data)
	Miss	259	Memory: 64 (data)
	Penalty	112	–

3.4 Block Type Stability

The block type stability is important for our cache mapping design. Some leading blocks and following blocks may switch roles when a section becomes active in different periods. For example, assuming a following block is cached in the DRAM cache via dynamic mapping, when the corresponding section is activated again due to the following accesses, this block may become a leading block and static mapping should be applied. Because the mapping switches, we have only $1/M$ (M is the set associativity) probability to locate the right position of this block.

We measure the percentage of blocks that their types dynamically switch for 18 workloads and show the sorted results in Fig.6. The average percentage of block type switches is less than 10%, and only two workloads (e.g., R7: vpr) have more than 30% type switches. These results show that block types are almost stable, implying that it is feasible for most workloads to apply different mapping schemes to leading and following blocks, respectively. However, we still face the challenge of minimizing the impact of block type switching for some workloads, such as R7 and R8.

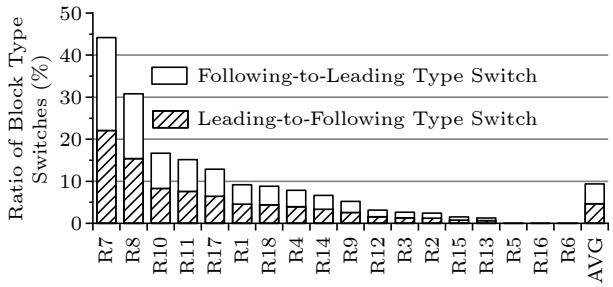


Fig.6. Percentage of block type switches.

4 P3DC Design

This section describes the design of P3DC. We first present the overview of P3DC in Subsection 4.1, and then describe the block mapping policy in Subsection 4.2. The proposed DRAM cache replacement algorithm and the scheme of handling frequent block type switches are detailed in Subsection 4.3 and Subsection 4.4, respectively.

4.1 Overview

Fig.7 shows the architecture of P3DC. Each core is configured with a MAP-I predictor to estimate whether the requested data is hit in the DRAM cache or not[9]. If a cache miss is predicted, the DRAM cache and the main memory are accessed simultaneously. An SRAM Tag-cache is used to store a batch of associated tags, and thus can further reduce memory access latency. Like BEAR[21], P3DC can also bypass the DRAM cache to directly fetch data with a low reuse probability to CPUs, and thus improves the DRAM cache utilization and reduces bandwidth consumption.

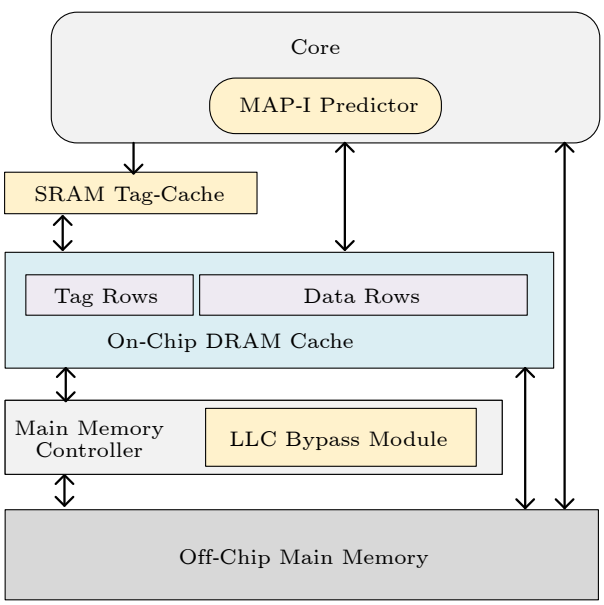


Fig.7. Architecture of P3DC.

As discussed in Section 3, using different mapping schemes for leading blocks and following blocks could improve the hit latency and hit rates simultaneously in the context of a tag cache. This is the key idea of P3DC. The type of the requested data blocks determines how P3DC serves the request. For a leading block, its direct mapping enables the cache controller to fetch the data from a specific position in the DRAM cache set without consulting the tag, and the related tag batch is also transferred to the tag cache in SRAM. In this way, the cache controller can detect the DRAM cache miss for the leading block, after checking the tag batch. In order to concurrently transfer both the leading block and the tag batch, the dataset and the corresponding tags are stored in different banks, as shown in Fig.8. For the following block, the requested data needs to consulting its tag to determine the position in the set, since the dynamic mapping is applied to the following block. The following block is most likely to hit the tag cache and can be quickly accessed from the DRAM cache.

Fig.8(a) shows the organization of a tag row. A tag batch contains the tags for the data blocks stored in the corresponding cache set, which is similar to the tag batching mechanism in previous work[7]. Four extra bits are introduced to a tag: a reference bit (A), a priority bit (H), and a two-bit filter (C). The reference bit represents the recency of the block. The priority bit denotes the block's caching priority and also acts as the flag of block types (1 and 0 denote the leading and following blocks, respectively). The filter bits record the transition of block types, as discussed in Subsection 4.4.

Fig.8(b) illustrates the organization of the data row. In conventional designs, the direct-mapped cache and the set-associative cache are commonly viewed as two exclusive structures. Generally, the cache associativity and the mapping policy are closely coupled. If the cache associativity is 1, the mapping policy degenerates to a static direct-mapping scheme. On the contrary, if the cache associativity is greater than 1, the mapping policy becomes a dynamic set-associative mapping. In order to support both static mapping and dynamic mapping in a unified structure, we decouple the set associativity from the data mapping policy. Basically, P3DC organizes the DRAM cache similar to set-associative caches. Each DRAM row has several sets and each set contains 16 data lines. Thus,

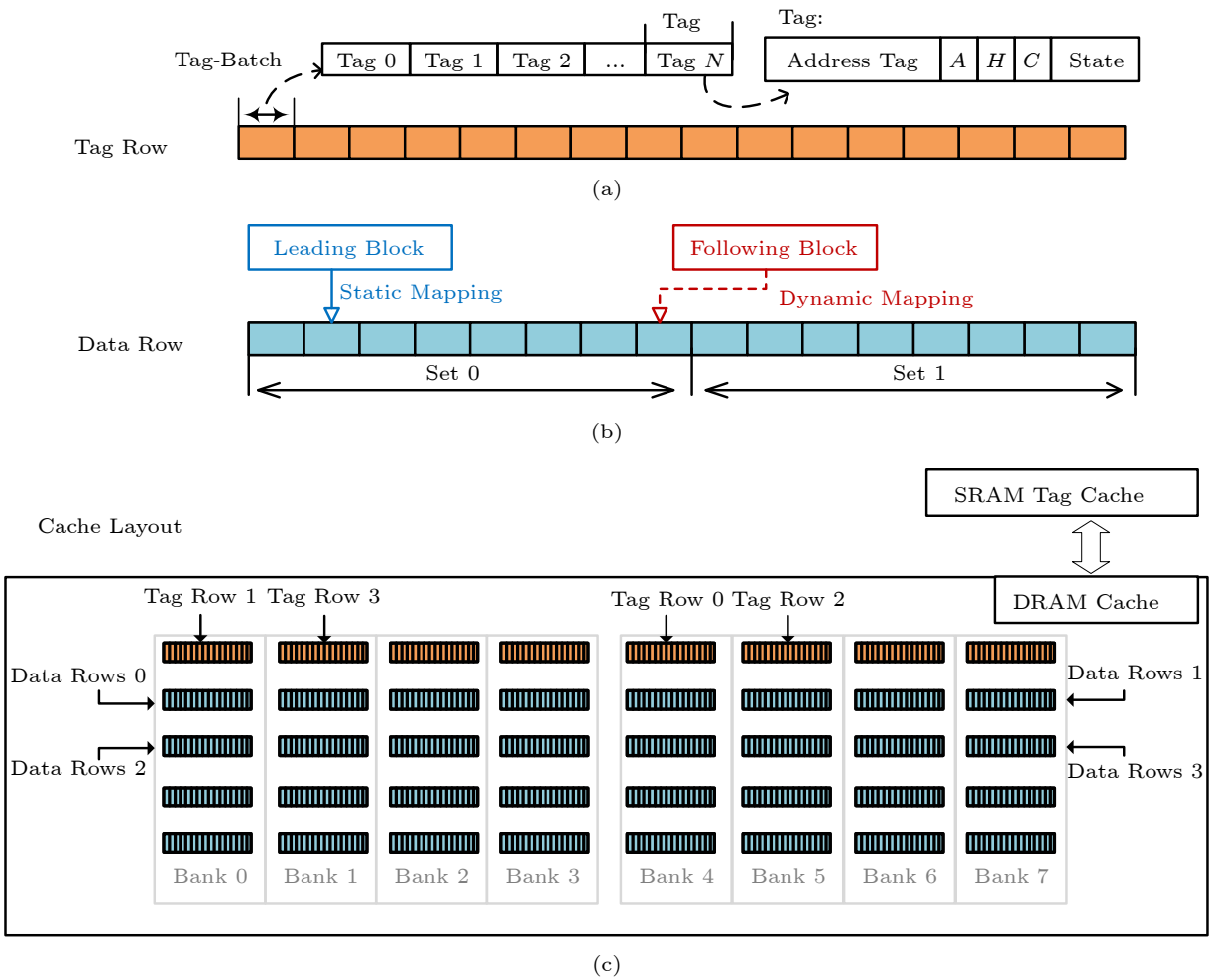


Fig.8. Organization of the P3DC cache. (a) Organization of a tag row. (b) Organization of a data row. (c) DRAM cache layout.

P3DC actually adopts an organization of a 16-way set-associative cache. Within each cache set, data blocks can be mapped via either static or dynamic mapping, depending on their block types, as shown in Fig.8. Specifically, following blocks are placed in the set according to the replacement algorithm, while leading blocks' positions are determined by static mapping, which is the predetermined hash function of the block address.

Fig.8(c) illustrates the layout of the P3DC cache. The DRAM rows are divided into tag rows and data rows. Tags and data are mapped to the rows of different banks to improve the bank-level parallelism.

4.2 Mapping Policy

We apply static address mapping and dynamic address mapping to leading blocks and following blocks to determine their positions in the cache set, respectively. The tag of each data block has a priority bit to indicate the block type. For a leading block, the address of the request block is hashed by the predetermined function *StaticMapping*() and its outcome is the position for this leading block. Once the position of a leading block is determined, the memory controller directly retrieves the data block from the DRAM cache. It is possible that there are multiple leading blocks in one cache set since the position depends on the request block address. In that case, position conflicts caused by the hash function could lead to cache evictions. For a following block, its position in the set can be determined by probing the tags stored in the tag cache in the case of a tag cache hit. Otherwise, the cache replacement algorithm determines the position where the following blocks will be placed in the cache set. As shown in Algorithm 1, we use *GetPosition*() to denote the position determined by the cache replacement policy.

When the data type is transformed from a following block to a leading block, it is necessary to update

the tag bits of this data block and migrate it to the position indicated by static mapping. Algorithm 1 shows how P3DC handles data type transitions. For the transition to the following one, P3DC resets its priority bit (lines 4 and 5). For the transition to the leading one, P3DC tries to migrate it to the statically mapped position (lines 6–17). If the data block's current position corresponds to the given static mapping (lines 7 and 8), no migration is needed, and P3DC sets its priority bit to 1. Otherwise, the mapping is determined in the following two situations. If the static-mapped position is of a high priority, and is recently referenced, P3DC just resets its reference bit (lines 10 and 11), without data migration. In other cases, P3DC migrates the data to its static-mapped position, with an expectation that the data will be continually accessed as a leading block in the near future.

Algorithm 1. Mapping Policy

```
Input: block_address, type
1  position←GetPosition(block_address)
2  static_position←StaticMapping(block_address)
3  last_type←GetType(block_address)
4  if last_type=Leading and type= Following then
5      cache[position].priority_bit←0
6  end
7  if last_type=Following and type= Leading then
8      if position=static_position then
9          cache[position].priority_bit←1
10     else
11         if cache[static_position].type= Leading and
12         cache[static_position].reference_bit = 1 then
13             cache[static_position].reference_bit←0
14         else
15             if IsDirty(cache[static_position]) then
16                 WriteBack(cache[static_position])
17             end
18             cache[static_position]←cache[position]
19             Free(cache[position])
               cache[static_position].priority_bit←1
21         end
22     end
23 end
```

4.3 Replacement Policy

As discussed in Section 3, leading blocks incur more miss penalties than following blocks in terms of latency and bandwidth. The cache replacement algorithm can improve DRAM cache performance by exploiting this observation. The replacement algorithm assigns the high caching priority to leading blocks and the low priority to following blocks. To this end, the priority bit for each data block is introduced. The priority is set to be 1 and 0 for the leading blocks and the following blocks, respectively. In this way, the replacement algorithm attempts to keep leading blocks in the cache longer to amortize its miss penalties. However, the leading blocks can gradually occupy more cache space than the following blocks due to the high priority. For example, there are three leading blocks and two following blocks in the set, as shown in Fig.9. This intensifies the cache space contention with the following blocks. On the other hand, the cache space occupied by the cold leading blocks leads to lower cache utilization without considering the cache temporal locality.

To balance miss penalties and the cache temporal locality, we propose the Range-Variable CLOCK (RV-CLOCK) algorithm, based on the low overhead CLOCK algorithm[29–32]. The CLOCK algorithm uses a reference bit to indicate the recency of a data block. If the reference bits of all the following blocks are set in a cache set, these recently accessed following blocks are chosen as victims for the upcoming following blocks since their priorities are lower than those of the leading blocks. It usually leads to cache thrashing for the following blocks. In order to address this issue, we try to evict a cold leading block to increase the cache space of following blocks if all the following blocks' recency bits are set. Specifically, we run the CLOCK algorithm for all data blocks in a cache set to find victims, without excluding leading blocks, in case that following blocks have high temporal localities.

As shown in Fig.9(b), the following blocks' reference bits can be used to choose victims because it indicates which blocks are involved with the CLOCK algorithm. If all the following blocks' recency bits are set, the rightmost AND gate outputs the logic 1 and all data blocks in the set are victim candidates, as shown in Fig.9(b). Otherwise, the rightmost AND gate, with the output of the logic 0, excludes leading blocks to be victim candidates, and only following blocks can be evicted.

4.4 Managing Block Type Transformation

The mapping policy works well for the data with stable block types, as discussed in Subsection 3.4. However, there still exist some workloads in which the type of data blocks changes frequently. Such transitions lead to a larger number of data block migra-

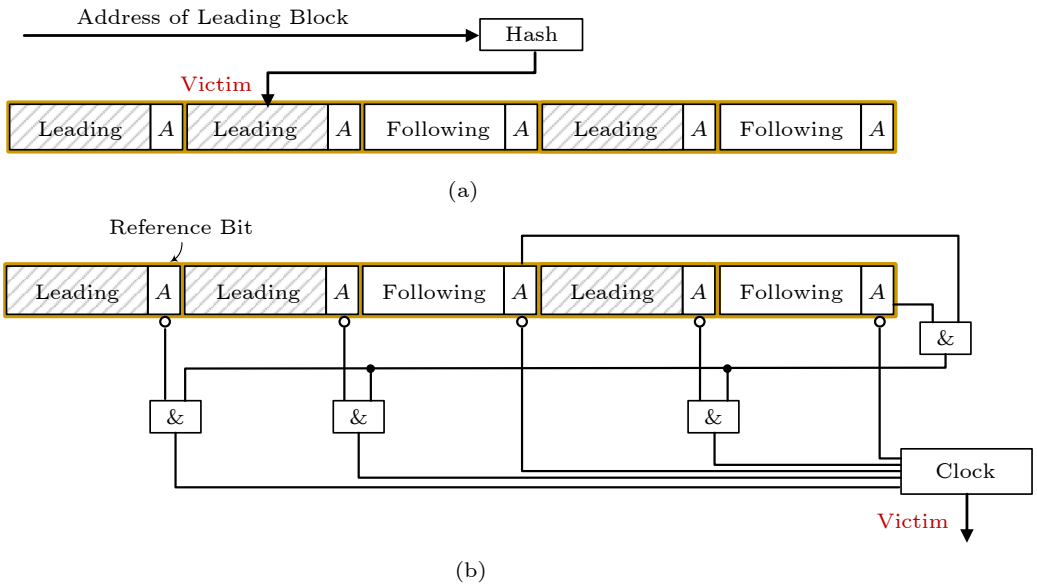


Fig.9. Victim selection policy. (a) Leading blocks occupy more space than following ones. (b) Victim selection upon following block insertion.

tions, degrading cache performance. Furthermore, the mapping policy mistakenly downgrades the blocks' priority and prematurely evicts blocks. For example, if an unstable block switches from a leading block to a following block, the mapping policy resets its priority bit, increasing the eviction probability of the block. At the next active period of the section, it is highly likely that this block will switch back to a leading block. In this example, the mapping policy untimely evicts this leading block, suffering from the costly miss penalty.

To address the issue, we further study block type stability. For a single block, L_{stable} represents the number of occurrences of the consecutive leading or following type. For example, consider a type sequence of {leading → leading → following → following → leading}. Their L_{stable} are 2, 2, 1, respectively. The larger the value of L_{stable} is, the more stable the block maintains the same type. Fig.10 shows the distributions of L_{stable} whose values are no greater than 10. Note that we should only handle workloads whose proportion of type switches is larger than 10% from Fig.10, and other workloads with few type switches can be handled by the default strategy. We observe that about 88% of the type variations occur at $L_{\text{stable}} \leqslant 2$. This implies that these blocks cannot keep their types stable and tend to continuously change their types, and thus there is little consistency in types. Since the percentage of type variations at $L_{\text{stable}} > 2$ is low, we should focus on type variations at $L_{\text{stable}} \leqslant 2$.

Motivated by the above observation, we propose a

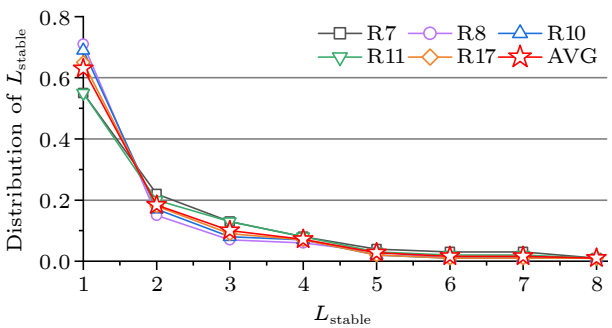


Fig.10. Distributions of block type stability. The points represent the ratios of the stable subsequences with different L_{stable}. Each workload's result is collected from several representative execution intervals, with each interval containing 5 billion instructions.

priority reservation mechanism with a frequent type transition filter to handle unstable blocks. As illustrated in Fig.11, the key idea is to track the blocks whose types frequently change, and set them as leading blocks to avoid potential miss penalties on their following→leading transitions. As discussed above, we mainly focus on the case with $L_{\text{stable}} \leqslant 2$. Specifically, we use a two-bit counter to filter these blocks. If there is a following→leading transition, the block's counter is increased by 2. The counter is decreased by 1 for the following→following transition. When the counter becomes 0, the two-bit counter remains "00" state for following→following transitions. Upon a type transition, P3DC identifies the block as highly unstable and treats it as a leading block if its count is not "00". Otherwise, this block is deemed as a following block. Note that a leading block always keeps its counter's state. This policy eliminates short, isolated

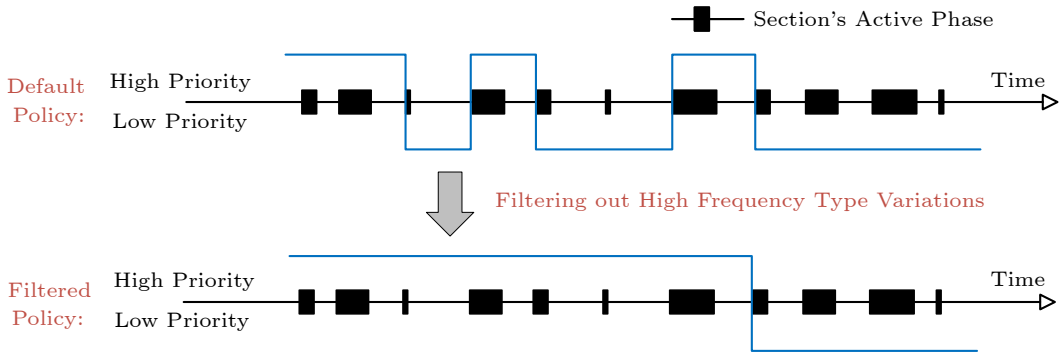


Fig.11. Demonstration of priority reservation mechanism.

type transitions.

4.5 Overhead

Our design leads to extra hardware resources for 1) hash function, 2) victim selection, and 3) data type variation threshold. The hash function (*GetPosition*() in Algorithm 1) involves a simple arithmetic operation to calculate the position for the leading block, and its hardware implementation requires several AND gates, XOR gates, OR gates, and a shift register. To support the victim selection policy (see Fig.9), a reference bit is added for each cache line. Each victim selection logic requires n AND gates, a hand register, and a simple CLOCK block, where n is the associativity of a cache and it is 16 in our design.

The hand register indicates the position of the hand in the CLOCK algorithm, requiring four bits. The CLOCK block includes a 4:16 decoder to indicate the position of the hand, 16 bits mask to specify which ways of cache lines are involved in the victim selection, and FSM to determine the victim. Note that the victim selection logic is only for a bank, rather than a cache set. Typically, there are eight banks in the DRAM cache. To specify the data block type for each way, each cache line needs one bit, which is the priority bit in our design. In addition, to avoid frequent data block type variations, a cache line needs two bits. Thus, each cache line requires three extra bits, which is a part of the tag of a data block. Since a tag has 21 bits (see ATcache[19]), the storage overhead is $3/(\text{size\_of\_tag} + \text{size\_of\_cacheline}) = 3/(21+64\times8) = 0.5\%$ in the DRAM cache. Therefore, our design introduces negligible area overhead. Operations on these introduced parts are performed inside the DRAM cache controller and do not incur expensive data movements. Thus, the energy overhead is negligible.

5 Experimental Methodology

We evaluate the performance of P3DC using Gem5[33], integrated with detailed models of 3D-stacked DRAM and off-chip DRAM[34]. The architectural parameters are summarized in Table 2. We simulate a processor with eight out-of-order cores, sharing 16 MB L2 cache. The simulated DRAM cache is 1 GB and has the same latency as the off-chip DRAM. However, the bandwidth of the DRAM cache is 8 times higher than that of the off-chip DRAM.

Table 2. Architectural Parameters

Hardware	Detailed Parameter
Processor	Out-of-order, 3.2 GHz, 8 cores
L1 I/D cache	32 KB I/D-cache, private, 4-way, 2-cycles
L2 cache	16 MB, Shared, 8-way, 20-cycle, non-inclusive
Tag cache	32 K entries, 8-way, 9-cycle
MAP-I predictor	256 entries, 1-cycle
DRAM cache	1 GB, 1.6 GHz (DDR 3.2 GHz), non-inclusive, 4 channels, 128 bits per channel, 16 banks per rank, 2 KB row buffer, tCAS-tRCD-tRP-tRAS 36-36-36-144 cycles
Main memory	16 GB, 800 MHz (DDR 1.6 GHz), 2 channels, 64 bits per channel, 8 banks per rank, 2 KB row buffer, tCAS-tRCD-tRP-tRAS 36-36-36-144 cycles

5.1 Cache Organizations

We compare P3DC with two state-of-the-art DRAM cache designs: the BEAR cache[21] and the tag data decoupled DEC-A8 cache[22]. The parameters for these three designs are shown in Table 3.

BEAR Cache[21]. We choose BEAR as the baseline of the direct-mapped DRAM cache. BEAR uses a bandwidth-aware bypassing (BAB) scheme to improve the bandwidth efficiency of filling missing entries. Tags and data are placed in DRAM rows in an

Table 3. System Configurations

Design	Mapping Policy	Associativity	Replacement	Tag Prefetch
BEAR[21]	Static	1	BAB	Neighboring tag
DEC-A8[22]	Dynamic	8	CLOCK	Tag batch
P3DC	Static+dynamic	16	RV-CLOCK	Tag batch

interleaving manner. A read request fetches data, as well as the corresponding tag and the next neighboring tag with one additional burst.

DEC-A8 Cache[22]. DEC-A8 is an 8-way set-associative DRAM cache. It decouples tags and data, and stores them in two regions of the DRAM cache. It maps spatially-adjacent cache blocks to the same DRAM row to fully exploit the spatial locality. In addition, DEC-A8 employs a DRAM absence table to implement the cache bypassing mechanism. DEC-A8 concurrently accesses the tag batch and data to reduce the access latency of the DRAM cache in the case of tag cache misses.

We note that although both P3DC and DEC-A8 place tags and data in different DRAM cache regions (or banks), their access policies are different. Specifically, DEC-A8 reads tags and data concurrently in the DRAM cache, regardless of data block types. This access pattern partially decouples the sequential accesses to tags and data by activating data rows in advance, but the remaining data still requires a successful tag comparison before the whole data is completely fetched. As a result, DEC-A8 has higher access latency than BEAR. In contrast, P3DC uses a static mapping policy for leading blocks, and accesses tags and data simultaneously. It shows low access latency similar to BEAR, at the expense of higher bandwidth consumption. For following blocks, the access policy and latency of P3DC is the same as those of DEC-A8.

The above three approaches use the same size of SRAM tag caches. Besides, the DRAM cache presence (DCP) bit[21] is implemented in all designs. The DCP tracks DRAM cache's present state in L2 and reduces the cache bandwidth consumption.

5.2 Workloads

We evaluate the three designs with 18 memory intensive benchmarks from the SPEC CPU2000 and the SPEC CPU2006 benchmark suites①. The benchmarks are classified based on two metrics: spatial locality and cache contention. The spatial locality is measured as $N_{\text{leading}}/N_{\text{total}}$, where N_{leading} and N_{total} are the access number of leading blocks and the total access number respectively. The cache contention is evaluated as the hit rate of the DRAM cache.

These benchmarks can be classified into four categories: contention-dominated (CD), locality-dominated (LD), both-friendly (BF), and non-beneficial (NB). The CD benchmarks have high cache demands with good spatial locality. The set-associative cache is friendly to these benchmarks since it can improve the cache space utilization and does not suffer greatly from tag fetching. The direct-mapped cache is suitable for LD benchmarks due to the shorter hit latency. BF benchmarks have much more cache demands and good spatial locality. Traditional cache organizations can perform well on these benchmarks. NB benchmarks do not work well due to their poor spacial locality and high miss rates.

Table 4 shows 18 workloads in a rate mode (four categories, called Rate-CD, Rate-LD, Rate-BF, Rate-NB in the following results analysis), which means all cores execute the same benchmark. We also evaluate 10 mixed workloads including intra-category and inter-category combinations, as shown in Table 5. In our experiments, we focus on CD workloads and LD workloads, to highlight the impact of the hit rate improvement and hit latency reduction. For each workload, the simulation runs for one billion instructions on each core after fast-forwarding the first ten billion instructions.

Table 4. Workloads form SPEC CU2000 and SPEC CPU2006

Category	Rate Workload with Tickers
Rate-CD	R1: wupwise×8, R2: lucas×8, R3: gap×8, R4: apsi×8, R5: cactusADM×8, R6: lbm×8
Rate-LD	R7: vpr×8, R8: bzip2×8, R9: soplex×8, R10: omnetpp×8, R11: astar×8, R12: xalan×8
Rate-BF	R13: equake×8, R14: mgrid×8, R15: gcc×8, R16: libquantum×8
Rate-NB	R17: GemsFDTD×8, R18: milc×8

①PEC CPU2000 and SPEC CPU2006 are two versions of industry-standardized CPU performance benchmarks which are used to evaluate both integer and floating-point computing performance of general-purpose CPUs. They are available at: https://www.spec.org/cpu2000/ and https://www.spec.org/cpu2006/, respectively.

Table 5. Mixed Workloads with Tickers

Category	Mixed Workload with Tickers
CD	M1: wupwise, luca, gap, apsi, cactusADM, lbm, wupwise, lbm
LD	M2: vpr, bzip2, soplex, omnetpp, astar, xalan, vpr, xalan
BF	M3: {equake, mgrid, gcc, libquantum}×2
NB	M4:{milc, GemsFDTD}×4
CD+LD	M5: wupwise, gap, apsi, cactusADM, vpr, bzip2, soplex, omnetpp
CD+BF	M6: gap, apsi, cactusADM, lbm, equake, mgrid, gcc, libquantum
CD+NB	M7: lucas, apsi, cactusADM, lbm, {milc, GemsFDTD}×2
LD+BF	M8: soplex, omnetpp, astar, xalan, equake, mgrid, gcc, libquantum
LD+NB	M9: vpr, bzip2, astar, xalan, {milc, GemsFDTD}×2
BF+NB	M10: equake, mgrid, gcc, libquantum, {milc, GemsFDTD}×2

6 Experimental Results

6.1 Performance

Fig.12 shows the instructions per cycle (IPC) of three designs, all normalized to BEAR, a direct-mapped cache. For Rate-CD workloads, P3DC outperforms BEAR by 38% on average and up to 66%. The performance of P3DC primarily stems from the hit rate improvement due to the dynamic mapping of its following blocks. The Rate-LD workloads have poor spatial locality, which leads to high miss rates for the tag cache. For these workloads, DEC-A8 shows the worst performance because it causes higher access latency for leading blocks. By applying static mapping to leading blocks, P3DC outperforms DEC-A8 by 8% on average and up to 16%. The Rate-BF workloads have both small working sets and high spatial locality, which is sensitive to different designs. Thus, P3DC outperforms BEAR and DEC-A8 by 6% and 9% on average, respectively. Since the Rate-NB workloads have low temporal locality and large working sets, none of designs perform well on these workloads. For mixed workloads, P3DC and DEC-A8 generally show higher performance than BEAR except for Mix3, Mix8 and Mix 10. Overall, P3DC outperforms BEAR and DEC-A8 by 12% and 6% on average, respectively.

6.2 DRAM Cache Hit Rate

Fig.13 shows the DRAM cache hit rates of all systems. For Rate-CD workloads, BEAR achieves relatively low hit rates because its static data mapping

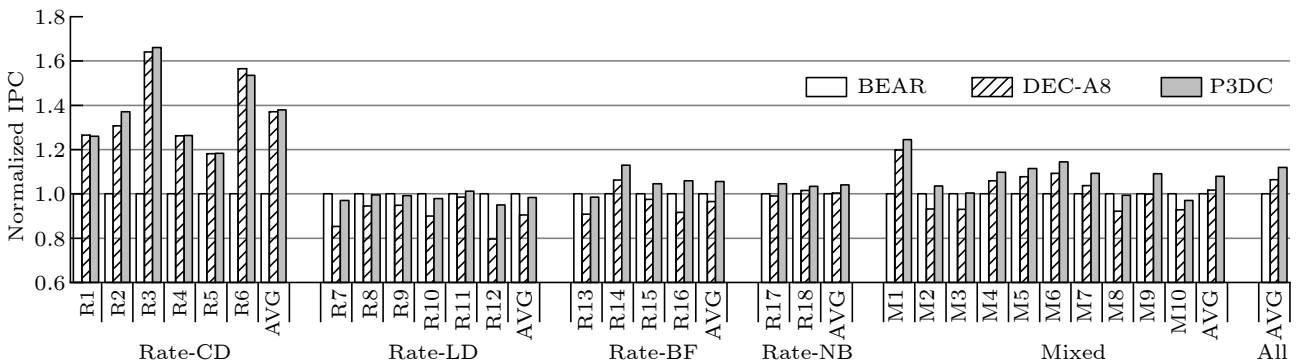


Fig.12. Normalized IPC. All results are normalized to BEAR.

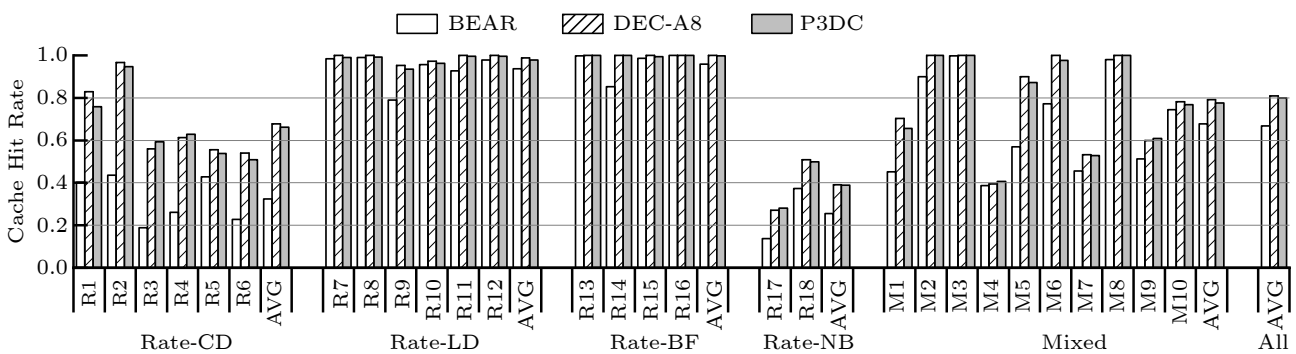


Fig.13. Hit rates of the DRAM cache in three designs.

limits the flexibility of block replacement. Compared with BEAR, P3DC increases the average hit rate by 32% to 63%. We note that the average cache hit rate for all workloads in P3DC is only 3% less than that of DEC-A8. This implies applying dynamic mapping for following blocks can offer good flexibility for block replacement, and the leading blocks can be mapped statically without sacrificing hit rates. In addition, P3DC adopts a static mapping policy with lower access latency for leading blocks, and thus improves the IPC by 1.7% compared with DEC-A8, as shown in Fig.12.

All the three designs achieve high hit rates for Rate-LD workloads, which have small working sets to fit in the DRAM cache. However, these workloads have poor spatial locality, leading to large miss rates in the tag cache on average, and introducing a large ratio of leading blocks. The performance of the cache system is dominated by the hit latency of the DRAM cache. For DEC-A8, all requests are served sequentially in a data-after-tag manner. In contrast, P3DC directly accesses leading blocks in their statically-mapped positions, and meanwhile fetches the corresponding tags concurrently. On average, 69% data requests are served by concurrent tags and data fetching in P3DC, which is very close to that of BEAR.

The working set of Rate-BF workloads can also fit in the DRAM cache, while offering high spatial locality. The tag batching mechanisms in P3DC and DEC-A8 achieve higher hit rates of the tag cache. But this does not imply higher performance for the set-associative cache DEC-A8, because the direct-mapped cache BEAR offers the same latency in the case of a DRAM cache hit. On the other hand, a small portion of requests still need to access the tags in the DRAM cache. These requests slightly degrade the performance of DEC-A8 by about 4%.

Due to the large working set and the poor temporal locality, Rate-NB workloads have low hit rates of the DRAM cache and are insensitive to data mapping policies. Therefore, all the three designs have similar request distributions and achieve a similar IPC for these workloads, as shown in Fig.12.

The mixed workloads M1, M5 and M6 provide optimization opportunities in both hit rate improvement and hit latency reduction. For these workloads, P3DC outperforms BEAR and DEC-A8 by 16% and 4% on average, respectively. Since the workloads M3 and M8 have extremely high hit rates in the DRAM cache (over 99%), P3DC and BEAR achieve similar performance, but higher performance than DEC-A8.

6.3 Impact of Tag Fetching Modes on Hit Latency

For LD workloads (R7–R12, M2) with poor spatial locality, the performance of the cache system is sensitive to the access latency of leading blocks. BEAR and DEC-A8 only use a single data mapping policy (static or dynamic), where neither leading nor following blocks are taken into account. In contrast, P3DC applies static mapping to leading blocks and dynamic mapping to following blocks. All three designs can concurrently fetch tags and data. To serve a leading block, P3DC issues two parallel DRAM cache requests for a tag batch and a data block, respectively. To co-locate a tag data pair, BEAR issues a request to fetch a tag-and-data entity and this fetching incurs an additional burst compared with fetching a data block. Although DEC-A8 can also issue two DRAM cache access requests concurrently, it has a higher access latency because the process of reading tags and data cannot be coordinated fairly, as discussed in Subsection 5.1.

Fig.14 shows hit latencies of three designs, all normalized to BEAR which shows the lowest hit latency. The cache hit latency of DEC-A8 is 1.41x on average and up to 1.55x higher than that of BEAR. Due to the queuing delay of P3DC, not all the tag and data requests can be done at the same time exactly. As a result, the hit latency of P3DC is 1.17x as high as that of BEAR on average. For LD workloads, the access latency of leading blocks directly affects the overall system performance. As shown in Fig.12, BEAR achieves an 11% higher IPC (on average) than DEC-A8, while P3DC can lower this performance gap to 3% with our sophisticated data mapping policy.

High hit rates benefit system performance in two ways. Firstly, requests hiting the DRAM cache can

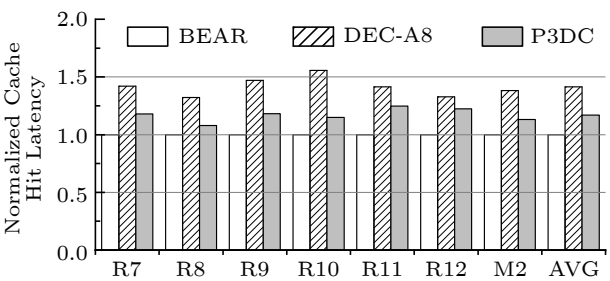


Fig.14. Normalized cache hit latency of locality-dominated workloads.

benefit from the high bandwidth of the die-stacking DRAM. Secondly, high hit rates can help to reduce the access latency of off-chip DRAM. Due to the filtering effect of the DRAM cache, a large portion of off-chip DRAM accesses are eliminated, resulting in relatively low queuing delay in off-chip DRAM.

6.4 Impact of Hit Rate Improvement on Queuing Latency of Off-Chip DRAM

Fig.15 compares BEAR and P3DC in terms of off-chip DRAM queuing latency for CD workloads. The portions labeled with left slashes in the left bars and right bars are the queuing latencies of the BEAR cache and the P3DC cache, respectively. Fig.15 also shows the hit rates with red lines. We can find that a low hit rate of the DRAM cache makes CD workloads suffer from a high queuing latency. In the worst case, i.e., R3, the queuing latency is up to 58% of the overall off-chip DRAM access latency. By improving the hit rate of the DRAM cache, P3DC reduces the queuing latency by 49% on average and up to 78% compared with BEAR.

6.5 Bandwidth Consumption

Fig.16 shows the DRAM cache bandwidth consumption of the three designs, including tag read, tag write-back, data read hit, data read miss, data fill, and data write-back. Data read miss is the case that the data is fetched concurrently with the tag, but turns out to be a DRAM cache miss. Among the three designs, BEAR shows the best bandwidth efficiency. P3DC and DEC-A8 consume more bandwidth than BEAR in tag read and tag write-back.

Tag read is the sum of on-demand tag fetching and tag prefetching. P3DC and DEC-A8 adopt the tag batching mechanism that fetches the whole set of tags in a single request. BEAR only fetches two tags at the same time: the requested tag along with one neighboring tag. In general, the tag batching helps to improve the hit rate of the tag cache. However, for workloads with poor spatial locality, this mechanism wastes bandwidth. For example, for the Rate-LD workload group, the bandwidth consumed by the tag read in P3DC and DEC-A8 is 7x and 8x larger than that in BEAR. DEC-A8 does not consume band-

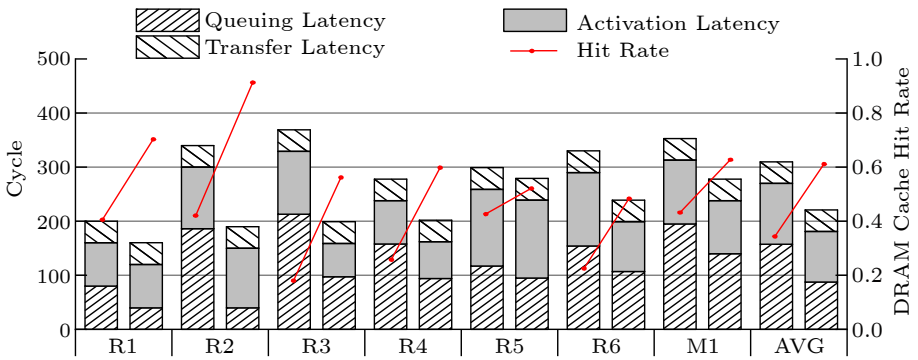


Fig.15. Impact of DRAM cache hit rates on off-chip DRAM queuing delay.

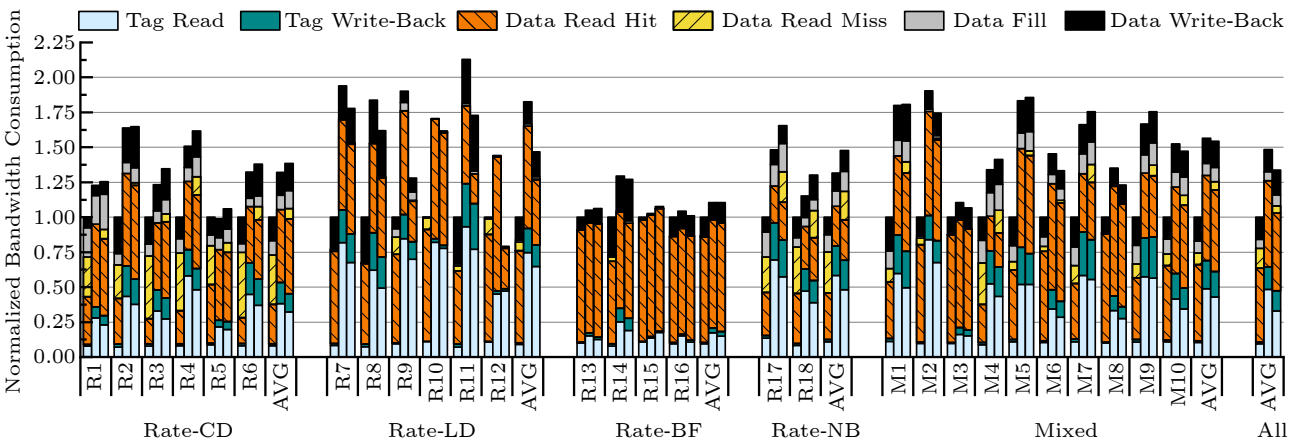


Fig.16. Normalized bandwidth consumption. From left to right, the three bars in each group are the results of BEAR, DEC-A8, and P3DC, respectively.

width upon data read misses because it only reads data if the corresponding tag is matched successfully. In contrast, BEAR fetches both tags and data concurrently, which may incur data read misses if the tag is not matched. A similar situation may occur when P3DC accesses leading blocks.

All the three designs adopt a DRAM bypassing scheme to reduce the bandwidth consumption on miss fills. They do not fill the missed data block to the DRAM cache after speculating the missed data block with low reuse probability. Filling all the missed data to the DRAM cache can improve hit rates in the case of good temporal locality. However, it may consumes more bandwidth in the case of poor temporal locality. On average, as shown in the rightmost group, P3DC consumes 1.36x more bandwidth than BEAR. The results imply that there is still large room for improving P3DC's performance towards better bandwidth utilization in the future.

6.6 High Frequency Type Variation Filter

P3DC relies on the high-frequent type variation filter to track unstable data blocks. Fig.17 shows the percentage of the identified cases with different L_{stable}. We can find that the filter has high resistance against short L_{stable}. The filtering rate of L_{stable}=1 and L_{stable}=2 are 92% and 95%, respectively. Meanwhile, it passes through all segments with $L_{\text{stable}} > 3$.

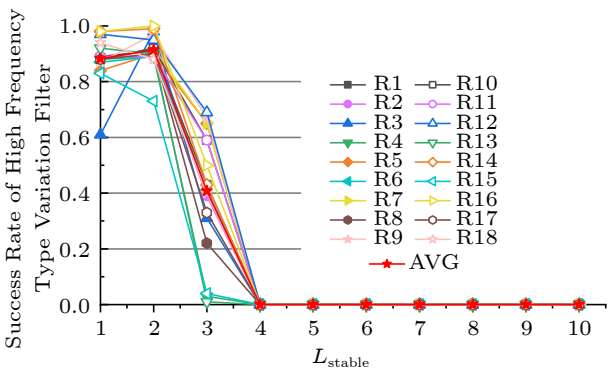


Fig.17. Success rate of the high frequency type variation filter.

7 Related Work

7.1 Set-Associative and Direct-Mapped Caches

Set-Associative Caches. Loh-Hill *et al.*[7, 8] proposed to organize each DRAM row (2 KB) as a 29-way set-associative cache, with good replacement flexibility. Since the tag and data are stored in the same row, the data is streamed out right after the tag read, getting the benefits of row-buffer hit. Bi-Modal Cache[28] and Unison Cache[35] try to take advantage of large-sized cache lines to exploit the spatial locality, and exploit the set-associative structure to mitigate cache conflicts.

Direct-Mapped Caches. Alloy Cache[9] breaks the data-after-tag serialization by co-locating a data block and its tag in the direct-mapped DRAM cache. It issues a command to fetch a data block and its tag together at the expense of one additional burst. CAMEO[36] further improves Alloy Cache by removing the data copy between the 3D DRAM cache and the off-chip memory.

Some studies[7–9, 28, 35, 36] proposed different optimizations on the set-associative DRAM cache or the direct-mapped DRAM cache. In this paper, we reveal that different data blocks have a significant impact on cache hit latency, and then propose a partial direct-mapped cache design to achieve both the high hit rates of the set-associative structure, and the low hit latency of the direct-mapped structure.

7.2 Other Cache Optimization Technologies

Bandwidth Optimization. Based on the direct-mapped structure, BEAR[21] focuses on reducing bandwidth consumption. It puts forward several methods such as the bandwidth-aware bypassing for miss fills, DRAM cache presence bit for write back probe, and neighboring tag cache for miss probe. Moreover, Mostly-Clean Cache[37] balances the bandwidth of the DRAM cache and the off-chip DRAM by fully utilizing the off-chip bandwidth when the DRAM cache is serving a burst of cache hits. TicToc[38] reduces memory bandwidth consumption by caching the tag of recently-accessed DRAM to the last level of cache. Red Cache[17] monitors data block accesses at runtime with low-cost counters, and stores blocks with frequently-reused or high-bandwidth requirements in the DRAM cache to improve the bandwidth efficiency and performance.

Data Prefetching. Data prefetching is another key technique for improving cache hit rates. A major approach is to apply page-granularity prefetching[39, 40]. However, it results in a waste of the bandwidth because the fetched pages may contain unused data.

Footprint Cache[10] mitigates this issue by tracking the workload's access footprints[41, 42] and prefetches the cache lines that have a high possibility to be accessed in the DRAM cache. F-TDC[43] integrates the footprint history table into the page table and hides the cache probes by TLB lookups.

Tag Prefetching. ATCache[19] accelerates the probe operation by introducing a small tag cache in SRAM. On the discovery that a tag access has spatial locality, it prefetches tags from nearby cache sets, and takes advantage of a clever on-demand tag prefetching scheme to reduce tag cache pollution. Tsukada *et al.*[44] found that the address offset between two consecutive memory accesses is almost the same. Based on this observation, they prefetched the corresponding tags to reduce the tag access latency and improve performance.

Way Prediction. Accord[45] and SODA[46] use two-way set-associative cached, but still achieve high cache hit rates. Accord uses historical information to guide the insertion and prediction of cache lines, while SODA exploits a way-locater cache in SRAM to store the location of the data in cache sets.

Tag Decoupled. Decoupled Fused Cache (DFC)[47] takes advantage of the redundancy of tags in the LLC and use a LLC tag array to store the location of data in the DRAM cache. By fusing the tags of the DRAM cache with tags of the LLC, the cost of tag access is mitigated.

The above proposals are orthogonal to our work that focuses on the data mapping structure. Our techniques can be applied to these schemes, and further improve the performance of the DRAM cache.

8 Conclusions

In this paper, we presented P3DC, a partial direct-mapped die-stacked DRAM cache. The P3DC cache classifies data blocks into leading blocks and following blocks, and places them with static mapping and dynamic mapping respectively in a unified set-associative structure. The key idea is inspired by the observation that different block types have different impacts on the cache hit rate and the cache hit latency. P3DC combines the advantages of direct-mapped caches and set-associative caches to achieve low hit latency while maintaining high cache hit rates through partial-direct mapping. In this way, P3DC is able to perform more efficiently in the presence of ambiguous application behavior or variable access characteristics. Experimental results demonstrated that, P3DC can reduce the cache hit latency by 20.5% and can achieve comparable hit rates with set-associative caches. P3DC improves the IPC by 12% on average and up to 66% compared with the direct-mapped BEAR cache, and 6% on average and up to 19% compared with the set-associative DEC-A8 cache. Particularly, the P3DC cache is orthogonal to prior state-of-the-art DRAM cache designs and can be applied to further performance improvement.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Jun H, Cho J, Lee K, Son H Y, Kim K, Jin H, Kim K. HBM (high bandwidth memory) DRAM technology and architecture. In *Proc. the 2017 IEEE International Memory Workshop (IMW)*, May 2017, pp.1–4. DOI: 10.1109/IMW.2017.7939084.

[2] Hadidi R, Asgari B, Mudassar B A, Mukhopadhyay S, Yalamanchili S, Kim H. Demystifying the characteristics of 3D-stacked memories: A case study for hybrid memory cube. In *Proc. the 2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct. 2017, pp.66–75. DOI: 10.1109/IISWC.2017.8167757.

[3] Shahab A, Zhu M, Margaritov A, Grot B. Farewell my shared LLC! A case for private die-stacked DRAM caches for servers. In *Proc. the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2018, pp.559–572. DOI: 10.1109/MICRO.2018.00052.

[4] Volos S, Jevdjic D, Falsafi B, Grot B. Fat caches for scale-out servers. *IEEE Micro*, 2017, 37(2): 90–103. DOI: 10.1109/MM.2017.32.

[5] Nassif N, Munch A O, Molnar C L, Pasdast G, Lyer S V, Yang Z, Mendoza O, Huddart M, Venkataraman S, Kandula S, Marom R, Kern A M, Bowhill B, Mulvihill D R, Nimmagadda S, Kalidindi V, Krause J, Haq M M, Sharma R, Duda K. Sapphire rapids: The next-generation intel Xeon scalable processor. In *Proc. the 17th IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2022, pp.44–46. DOI: 10.1109/ISSCC42614.2022.9731107.

[6] Zahran M. The future of high-performance computing. In *Proc. the 17th International Computer Engineering Conference (ICENCO)*, Dec. 2021, pp.129–134. DOI: 10.1109/ICENCO49852.2021.9698918.

[7] Loh G H, Hill M D. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2011, pp.454–464. DOI: 10.1145/2155620.2155673.

[8] Loh G, Hill M D. Supporting very large DRAM caches with compound-access scheduling and MissMap. *IEEE Micro*, 2012, 32(3): 70–78. DOI: 10.1109/MM.2012.25.

[9] Qureshi M K, Loh G H. Fundamental latency trade-off in architecting dram caches: Outperforming impractical SRAM-tags with a simple and practical design. In *Proc. the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2012, pp.235–246. DOI: 10.1109/MICRO.2012.30.

[10] Jevdjic D, Volos S, Falsafi B. Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? Have it all with footprint cache. *ACM SIGARCH Computer Architecture News*, 2013, 41(3): 404–415. DOI: 10.1145/2508148.2485957.

[11] Shin D, Jang H, Oh K, Lee J W. An energy-efficient dram cache architecture for mobile platforms with PCM-based main memory. *ACM Trans. Embedded Computing Systems (TECS)*, 2022, 21(1): 1–22. DOI: 10.1145/3451995.

[12] Zhang Q, Sui X, Hou R, Zhang L. Line-coalescing DRAM cache. *Sustainable Computing: Informatics and Systems*, 2021, 29: 100449. DOI: 10.1016/j.suscom.2020.100449.

[13] Zhou F, Wu S, Yue J, Jin H, Shen J. Object Fingerprint Cache for Heterogeneous Memory System. *IEEE Transactions on Computers*, 2023, 72(9): 2496–2507. DOI: 10.1109/TC.2023.3251852.

[14] Chi Y, Yue J, Liao X, Liu H, Jin H. A hybrid memory architecture supporting fine-grained data migration. *Frontiers of Computer Science*, 2024, 18(2): 182103. DOI: 10.1007/s11704-023-2675-y.

[15] Hameed F, Bauer L, Henkel J. Architecting on-chip DRAM cache for simultaneous miss rate and latency reduction. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2016, 35(4): 651–664. DOI: 10.1109/TCAD.2015.2488488.

[16] Hameed F, Bauer L, Henkel J. Simultaneously optimizing DRAM cache hit latency and miss rate via novel set mapping policies. In *Proc. the 16th International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Sept. 29–Oct. 4, 2013. DOI: 10.1109/CASES.2013.6662515.

[17] Behnam P, Bojnordi M N. Adaptively reduced DRAM caching for energy-efficient high bandwidth memory. *IEEE Trans. Computers*, 2022, 71(10): 2675–2686. DOI: 10.1109/TC.2022.3140897.

[18] Kumar S, Zhao H, Shriraman A, Matthews E, Dwarkadas S, Shannon L. Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy. In *Proc. the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2012, pp.376–388. DOI: 10.1109/MICRO.2012.42.

[19] Huang C C, Nagarajan V. ATCache: Reducing DRAM cache latency via a small SRAM tag cache. In *Proc. the 23rd International Conference on Parallel Architectures and Compilation (PACT)*, Aug. 2014, pp.51–60. DOI: 10.1145/2628071.2628089.

[20] Hameed F, Bauer L, Henkel J. Reducing latency in an SRAM/DRAM cache hierarchy via a novel tag-cache architecture. In *Proc. the 51st Annual Design Automation Conference (DAC)*, Jun. 2014. DOI: 10.1145/2593069.2593197.

[21] Chou C, Jaleel A, Qureshi M K. BEAR: Techniques for mitigating bandwidth bloat in gigascale DRAM caches. *ACM SIGARCH Computer Architecture News*, 2015, 43(3S): 198–210. DOI: 10.1145/2872887.2750387.

[22] Hameed F, Khan A A, Castrillon J. Improving the performance of block-based DRAM caches via tag-data decoupling. *IEEE Trans. Computers*, 2021, 70(11): 1914–1927. DOI: 10.1109/TC.2020.3029615.

[23] Kawano M, Wang X Y, Ren Q, Loh W L, Rao B C, Chui K J. One-step TSV process development for 4-layer wafer stacked DRAM. In *Proc. the 71st IEEE Electronic Components and Technology Conference (ECTC)*, Jun. 1–Jul. 4, 2021, pp.673–679. DOI: 10.1109/ECTC32696.2021.00117.

[24] Jiang X, Zuo F, Wang S, Zhou X, Wang Y, Liu Q, Ren Q, Liu M. A 1596-GB/s 48-Gb stacked embedded DRAM 384-core SoC with hybrid bonding integration. *IEEE Solid-State Circuits Letters*, 2022, 5: 110–113. DOI: 10.1109/LSSC.2022.3171862.

[25] Bose B, Thakkar I. Characterization and mitigation of electromigration effects in TSV-based power delivery network enabled 3D-stacked DRAMs. In *Proc. the 31st Great Lakes Symposium on VLSI*, Jun. 2021, pp.101–107. DOI: 10.1145/3453688.3461503.

[26] Agarwalla B, Das S, Sahu N. Process variation aware DRAM-Cache resizing. *Journal of Systems Architecture*, 2022, 123: 102364. DOI: 10.1016/j.sysarc.2021.102364.

[27] Cheng W, Cai R, Zeng L, Feng D, Brinkmann A, Wang Y. IMCI: An efficient fingerprint retrieval approach based on 3D stacked memory. *Science China Information Sciences*, 2020, 63: 179101. DOI: 10.1007/s11432-019-2672-5.

[28] Gulur N, Mehendale M, Manikantan R, Govindarajan R. Bi-modal DRAM cache: Improving hit rate, hit latency and bandwidth. In *Proc. the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2014, pp.38–50. DOI: 10.1109/MICRO.2014.36.

[29] Jiang S, Chen F, Zhang X. CLOCK-Pro: An effective improvement of the CLOCK replacement. In *Proc. the 2005 Annual Conference on USENIX Annual Technical Conference*, Apr. 2005.

[30] Janapsatya A, Ignjatović A, Peddersen J, Parameswaran S. Dueling CLOCK: Adaptive cache replacement policy based on the CLOCK algorithm. In *Proc. the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, Mar. 2010, pp.920–925. DOI: 10.1109/

DATE.2010.5456920.

[31] Bansal S, Modha D S. CAR: Clock with adaptive replacement. In *Proc. the 3rd USENIX Conference on File and Storage Technologies (FAST)*, Mar. 2004, pp.187–200.

[32] Li C. CLOCK-pro+: Improving CLOCK-pro cache replacement with utility-driven adaptation. In *Proc. the 12th ACM International Conference on Systems and Storage (SYSTOR)*, May 2019, pp.1–7. DOI: 10.1145/3319647.3325838.

[33] Binkert N, Beckmann B, Black G, Reinhardt S K, Saidi A, Basu A, Hestness J, Hower D R, Krishna T, Sardashti S, Sen R, Sewell K, Shoaib M, Vaish N, Hill M D, Wood D A. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011, 39(2): 1–7. DOI: 10.1145/2024716.2024718.

[34] Poremba M, Xie Y. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *Proc. the 2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Aug. 2012, pp.392–397. DOI: 10.1109/ISVLSI.2012.82.

[35] Jevdjic D, Loh G H, Kaynak C, Falsafi B. Unison cache: A scalable and effective die-stacked DRAM cache. In *Proc. the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2014, pp.25–37. DOI: 10.1109/MICRO.2014.51.

[36] Chou C C, Jaleel A, Qureshi M K. CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *Proc. the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2014, pp.1–12. DOI: 10.1109/MICRO.2014.63.

[37] Sim J, Loh G H, Kim H, OConnor M, Thottethodi M. A mostly-clean DRAM cache for effective hit speculation and self-balancing dispatch. In *Proc. the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2012, pp.247–257. DOI: 10.1109/MICRO.2012.31.

[38] Young V, Chishti Z A, Qureshi M K. TicToc: Enabling bandwidth-efficient DRAM caching for both hits and misses in hybrid memory systems. In *Proc. the 37th IEEE International Conference on Computer Design (ICCD)*, Nov. 2019, pp.341–349. DOI: 10.1109/ICCD46524.2019.00055.

[39] Zhang M, Kim J G, Yoon S K, Kim S D. Dynamic recognition prefetch engine for DRAM-PCM hybrid main memory. *The Journal of Supercomputing*, 2022, 78(2): 1885–1902. DOI: 10.1007/s11227-021-03948-5.

[40] Choi S G, Kim J G, Kim S D. Adaptive granularity based last-level cache prefetching method with eDRAM prefetch buffer for graph processing applications. *Applied Sciences*, 2021, 11(3): 991. DOI: 10.3390/app11030991.

[41] Kilic O O, Tallent N R, Friese R D. Rapid memory footprint access diagnostics. In *Proc. the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Aug. 2020, pp.273–284. DOI: 10.1109/ISPASS48437.2020.00047.

[42] Oh Y S, Chung E Y. Energy-efficient shared cache using way prediction based on way access dominance detection. *IEEE Access*, 2021, 9: 155048–155057. DOI: 10.1109/ACCESS.2021.3126739.

[43] Jang H, Lee Y, Kim J, Kim Y, Kim J, Jeong J, Lee J W. Efficient footprint caching for Tagless DRAM Caches. In *Proc. the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Mar. 2016, pp.237–248. DOI: 10.1109/HPCA.2016.7446068.

[44] Tsukada S, Takayashiki H, Sato M, Komatsu K, Kobayashi H. A metadata prefetching mechanism for hybrid memory architectures. *IEICE Trans. Electronics*, 2022, E105.C(6): 232–243. DOI: 10.1587/transele.2021LHP0004.

[45] Young V, Chou C, Jaleel A, Qureshi M. ACCORD: Enabling associativity for gigascale DRAM caches by coordinating way-install and way-prediction. In *Proc. the 45th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp.328–339. DOI: 10.1109/ISCA.2018.00036.

[46] Chen P, Yue J, Liao X, Jin H. Trade-off between hit rate and hit latency for optimizing DRAM cache. *IEEE Trans. Emerging Topics in Computing*, 2021, 9(1): 55–64. DOI: 10.1109/TETC.2018.2800721.

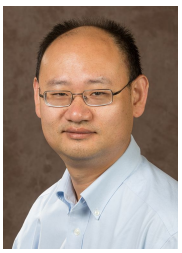
[47] Vasilakis E, Papaefstathiou V, Trancoso P, Sourdis I. Decoupled fused cache: Fusing a decoupled LLC with a DRAM cache. *ACM Trans. Architecture and Code Optimization (TACO)*, 2018, 15(4): 65. DOI: 10.1145/3293447.

Ye Chi received his Ph.D. degree in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, in 2023. He is now working at the School of Big Data and Internet, Shenzhen Technology University (SZTU), Shenzhen. His search interests are in the areas of computer architecture, die-stacked DRAM, in-memory computing, hybrid memory system architecture and memory pooling.

Ren-Tong Guo received his B.E. degree in software engineering from Xi'an University of Science and Technology, Xi'an, in 2011, and his Ph.D. degree in computer science and engineering from Huazhong University of Science and Technology (HUST), Wuhan, in 2017. His research interests are in the areas of caching systems and distributed systems.

Xiao-Fei Liao is a professor in the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan. He received his Ph.D. degree in computer science and engineering from HUST, Wuhan, in 2005. His research interests are in the areas of system software, P2P system, cluster computing, and streaming services.

Hai-Kun Liu received his Ph.D. degree in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, in 2012. He is a professor at the School of Computer Science and Technology, HUST, Wuhan. His current research interests include in-memory computing, virtualization technologies, cloud computing, and distributed systems.



Jianhui Yue received his Ph.D. degree from the University of Maine, Orono, in 2012. He is an assistant professor of the Computer Science Department, Michigan Technological University, Michigan. His research interests include computer architecture and systems.