Morphable DRAM Cache Design for Hybrid Memory Systems

SANGHOON CHA, BOKYEONG KIM, CHANG HYUN PARK, and JAEHYUK HUH,
KAIST, Republic of Korea

DRAM caches have emerged as an efficient new layer in the memory hierarchy to address the increasing diversity of memory components. When a small amount of fast memory is combined with slow but large memory, the cache-based organization of the fast memory can provide a SW-transparent solution for the hybrid memory systems. In such DRAM cache designs, their effectiveness is affected by the bandwidth and latency of both fast and slow memory. To quantitatively assess the effect of memory configurations and application patterns on the DRAM cache designs, this article first investigates how three prior approaches perform with six hybrid memory scenarios. From the investigation, we observe no single DRAM cache organization always outperforms the other organizations across the diverse hybrid memory configurations and memory access patterns. Based on this observation, this article proposes a reconfigurable DRAM cache design that can adapt to different hybrid memory combinations and workload patterns. Unlike the fixed tag and data arrays of conventional on-chip SRAM caches, this study advocates to exploit the flexibility of DRAM caches, which can store tags and data to DRAM in any arbitrary way. Using a sample-based mechanism, the proposed DRAM cache controller dynamically finds the best organization from three candidates and applies the best one by reconfiguring the tags and data layout in the DRAM cache. Our evaluation shows that the proposed morphable DRAM cache can outperform the fixed DRAM configurations across six hybrid memory configurations.

CCS Concepts: • **Computer systems organization** → **Architectures**; • **Hardware** → *Memory and dense storage*;

Additional Key Words and Phrases: DRAM cache, hybrid memory systems, reconfigurable cache design, high bandwidth memory, nonvolatile memory

ACM Reference format:
Sanghoon Cha, Bokyeong Kim, Chang Hyun Park, and Jaehyuk Huh. 2019. Morphable DRAM Cache Design for Hybrid Memory Systems. *ACM Trans. Archit. Code Optim.* 16, 3, Article 31 (July 2019), 24 pages.
https://doi.org/10.1145/3338505

This work was supported by Samsung Electronics (DRAM-NAND Hybrid Memory Architecture Research), National Research Foundation of Korea (NRF-2019R1A2B5B01069816) and by the Institute for Information and Communications Technology Promotion (IITP-2017-0-00466). The second and third grants are funded by the Ministry of Science and ICT, Korea.
Authors' address: S. Cha, B. Kim, C. H. Park, and J. Huh, School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea; emails: {shcha, bokyeong, changhyunpark}@calab.kaist.ac.kr, jhhuh@kaist.ac.kr.

1544-3566/2019/07-ART31
https://doi.org/10.1145/3338505

1 INTRODUCTION

The advent of new DRAM technologies, such as 3D stacked memory (Lee et al. 2014; Pawlowski 2011) and reduced-latency DRAM (RL-DIMM) (Micron 2016), has been increasing the heterogeneity of DRAM. In addition to the diversification of DRAM technologies, new non-volatile memory technologies have emerged to complement DRAM as high capacity memory (Wong et al. 2010). Such increasing heterogeneity in memory components has enabled a composite memory system consisting of more than two types of different memory technologies, such as hybrid HBM-DDR (Sodani et al. 2016) or DDR-NVM (Qureshi et al. 2009) memory systems. Commonly, such a composite or hybrid memory system consists of low latency and high bandwidth near memory (fast memory) and capacity-oriented far memory (slow memory). Based on the temporal and spatial locality of data, data are migrated between the two memory types, to provide fast accesses to frequently used data (Chou et al. 2014, 2015, 2016; Huang and Nagarajan 2014; Jang et al. 2016; Jevdjic et al. 2014, 2013; Jiang et al. 2010; Lee et al. 2015; Loh 2009; Loh and Hill 2011; Qureshi and Loh 2012; Sim et al. 2014; Volos et al. 2017; Yu et al. 2017).

Among various approaches to organize the hybrid memory, one of the promising techniques is to use the fast memory as a memory-based cache. Such DRAM-based caches can provide a SW-transparent performance improvement with the HW-controlled data mapping and migration between the DRAM cache and slow backing memory (Jevdjic et al. 2014, 2013; Loh and Hill 2011; Qureshi and Loh 2012). Such DRAM cache organizations have advanced to reduce latencies, while improving hit rates of DRAM caches. DRAM caches commonly store both tags and data in DRAM instead of a separate SRAM array for tags, as using SRAM tags requires a large SRAM storage for increasing DRAM cache capacity.

The main differences among the prior DRAM cache designs stem from how tags are organized. Unlike the traditional on-chip SRAM caches, accessing tags is costly as they are also stored in the relatively slow DRAM. Due to the tag access limitation, the prior DRAM cache organizations have tradeoffs between the associativity and hit latency. For example, LH cache supports 29 ways, as it places 29 data blocks and tags in a DRAM row (Loh and Hill 2011). However, to support such a high associativity, it requires one to access DRAM three times for tag accesses only. On the other hand, Alloy cache is a direct-mapped cache, and allows a single access to retrieve both tag and data with an extended 72B interface (Qureshi and Loh 2012). In addition, the block size can pose another tradeoff. A large block can reduce the tag overhead and amortize the migration cost, but it can waste the capacity of the fast memory, if spatial locality is low (Jevdjic et al. 2014, 2013).

Motivated by the tradeoffs in the prior DRAM cache designs, this article first investigates the effective DRAM cache organizations for several different hybrid memory scenarios. Using HBM, DDR, PCM, and NAND technologies, this article uses six different hybrid memory systems, and evaluates the prior approaches with a range of applications on the six memory configurations. Unlike the results from the prior work, the article reports that the best DRAM cache organization is highly affected by the hybrid memory configurations as well as the memory access patterns of applications. Our evaluation found that no single design is superior to the other designs across the diverse application workloads and hybrid memory configurations.

However, one of the advantages of DRAM caches over the conventional on-chip SRAM caches is its flexibility. Tags and data are stored in DRAM rows, unlike the separate fixed SRAM arrays for tags and data in conventional caches. The organization of DRAM caches can be controlled by the DRAM cache controller without changing the DRAM chip itself. As the DRAM cache controller mandates the layout of data and tags in rows of DRAM, the DRAM cache organization can be reconfigured to match the current workload characteristics and hybrid configuration.

To exploit the flexibility of DRAM cache configuration, this article proposes a new Morphable DRAM Cache (MDC) design. The MDC design reconfigures the tag and data organizations to minimize the memory access time, considering the characteristics of fast and slow memory components in addition to the application behaviors. A key mechanism to support the reconfigurability is the identification of the best DRAM cache organization. This study uses three organizations as candidate cache organizations. Each configuration has different associativities or block sizes. A small subset of DRAM rows are fixed to each candidate design, and their effectiveness is measured online. One important technique for the sampling mechanism is to isolate the bandwidth effect of sampled sets from the rest of sets, as the bandwidth of DRAM cache is a critical factor selecting the best design.

To the best of our knowledge, this study is the first study to propose a morphable organization for DRAM caches, exploiting its flexibility. Our simulation-based evaluation shows that the dynamic reconfiguration can find the best DRAM configuration for a given hybrid memory system and workload. MDC can outperform three fixed DRAM cache organizations by 16.7%, 20.2%, and 29.8%.

The key contributions of the article are as follows:

—It reports that the best DRAM cache design can be affected by the application behaviors and hybrid memory characteristics.
—It proposes the first reconfigurable DRAM cache design, which can adapt to the memory characteristics and application behaviors.
—It provides an efficient sample-based selection and reconfiguration mechanism for the morphable DRAM cache. Its support for bandwidth isolation is critical for accurate decision.

The rest of the article is organized as follows. Section 2 presents the background in the prior DRAM cache designs and the hybrid memory configurations this article explores. Section 3 presents the tradeoffs in the prior DRAM cache designs with diverse hybrid memory configurations. Section 4 presents the architecture of the morphable DRAM cache with its selection and reconfiguration mechanism. Section 5 presents the evaluation results. Section 6 discusses possible optimizations to this work, Section 7 presents the related work, and Section 8 concludes the article.

2 BACKGROUND

This section presents the prior DRAM cache organizations and their potential tradeoffs. Considering their distinct characteristics, it introduces three representative organizations we use for this article. In addition, it discusses possible hybrid memory organizations with different DRAM and NVM technologies.

2.1 DRAM Cache Organization

A key design issue with DRAM caches is the storage and management of cache tags. The most simple organization of the DRAM cache tags is to use an on-chip SRAM structure. The SRAM tag lookup allows quick checks of hits or misses in the DRAM cache with a fast SRAM access time. If the tag matches in the SRAM tag, the DRAM cache is accessed for the data block. Otherwise, the slow backing memory will be accessed. The use of SRAM tags allows low hit latencies and minimal miss penalties, as the DRAM cache is not accessed on tag misses.

However, the problem with the SRAM tag is the amount of tag space required to represent the blocks in a DRAM cache. For DRAM caches with block sizes of conventional cache blocks (i.e., 64B), the tag size would be about 96MB per GB of DRAM cache (Loh and Hill 2011). Such a large size is prohibitive for on-chip SRAMs. For DRAM caches with a larger block size (page) of 4KB, the tag overhead is smaller, but even for the page-based DRAM caches sized to be 8GB, the tag

Table 1. Summary of Three Representative DRAM Cache Organizations

Scheme	Granularity	Placement	Worst case access	Optimization
LH Cache	block (64B)	set-associative (29-way)	3 tag + 1 data	Skip high-latency tag miss lookups with MissMap (SRAM)
Alloy Cache	block (64B)	direct mapped	1 access	Access memory in parallel with cache lookup for predicted miss
Unison Cache	page (960B)	set-associative (4-way)	1 tag + 1 data	Reduce B/W with page footprint and latency with way prediction

sizes range up to 48MB (Jevdjic et al. 2014, 2013), too large to be in on-chip SRAM. Therefore, a more feasible approach is to store the tags in the DRAM cache together with the actual data.

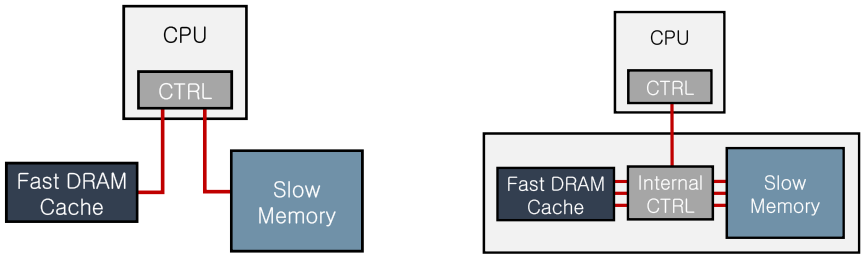
Due to DRAM architecture, the placement of the tags results in different DRAM cache hit latencies. To minimize the access latency between the tag and data, the prior work places the data and the tag on the same DRAM row (Jevdjic et al. 2014; Loh and Hill 2011; Qureshi and Loh 2012). Such placement allows the tag and data to be read without having to switch DRAM rows, which would introduce additional latencies to activate a new row. Furthermore, the size of the DRAM cache block also plays a big role in the performance of the DRAM cache. Block-sized DRAM cache proposals (Loh and Hill 2011; Qureshi and Loh 2012) aim to store the fine-grained blocks that are currently being used accurately, whereas the page sized DRAM cache proposals (Jevdjic et al. 2014) aim to provide better cache hit rates by exploiting spatial locality.

In this work, we study three representative prior approaches for different DRAM cache configurations. We chose the three studies as they were the pioneering work in DRAM cache organizations, and are competitive as of this writing. Table 1 summarizes the characteristics of three DRAM cache designs. The last column of the table shows the additional optimization details employed by the designs. These optimizations are included in the evaluation of each design.

Firstly, Loh and Hill Cache (henceforth labeled LH Cache) implements a block-based DRAM cache with tags clustered in the front of a row, and the data located in the same row after the tags (Loh and Hill 2011). LH Cache supports a 29-way set-associative DRAM cache, where different sets are mapped to different DRAM rows. As the 29 tags of the same set are spread across three 64B blocks, it will require up to three DRAM accesses to check the tag, followed by a DRAM access for data if it is a hit. In addition, LH cache has `MissMap` to reduce the latencies for DRAM cache misses. MissMap uses a bit vector-based tracking for filtering DRAM cache tag lookups. If a miss is reported by MissMap, the processor can quickly determine to go directly to the backing memory without any DRAM cache tag lookups. Otherwise, the access goes to the DRAM cache for a tag lookup. Since the MissMap structure requires a few megabytes of storage, LH cache proposes to embed the MissMap structure in the L3 cache.

Secondly, Alloy cache is also a block-based DRAM cache where the tag and data are grouped and placed together (Qureshi and Loh 2012). The tag and the data, which are adjacently placed, are read together via a single DRAM burst operation. However, this requires an additional DRAM burst length. To provide the single burst operation, Alloy cache is designed as a direct mapped cache, which may result in increased conflict misses in the DRAM cache. In addition, Alloy cache employs a hardware-based miss predictor called MAP-I. When MAP-I predicts a DRAM cache miss, the DRAM cache and backing memory are accessed in parallel.

Finally, Unison cache is a page-based cache, which stores the tags of the same set together into a single cache block of 64B (Jevdjic et al. 2014). This allows the cache to be set-associative, and places the tag and data on the same row to minimize the subsequent DRAM data fetching time. An 8KB DRAM row is partitioned into two sets. Each set holds four ways, and the tags of four ways are stored at the beginning of the set, followed by four 960B data blocks. The large block size (960B) of Unison can potentially increase cache hit rates, as it can prefetch data to the DRAM



(a) Memory directly attached to the CPU

(b) Memory internally coupled and attached as a whole to the CPU

Fig. 1. Two different organizations of hybrid memory with a DRAM cache.

cache, before they are demanded by the processor. However, it can increase the memory traffic to transfer unused portions of the data between the DRAM cache and backing memory. To reduce the unnecessary memory traffic, Unison cache takes advantage of the page footprint predictor proposed by the Footprint cache (Jevdjic et al. 2013), and loads only the predicted part of a page to the DRAM cache to reduce bandwidth consumption. To reduce hit latencies, Unison uses a way predictor. The tag and data lookups can be done concurrently if the way prediction is correct. In addition, through the footprint prediction mechanism, it can identify a page which uses only a single block. Unison does not allocate such a page in the DRAM cache to increase effective capacity, and the requested block is bypassed to the on-chip cache. However, in the evaluation of this article, the single block bypassing is not simulated.

The prior works that have been discussed are targeted for a certain combination of memory technologies incorporated into a single system. Looking forward, there are more possible combinations of memory technologies available, and different memory types result in different behaviors of the DRAM cache. We find that a single DRAM cache organization does not perform well in all the combinations of memory technologies that may be introduced into products in the near future. Furthermore, application behaviors also affect the effectiveness of each design significantly. In the following sections, we explore the effect of the prior work on the application memory patterns under different combinations of memory technologies.

2.2 Hybrid Memory Scenarios

3D stacked memory such as MCDRAM, HMC, and HBM (High Bandwidth Memory) can increase the available DRAM bandwidth significantly with vertical channels to memory banks. In addition, they can potentially reduce latencies. Oppositely to such high bandwidth and low latency memory components, new non-volatile memory technologies provide increased capacity for memory. Such a backing memory component with high capacity can accommodate ever-growing memory footprints of in-memory big data applications. A hybrid memory system with such two distinct memory technologies provides fast high bandwidth accesses for frequently accessed data, while the large capacity of the backing memory can avoid costly page faults which otherwise occur by the limited memory capacity.

In this study, the fast first level memory is organized as a DRAM cache. We consider two types of organization for the DRAM cache and second level memory. Figure 1 describes the two organizations. In the first organization (Figure 1(a)), both fast and slow memory are connected to the CPU via separate interconnects. The DRAM cache controller resides in the CPU. In the second organization (Figure 1(b)), a pair of fast and slow memory forms a single memory module,

Table 2. Combinations of Memory Technologies Organized to form DRAM Cache and Backing Memory

		fastHBM-DDR	HBM-DDR	fastHBM-PCM	HBM-PCM	DDR-PCM(T)	DDR-NAND(T)
1st Mem	Lat.	Half of HBM	DDR4-1600	Half of HBM	DDR4-1600	DDR4-1600	DDR4-1600
	BW	409.6GB/s	204.8GB/s	409.6GB/s	204.8GB/s	51.2GB/s	51.2GB/s
2nd Mem	Lat.	48.75ns (R/W)	48.75ns (R/W)	150/300ns (R/W)	150/300ns (R/W)	150/300ns (R/W)	3/100us (R/W)
	BW	51.2GB/s	51.2GB/s	51.2GB/s	51.2GB/s		

which is connected to the CPU via a common interconnect. The fast memory is managed transparently from the CPU with the internal cache controller. In such *tiered* memory organizations, fast DRAM memory functions as an internal buffer. We denote these configurations as *tiered*, (T). The tiered memory does not require the changes in the CPU side controller, and thus it is employed by the commercial and academic approaches to support a hybrid memory without the required CPU changes (Amidi 2016).

Considering the diverse characteristics of DRAM and NVM technologies, we use six different hybrid memory scenarios. The list of the selected configurations is shown in Table 2. The timing parameters used for HBM, DDR (Samsung Electronics 2017), PCM (Kannan et al. 2017), and NAND (Cheong et al. 2018) are shown in Table 2. Unlike the prior NAND flash tuned for storage, the NAND flash memory we evaluate is optimized to provide high bandwidth and low latency as the backing memory.

The fastHBM-DDR scheme is where the DRAM cache is implemented by fastHBM, and the backing memory is DDR4 memory. FastHBM is similar to the 3D stacked DRAM configuration used by many prior DRAM cache studies with the stacked DRAM as DRAM cache (Chou et al. 2014, 2015; Huang and Nagarajan 2014; Jevdjic et al. 2014, 2013; Jiang et al. 2010; Loh and Hill 2011; Qureshi and Loh 2012; Sim et al. 2014). FastHBM has a half of HBM latency and its bandwidth is doubled. FastHBM-PCM scheme uses PCM instead of DDR4 memory for the backing memory.

HBM-DDR and HBM-PCM are hybrid memory configurations using the HBM technology. The 3D stacked HBM provides a large DRAM bandwidth with eight channels, while its latency is close to those with conventional DDR memory, as the current HBM uses the same DRAM organization for each layer of the 3D stacked DRAM modules. In the two configurations, the backing memory is either the conventional DDR or the non-volatile PCM. In HBM-PCM, PCM read and write latencies are slower compared to DDR. The organization of the three memory configurations is described in Figure 1(a).

The last two configurations, DDR-PCM and DDR-NAND, are the hybrid tiered memory design with DDR4 interface. The DDR-PCM configuration uses the conventional DDR4 memory as the fast memory and PCM as the slow memory. As represented by Figure 1(b), the interface uses DDR4, and the DDR DRAM is used as the internal buffer for the high capacity PCM or NAND. The bandwidth of the DDR4 memory is limited to 51.2GB/s.

3 MOTIVATION

This section investigates the performance of three DRAM cache designs for various workloads on the six hybrid memory scenarios. We use the ZSim simulator (Sanchez and Kozyrakis 2013) to evaluate the DRAM cache design. The details of the simulation and application configurations are shown in Section 5.

3.1 The Effect of Application Memory Patterns

As each DRAM cache design has a different flow for tag and data accesses in addition to its block size, the latency and bandwidth of fast memory as well as slow memory can affect the DRAM cache

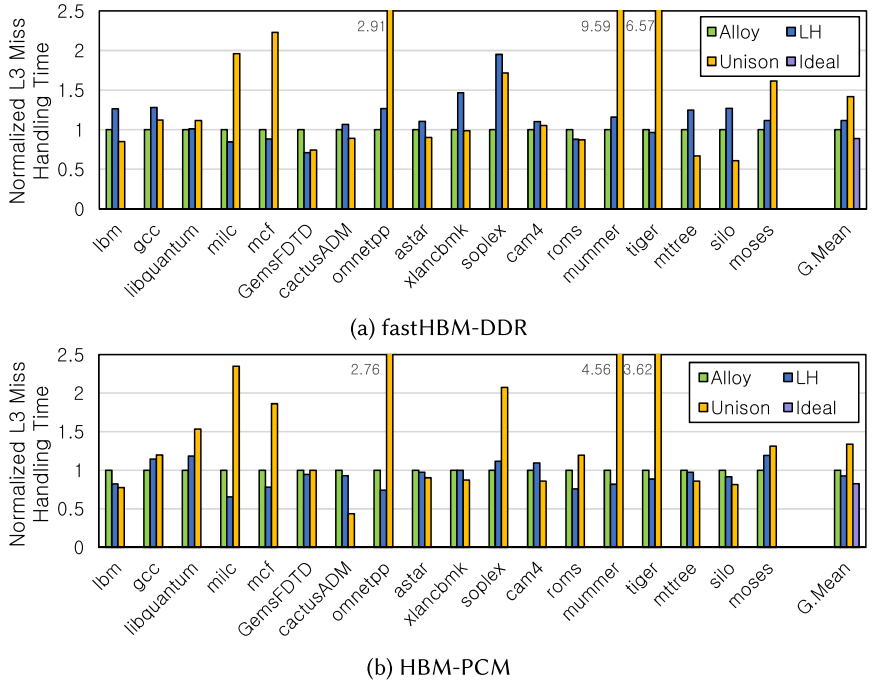


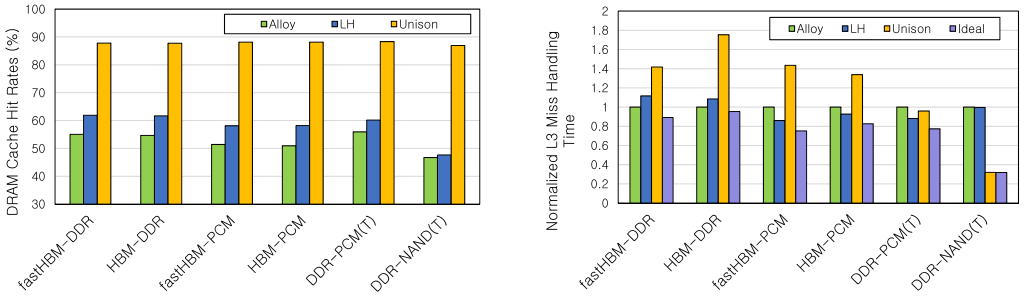
Fig. 2. Normalized L3 miss handling times of three approaches and ideal (per-application best).

differently. Alloy cache can reduce the fast memory hit latency and DRAM bandwidth consumption, but have higher miss rates due to the direct-mapped organization. In Unison cache, increasing block sizes can improve spatial locality, but have a higher cost for migrating data from the slow memory to the fast memory. Such migration consumes the limited bandwidth of the slow memory as well as that of the fast memory. Note that in the page-based Unison cache, when a miss occurs at the DRAM cache, the request is served for the CPU as soon as the requested 64B block arrives from the slow memory. After that, the rest of the page is transferred in the background from the slow memory to the DRAM cache.

In this section, we first show that application access patterns determine the best DRAM cache organization. Figure 2(a) and (b) present the normalized L3 miss handling time for all applications with the fastHBM-DDR and HBM-PCM configurations. Note that the L3 miss handling time is the latency from the L3 *on-chip cache* miss to the final return of the data either from the DRAM cache or backing memory. The latency encompasses the fast memory access time, and additional slow memory access time, if the access is a fast memory miss. Note that it is measured from the simulation by counting the number of cycles taken for every L3 miss handling.

In the figure, in addition to the three prior approaches (Alloy, LH, and Unison), the geomean bars show the per-application best configuration (`ideal`). For the geomean of the ideal performance, the best DRAM cache design is used for each application, and thus the shortest L3 miss handling time for each application from the three designs is averaged.

For the fastHBM-DDR configuration (a), the best fixed DRAM cache across all applications is Alloy cache, with the lowest average L3 miss handling time. LH and Unison caches have 11.5% and 41.7% higher latencies than Alloy cache for the hybrid memory. However, the ideal design by selecting the best one for each application can reduce the latency significantly from Alloy cache. The average miss handling time can be reduced by 10.9% with the ideal one, compared to Alloy



(a) DRAM cache hit-rates (b) Normalized L3 miss handling times

Fig. 3. DRAM cache hit rates and normalized L3 miss handling times of six hybrid memory scenarios.

cache. The figure shows that each application has a different preference for the best DRAM cache design.

The HBM-DDR configuration (b) also exhibits similar diversities for the effectiveness of the DRAM caches. In the configuration, the best fixed one across all applications is LH cache. However, with the ideal selection, the miss handling time can be reduced by 11.1% from LH cache. As shown in the results, even though the averaged latency may prefer one design, each application prefers a different DRAM cache design. To accommodate such diverse preferences for DRAM cache organizations, it is necessary to reconfigure DRAM caches dynamically for different access patterns.

3.2 The Effect of Diverse Hybrid Memory Organizations

In addition to the application patterns, the effectiveness of the DRAM cache designs can be affected by the characteristics of memory components. Figure 3 presents the DRAM cache hit rates and the geomean of the average L3 miss handling times for all applications with six hybrid scenarios. For the hit rates of the DRAM cache shown in Figure 3(a), LH cache provides higher hit rates than Alloy with its improved associativity. LH cache improves the hit rates of Alloy by 7.5–14.2% for the five scenarios except for DDR-NAND(T). In addition, Unison cache further improves the hit rates significantly with a larger block size by its prefetching effect of fetching about 1KB for each fast memory miss.

However, the average L3 miss time results in Figure 3(b) show that the hit rates alone do not reflect the actual performance of DRAM caches. Alloy with its single access for tag and data, outperforms the others for the first two configurations (fastHBM-DDR, HBM-DDR). Alloy improves the L3 miss handling time by 8.4–11.5% compared to LH cache, and 41.7–75.3% compared to Unison cache for the two scenarios. Although Unison has high hit rates for all the scenarios, its actual L3 miss handling time is the slowest for the first four scenarios, due to the wasted memory bandwidth consumption for migration.

In fastHBM-PCM, HBM-PCM, and DDR-PCM(T), LH provides the shortest L3 miss handling time with its improved hit rates. It reduces the L3 miss handling time of Alloy and Unison caches by 7.8–16.4% and 8.8–67.1%. Unison can reduce the average L3 miss handling time significantly for DDR-NAND(T). With the relatively slow flash memory as the backing memory, the high hit rates of Unison have strong positive effects on the final miss handling time, combined with the effect of the coarse-grained flash access efficiency.

As shown in this analysis, the performance of DRAM cache is not only dependent on the hit rates, but also on the hit latency and miss penalty, which are determined by the DRAM cache organization and fast/slow memory characteristics. The latency and bandwidth of both levels of

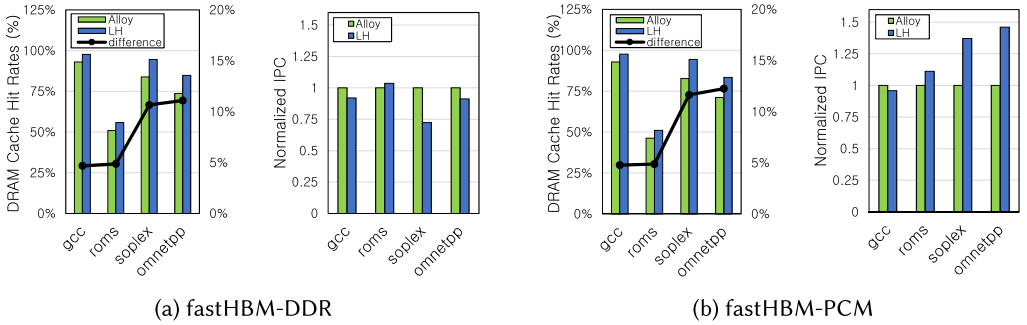


Fig. 4. The effect of difference in hit rates between Alloy and LH for two backing memory configurations.

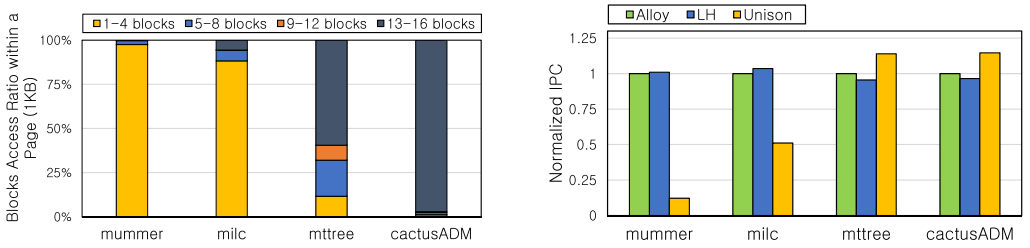
memory can significantly affect the best cache design. The last bar in Figure 3(b) is the ideal miss handling time when the per-application best design selection is used. The results confirm that across different hybrid scenarios, using the best DRAM cache design is necessary as the best design differs by both applications and hybrid configurations.

3.3 Case Analysis

In this section, we show how the effect of application patterns is combined with the different hybrid memory characteristics for the final performance of DRAM caches for a given hybrid configuration. Figure 4 presents the hit rates and IPCs (performance) for two hybrid configurations, fastHBM-DDR and fastHBM-PCM. The memory for the DRAM cache is fixed to fastHBM, while the backing memory is different between (a) and (b). For each hybrid configuration, the left graph shows the hit rates of four selected applications with Alloy and LH. The curve in the graph is the difference of hit rates between Alloy and LH. The right graph shows the corresponding performance in terms of instructions-per-cycle (IPC).

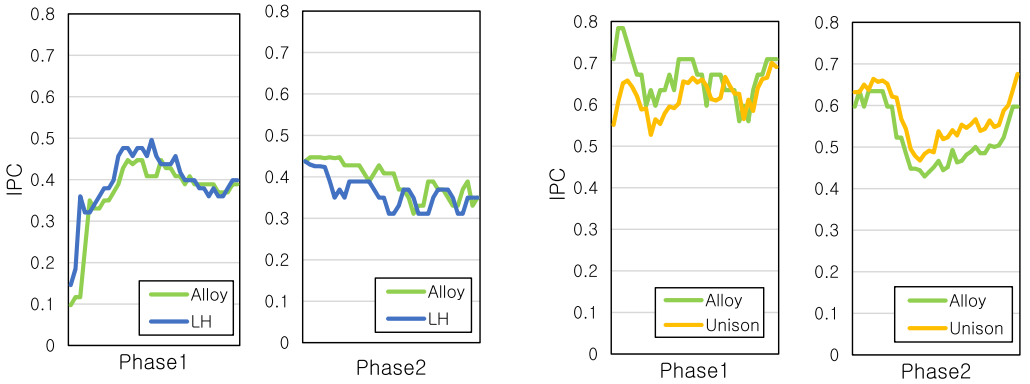
In the figures, the left hit rate graphs of both (a) and (b) are similar since they use the same fastHBM capacity as the DRAM cache. For the four applications, LH has higher hit rates and the hit rate differences for soplex and omnetpp are larger than those for gcc and roms. However, the actual performances differ between (a) and (b). When the backing memory is relatively fast DDR in (a), the high hit rate improvements by LH are not translated to the final performance. Instead, the short hit time of Alloy has stronger positive effects, leading to the better performance with Alloy. However, with PCM as the backing memory in (b), LH outperforms Alloy significantly with its high hit rates. The results indicate that even similar hit rates with a DRAM cache can have different impacts on the final performance, depending on the latency and available bandwidth of the hybrid memory components.

The second analysis shows how the spatial locality affects the effectiveness of the page-based Unison cache. Figure 5(a) presents the decomposition of how much of a page is actually accessed in the DRAM cache once a chunk of 1KB data is brought in the Unison cache. In the figure, mummer and milc have low spatial localities, and in the majority of pages in the DRAM cache, only 1–4 64 blocks are actually accessed. However, for mttree and cactusADM, large portions of a page are accessed once they are brought into the DRAM cache with much higher spatial localities than mummer and milc. Figure 5(b) shows the performance (IPCs) of the DRAM cache designs on fastHBM-DDR. As shown in the figure, when the spatial localities are high in mttree and cactusADM, the page-based Unison cache outperforms the other two designs, while Unison suffers severely for low-locality mummer and milc.



(a) Fraction of block (64B) accesses in a page (1KB) (b) Normalized performance (IPCs)

Fig. 5. The relationship between page-based cache and spatial locality: fastHBM-DDR.



(a) milc : fastHBM-DDR (b) lbm : HBM-DDR

Fig. 6. The effect of memory access pattern changes.

The final analysis is how changes in memory access patterns affect different cache organizations. Figure 6(a) shows the two phases of milc on fastHBM-DDR. This graph presents the IPC changes during the execution for Alloy and LH. The performance of LH in phase1 is slightly better, but Alloy outperforms in the latter phase. Figure 6(b) represents the performance changes of lbm running on HBM-DDR. Similarly, the type of cache that shows high performance varies across phase changes. Alloy shows higher performance in the first phase, but Unison has higher performance in the next phase. This experiment shows that even within an application, the preferred cache type is different depending on the memory access pattern.

This case analysis re-confirms that multiple factors affect which design is the best for given application behaviors and hybrid memory characteristics. These factors include the working set size and temporal/spatial locality of an application, and the latency and available bandwidths of the fast and slow memory components.

4 RECONFIGURABLE DRAM CACHE ARCHITECTURE

4.1 Overall Design

In DRAM caches, without any physical separation of tags and data in their storage, rows of DRAM can be used in any way the memory controller chooses to store tags and data. Using this flexibility, we propose a morphable DRAM cache (MDC). In this section, three morphable modes are presented: direct-mapped mode (`MDC_DM`), eight-way set-associative mode (`MDC_8way`), and four-way set-associative page mode (`MDC_page`). For each row of the fast memory (HBM or DDR4),

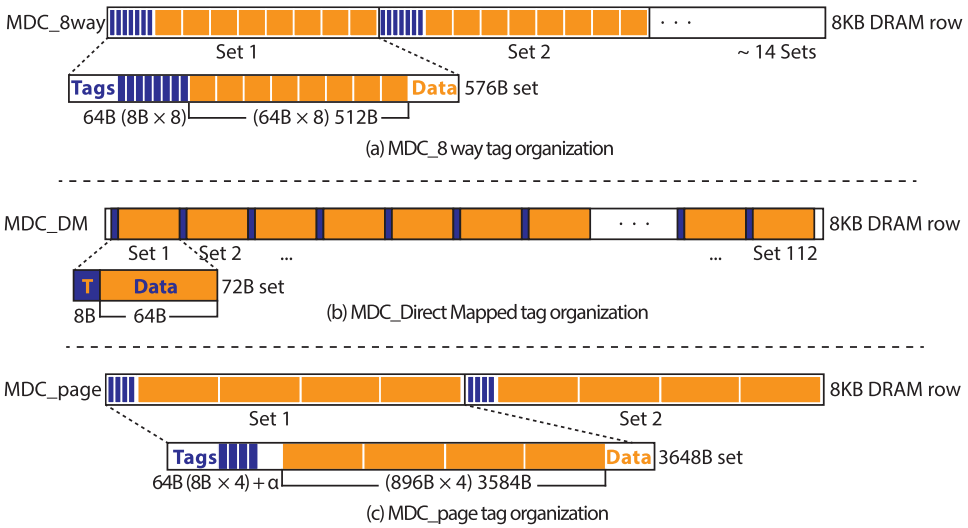


Fig. 7. Tag and data layouts in a row for three DRAM cache modes.

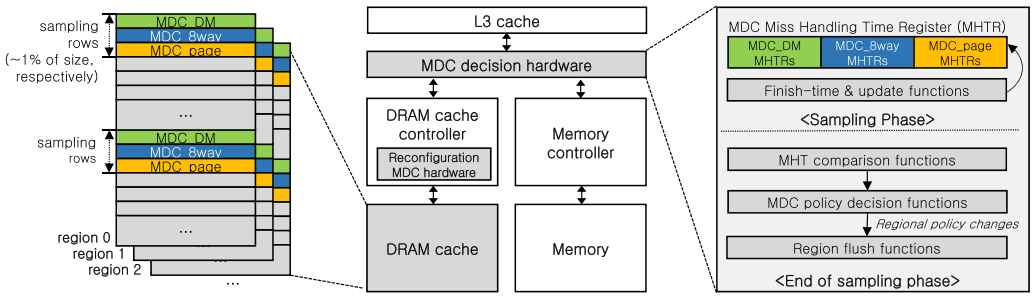


Fig. 8. Morphable DRAM cache architecture.

tags and data are located to be reconfigurable among the three different mapping modes. Figure 7 presents the three tag and data layouts stored in a row.

MDC is periodically reconfigured based on the performance assessment under the current workload and hybrid configuration. To identify the best DRAM cache configuration, small subsets of DRAM cache sets are fixed to each of the three organizations. Based on the sampled measurement of each option, the DRAM cache controller reconfigures the DRAM cache. We discuss the sampling method in detail in Section 4.3. In this work, we organize the DRAM cache (whether it be HBM or DDR4) into 32 regions. Each region is a 32MB DRAM cache. Each region acts independently and can be morphed into a different scheme based on the performance results of the region.

Figure 8 presents the overall architecture of MDC. The DRAM cache controller is extended to support three modes, by adding more states in the internal state for each access transaction. The required changes to the DRAM cache controller is relatively minor. An extra module is the MDC decision hardware, which consists of two components. First, it has a set of three MDC miss handling time registers for each region of DRAM cache. The registers aggregate the execution latencies for sampled accesses to each design. Second, it has a logic to reconfigure the DRAM cache at 32MB region granularity. The logic is responsible for writing back the modified data to

Table 3. Modifications Made During the Adoption of the Prior Work to be Reconfigurable

Scheme	Based on	Modifications	Reason
MDC_8way	LH	29-way reduced to 8-way	3 tag cacheline lookups found to be expensive
MDC_DM	Alloy	28 block to 112 block increase	Initial design for 2KB rows, modified for 8KB rows
MDC_page	Unison	960B page size reduced to 896B	Compatibility with 112 block per row configurations

the backing memory, if the data cannot be in the DRAM cache due to the mapping change during a reconfiguration process for a region.

4.2 Reconfigurable Tag and Data

In the MDC, the original three DRAM cache designs are modestly modified to enable reconfigurability. The modifications made to make the prior schemes configurable are shown in Table 3. The schemes MDC_8way, MDC_DM, MDC_page are the schemes used in this work, which stem from the prior work as shown in the table.

Firstly, the original LH cache provides a 29-way associative DRAM cache. As the 29 tags of the same set are spread across three 64B blocks, it requires three DRAM accesses to check the tag in the worst case. In this work, we use an eight-way associative LH cache that always requires a single tag access, as our initial testing found that the 29-way LH cache tag lookup costs are too expensive. This mode is denoted as MDC_8way. Secondly, the original direct-mapped Alloy cache is designed to use 28 cache blocks per row in the 2KB buffer configuration. In this work, we use an 8KB buffer, and thus we increase the number of cache blocks to 112 blocks per row. This mode is denoted as MDC_DM. Thirdly, the original Unison cache is a page-based cache. A 8KB DRAM row is partitioned into two sets. Each set holds four ways, and the tags of the four way are stored at the beginning of the set, followed by four 960B data blocks. This work reduces the page size to 896B, which is 14 cache blocks, to make the amount of data stored in a row equivalent to the re-configurable Alloy and LH modes. This organization is denoted as MDC_page. Unifying the amount of cache blocks stored in each row allows consistent address indexing to all DRAM cache rows, regardless of the mapping scheme of each DRAM cache row.

In MDC, tags and data can be stored in the DRAM cache in three different ways. Figure 7 presents the tag and data mapping schemes in a row. Firstly, (a) represents MDC_8way where the row is divided into 14 sets, and within each set the eight tags are clustered together at the front of the set, followed by the eight corresponding data blocks. Such organization results in a eight-way set-associative organization. Secondly, (b) of Figure 7 represents MDC_DM where the tag and data are stored adjacently and form a direct-mapped cache. Finally, (c) depicts MDC_page mode where the row is divided into two sets, and each set begins with four tags followed by four 896B blocks of data. The MDC_page mode offers a four-way associative page sized DRAM cache.

To provide the proper lookup of cachelines in the presence of reconfigurable DRAM cache, the incoming address needs to be properly parsed to locate the corresponding cacheline across reconfigurations of the DRAM caches. Also, to minimize data movement between reconfigurations, the cache block placement must change as little as possible. In the following paragraphs, we discuss how the address bits are used to locate a cache block, and how the index bits change for a given cacheline across different mapping schemes.

Firstly, the address is split into two regions: the row addressing region, and the within-row addressing region. The row addressing region (*row index*) is hashed by a hash function, and the DRAM row is located from the outcome of the hash function. The within-row addressing region (*within_row index*), is the region of bits that are used to locate the cache block within a row. Unlike conventional caches, the number of sets within a row is not a power of two, and so the row index

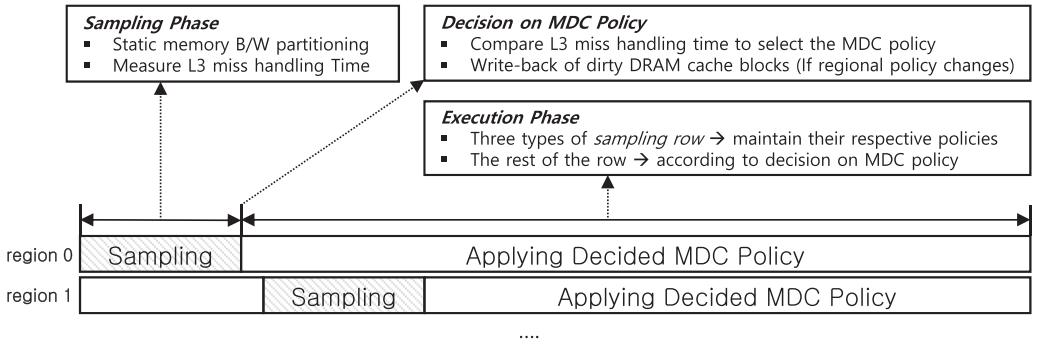


Fig. 9. MDC sampling, decision, and execution process.

part and the within-row index part of the address cannot be partitioned in the bit level. In this work, a design invariant has been placed, where each row holds 112 cache blocks (64B) of cache data. Even for the page sized scheme where the page size is 896B, the total data size is 896B × 4 ways × 2 sets, which corresponds to 112 cache blocks. This invariant allows the within-row index portion to be a modulo of 112 excluding the lower six bits which is the cache block offset. Naturally the row index is the address divided by 112. The index calculation is shown in Equation (1).

$$\begin{aligned} within\_row\ index &= (address >> 6)\%112, \\ row\ index &= (address >> 6)/112. \end{aligned} \tag{1}$$

The three mapping schemes make use of the within row address which is in the range of [0, 112) in the following manner. Firstly, `MDC_DM` uses the entire value to index the 112 sets in a row of the direct mapped cache. Secondly, `MDC_8way` has 14 sets with 8 ways per set in a row.

4.3 Periodic Reconfiguration

For periodic reconfiguration, MDC manages the DRAM cache as 32 regions. DRAM rows are equally partitioned to each region, where the upper five bits of a valid physical address decide which region the address can be cached into. Each region is independently assessed to get the preference of DRAM cache organization based on the memory access characteristic. The sampling and decision making to enforce a certain configuration for each region is discussed in the Section 4.4. Figure 9 presents the region-oriented periodic reconfiguration process.

The reconfiguration from one mapping mode to another results in memory bandwidth being consumed by the DRAM cache. The region reconfigured is invalidated, and any dirty blocks or pages, depending on the mapping scheme before the reconfiguration, are written back to the backing memory. To limit the drop in system performance due to the writebacks, the DRAM cache is reconfigured by a region at a time. All the DRAM rows within a region are accessed, and the controller issues writebacks for any dirty blocks in each row.

The reconfiguration cost in terms of writeback data size and reconfiguration frequency needs to be low for our proposal to be effective. From our evaluation, we confirmed that the worst case reconfiguration cost was observed when running the mttree benchmark, writing back at most 13.4MB dirty cachelines from a DRAM cache region of 32MB. On average, each region performed writebacks of modest 423KB data. Furthermore, some benchmarks resulted in zero writebacks when reconfiguring.

The writeback cost may seem high, however, after the initial reconfiguration to the optimal mapping scheme, most benchmarks did not perform any additional reconfiguration. There were some benchmarks that performed another reconfiguration, however this was due to the application

phase change, and resulted in better MDC performance by the reconfiguration. Therefore, there is some reconfiguration cost. However, once the optimal mapping for an application is found, additional reconfiguration costs are minor. Also, on program phase changes, MDC detects the phase change and transitions to a better mapping scheme for the new phase. In Section 6.1, we discuss additional optimizations to reduce writebacks across reconfigurations.

4.4 Sampling Mechanism

As shown in Section 3.2, each workload has a different optimal configuration for a hybrid memory organization. The optimal configuration may also change depending on execution phase during runtime. Thus, the reconfiguration decision needs to be made by inspecting the MDC performance of three MDC mapping schemes at runtime. To apply a preferred configuration in a fine-grained manner, MDC inspects the performance of configuration by region.

We assign a small portion of DRAM rows of a region to three sampling sets: `MDC_DM`, `MDC_8way`, and `MDC_page` schemes. In this work, we allocated 1/128 rows for each sampling scheme. In total, 3/128 rows of each region are used to sample the performance of each scheme at runtime. These sampling rows are mapped in their individual schemes, and also serve as part of the DRAM cache. As all data blocks map to a single unique row, regardless of the mapping scheme of the row, having sampling rows with different mapping schemes within a region is allowed in MDC.

Sampling a small fixed subset of the total cache sets was proposed and extensively used by the prior set-sampling techniques (Qureshi et al. 2008). The prior work showed that even if a small fixed subset of the total sets are sampled, the access behaviors of the sampled sets accurately represent the behaviors of the entire cache. The rationale behind the cache set-sampling is that the mapping between the virtual address to the cache set is in general random, and thus sampled sets can represent the behaviors of the entire cache due to the uniform random set behaviors. In the prior study, only 32 sampled sets were shown to be good enough for representing thousands of sets, with a statistical analysis. We apply the same principle of the cache set sampling to the MDC selection mechanism.

During the sampling period, MDC measures the average request handling latency values for each sampling section, as a hit rate alone is not a proper metric for selecting the best mode. One of the key mechanisms for accurate sampling of the request handling latency is the bandwidth isolation for accesses to sampled rows, since the bandwidth consumption by the normal rows can affect the latencies for the sampled rows. For accurate assessment of sampled rows, MDC isolates the bandwidth for sampled rows from the other rows. We implement a memory bandwidth partitioning mechanism similar to the fair queuing memory system (Nesbit et al. 2006). However, we only use the static partitioning mechanism to ensure that the sampling rows are not negatively affected by the other rows. The fair queuing scheduling algorithm updates the private virtual clock per thread through the calculations of virtual start and finish time. When a thread shares the memory bandwidth in a system of `Q` threads, it will have `1/Q` bandwidth to be fair. Similar to this approach, in this work, we assign virtual times to rows instead of threads. 1/128 of the memory bandwidth is allocated for the sampled rows of each mode. The rest of normal rows can utilize 125/128 of the DRAM cache and slow memory bandwidth.

At the end of the sampling phase, which is 10M cycles in this work, the controller chooses an appropriate policy for each region. By comparing the average request handling latencies of the three schemes, it selects the best policy for each region.

4.5 Required HW Changes

To support the reconfigurable DRAM cache, three components must be modified or added. Firstly, the states of the DRAM cache controller must be extended to represent necessary state transitions

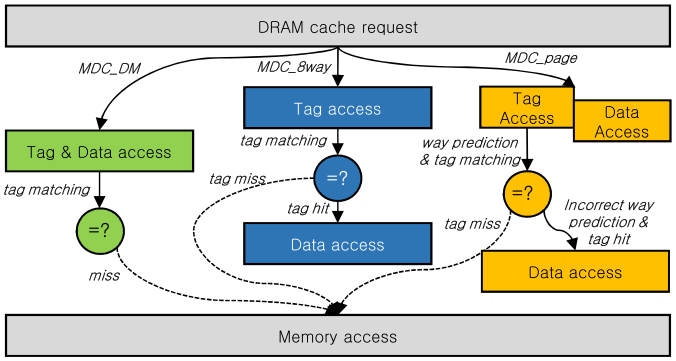


Fig. 10. MDC execution flows for three modes.

for three modes. Figure 10 shows the simplified state transitions for three modes. The original DRAM cache controller of each mode needs to represent only one flow, but the MDC controller can provide three flows. Secondly, the miss handling times must be tracked for sampled accesses. For each region of the DRAM cache, a set of three registers are added to aggregate the handling time for a sampling period. However, the extra storage for the registers is relatively minor, with $32 \times 3 \times 4B = 384B$ extra SRAM storage for 32 regions. Finally, another extra logic component is for flushing the modified data during a reconfiguration step of a region. The flushing logic sweeps through the DRAM cache sets of the region, and flushes the modified data to the backing memory.

Although optimizations for each DRAM cache organization are not essential for reconfiguration, MDC includes a subset of the optimizations used by the prior three approaches. MDC uses the MAP-I optimization of Alloy, and the storage overhead for MAP-I is 3.5KB (256 entries $\times$ 3-bit $\times$ 36 cores). For the footprint prediction and way prediction from Unison, MDC uses 16K footprint history table entries (144KB storage overhead) and 1KB storage for way prediction. Note that those extra structures for optimizations are shared by all 32 regions, and they are not replicated for each region.

5 EVALUATION

5.1 Methodology

We simulated our reconfigurable DRAM cache on an execution-driven simulator based on ZSim (Sanchez and Kozyrakis 2013). It models a 36-core processor and its cache architectures with the heterogeneous memory system, as shown in Table 4. Each core has 32KB L1 caches, and a 256KB 8-way L2 private cache. All cores share a 36MB 16-way set-associative L3 cache and 1GB DRAM cache. The processor model is similar to the one used in the prior study (Tsai et al. 2017). We organized six types of hybrid memory architectures (fastHBM-DDR, HBM-DDR, fastHBM-PCM, HBM-PCM, HBM-PCM, DDR-PCM (T), and DDR-NAND (T)) to investigate the effects of different memory characteristics such as bandwidth, latency, and capacity (as discussed in Section 2.2). To model DRAM cache and memory, we use two separately configured memory models from DRAMSim2 (Rosenfeld et al. 2011), and they are integrated into ZSim to model fastHBM, HBM, DRAM, PCM, and NAND.

Our reconfigurable architecture is modeled with the morphable DRAM cache mechanism with a dynamic selection. We compare the proposed MDC performance with diverse schemes based on the prior approaches (Alloy, LH, and Unison Cache). The three prior techniques include the optimizations discussed in Table 1. Additionally, Unison allows bypassing the DRAM cache and directly forwards a block to the L3 cache when only a single block is used in a page, but the

Table 4. Configuration of the Simulated System

Component	Configuration
CPU	36 cores, x86-64 ISA, 3.2GHz
L1 cache	32KB private, 8-way set-associative, 3-cycle latency
L2 cache	256KB private, 8-way set-associative, 7-cycle latency
L3 cache	36MB shared, 16-way set-associative, 27-cycle latency
fastHBM	half latency of DDR4-1600, 8 channels, 128 bits per channel
HBM	DDR4-1600, 8 channels, 128 bits per channel
DDR	DDR4-1600, 4 channels, 64 bits per channel (tCAS=11, tRCD=11, tRP=11, tRAS =28)
PCM	150ns read latency, 300ns write latency, 4 channels (in fastHBM-PCM/HBM-PCM)
NAND	3us read latency, 100us write latency

Table 5. Benchmarks Used to Evaluate MDC

Suite	Benchmarks
SPECCPU 2006	lbm, gcc, libquantum, milc, mcf, GemsFDTD, cactusADM, omnetpp, astar, xalancbmk, soplex
SPECCPU 2017	cam4, roms
Biobench	mummer, tiger
TailBench	mttree (masstree), silo, moses

single-block bypassing optimization is not included in our simulation for Unison. MDC employs the same MAP-I optimization of Alloy. `MDC_8way` does not use MissMap because of its incompatibility with other modes. `MDC_page` adopts a page footprint predictor and way predictor of Unison.

We run memory-intensive workloads in SPECCPU 2006 (Henning 2006), SPECCPU2017 (spec.org 2017), Biobench (Albayraktaroglu et al. 2005), and TailBench (Kasture and Sánchez 2016) with large memory footprints. Our workloads are shown in Table 5. We executed 36 copies of each workload for 36B cycles with fast-forwarding 20B instructions. In addition to the runs with the same application in all cores, we evaluate eight mixed runs as shown in Table 6.

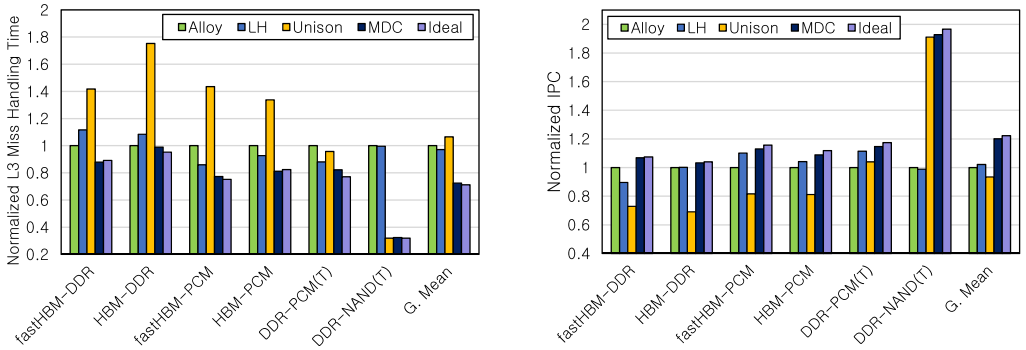
5.2 Evaluation Results

In the evaluation results, we first present the average performance for six hybrid scenarios, and show the performance of each individual application. In addition to the single application runs, the mixed runs are presented.

L3 Miss Handling Time: Figure 11(a) presents the average L3 miss handling time for six hybrid memory scenarios. As shown in the figure, the proposed morphable cache can provide the shortest miss handling time compared to the previous DRAM cache organizations. For fastHBM-DDR and HBM-DDR, MDC can reduce the latency by 7.4% from Alloy cache, which is the best organization for the three scenarios. For fastHBM-PCM, HBM-PCM, and DDR-PCM (t), MDC can reduce the latency by 10.7% from LH cache. For DDR-NAND (t), MDC can provide latencies similar to those with Unison cache. For the first two scenarios, MDC can provide shorter latencies even compared to the best configuration (Alloy cache), since its dynamic reconfiguration can select the best configuration for different applications. With the geomean across the six scenarios, MDC can reduce the miss handling time by 26.3%, 25.1%, and 30.8%, compared to Alloy, LH, and Unison caches, respectively.

Table 6. Mixed Workloads

Name	Workloads in mix
mix1	{soplex-GemsFDTD} ×18
mix2	{lbm-tiger} ×18
mix3	{gcc-astar} ×18
mix4	{libquantum-mummer} ×18
mix5	{mcf-omnetpp} ×18
mix6	{milc-xalancbmk} ×18
mix7	{lbm-tiger-libquantum-mummer} ×9
mix8	{soplex-GemsFDTD-gcc-astar} ×9



(a) L3 miss handling times normalized to Alloy

(b) IPCs normalized to Alloy

Fig. 11. Normalized L3 miss handling times and IPCs with six hybrid memory scenarios.

Instruction Throughput: Figure 11(b) presents the IPCs normalized to those with Alloy cache. This final IPC result includes the effect of reconfiguration overheads. Even with the overhead for flushing the DRAM cache for reconfiguration, MDC outperforms or provides similar performance to the best configuration for each scenario. Across all six scenarios, MDC can outperform Alloy, LH, and Unison caches by 20.1%, 17.6%, and 28.5%, respectively. For fastHBM-DDR and HBM-DDR, MDC slightly outperforms even the best configuration (Alloy) since MDC can choose the best cache mode for each application as will be shown in the next result. Even within an application run, MDC can adapt to phase changes. The next detailed results will discuss the per-application performance.

Detailed Results: Figure 12 presents the detailed normalized IPCs for all the applications with the six hybrid configurations. As shown in Figure 12, MDC can select the best organization for different applications. For example, Alloy cache is the best one for omnetpp and tiger with (b) HBM-DDR, and MDC provides close performance to Alloy even with the reconfiguration overheads. With (d) HBM-PCM, Unison is the best one for gemsFDTD and roms, and MDC matches the best performance. For all applications with the six scenarios, MDC can select the best configuration for each application accurately, and in addition, its reconfiguration overhead is small.

As shown in libquantum with (a) fastHBM-DDR, MDC can outperform the best configuration (Unison cache). This is due to the phase change occurring during the execution. Figure 13 presents the two phases of libquantum. The x-axis is the timeline and the y-axis is the normalized IPC. Note that Unison is not shown in the figure, as the presented two modes (Alloy and LH) outperform

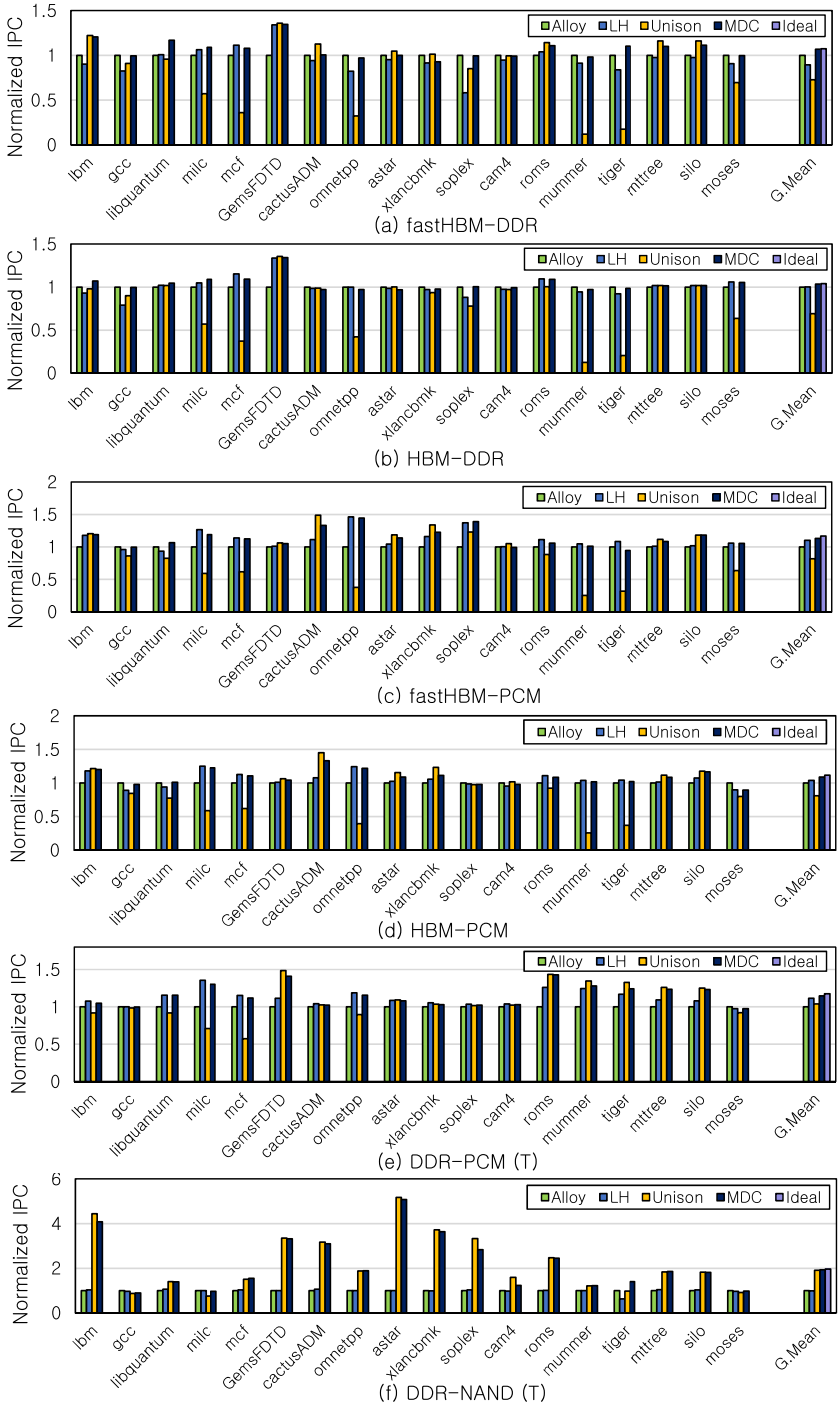
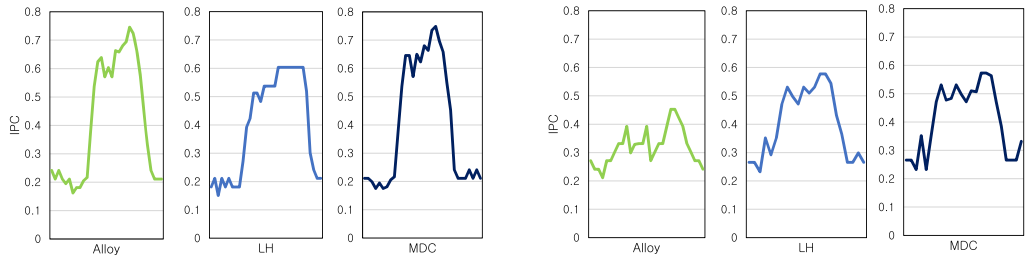
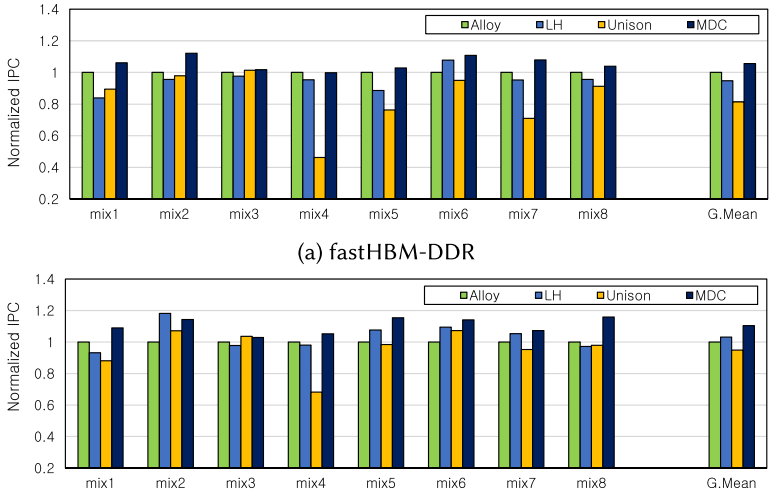


Fig. 12. Performance (IPCs) of six different hybrid memory organizations.



(a) Phase 1 (b) Phase 2

Fig. 13. IPC changes per 10m cycle for libquantum with fastHBM-DDR.



(a) fastHBM-DDR

(b) HBM-PCM

Fig. 14. Performance (IPCs) of multi-programmed workloads.

Unison for libquantum. As shown in the figure, the two phases show different IPC trends. In each phase, MDC shows an IPC curve that is similar to the better performing mode.

Mixed Applications: In addition to single application setups, we evaluate mixes of workloads with different preferences. Table 6 presents the list of mixes we evaluate in this section. The application mixes are selected to combine applications with different cache preferences, although the preference can also change by hybrid memory configurations.

Figure 14 shows the normalized IPCs for eight mix workload scenarios with fastHBM-DDR and HBM-PCM. As shown in the figure, MDC can find the configuration for the mixed workload setup. The multi-programmed application setups pose more difficult challenges than the single application runs, since different workloads prefer different DRAM cache policies. However, as shown in Figure 14, MDC still improves performance over three fixed DRAM caches by 5.5–29.7% with fastHBM-DDR and 7.1–16.3% with HBM-PCM.

Sampling Accuracy: Figure 15 shows the mean error rates of the sampled decision for six hybrid memory systems. The error rate is defined as the ratio of the incorrect decision compared to the static best configuration found by an offline analysis. The geomeans of the decision error rates are 0.32%–3.69%. However, some workloads show more than 10% errors, such as milc (15.5%) and

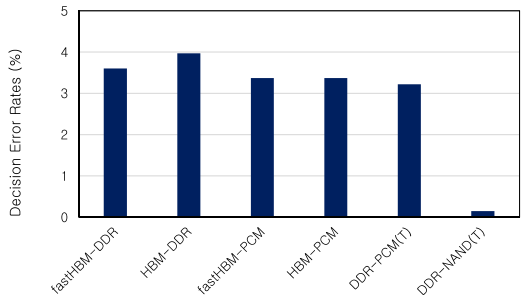


Fig. 15. Decision error rates from sampling phase.

xalancbmk (13.2%) with fastHBM-DDR, and milc (12.6%) and mcf (11.7%) with HBM-PCM. In these cases, the decisions are mispredicted because the L3 miss handling times between two policies are similar. However, there was no significant performance decrease by the misprediction, because the difference in miss handling times between two policies is small anyway.

6 OPTIMIZATION OPPORTUNITIES

In this section, we first discuss how to retain data in the DRAM cache across a reconfiguration step if the DRAM internal mechanism provides flexible data path from/to row buffers. Secondly, we discuss how the OS awareness of the morphable DRAM cache can make MDC more effective in multi-programmed workloads.

6.1 Retaining Data During Reconfiguration

In the prior sections, we assume that only the DRAM cache controller is modified without any change in DRAM architectures. However, if the internal operation of DRAM can be modestly modified, reconfiguration can be further optimized by intra-row data transfers. There are three optimizations to retain cached data within a DRAM cache row when transitioning from one mode to another.

Firstly, when transitioning from a block-based mapping to a page-based mapping, valid bitmaps can be used to support a partially filled page. If there are opportunities where a subset of blocks of a page exist in a row when transitioning, these blocks can be relocated into the page within the same row for the page-based mapping. Then, valid bitmaps for the page in the tag section (the MDC_page has 32B extra space at the tag per set; refer to Figure 7) are set for the resident blocks. On an access to the page, the bitmap is consulted to check if the block is resident or not. For partially filled pages, any future access triggers that the missing blocks are filled to complete the page in the DRAM cache.

Secondly, if the DRAM cache provides functionality to efficiently rearrange data blocks by moving data to different locations within the same row, part of the DRAM row can be reused after a reconfiguration. When such functionality is available, the transitions from MDC_8way/MDC_page to MDC_DM or from MDC_page to MDC_8way can keep blocks in the same row after a reconfiguration step. We use the MDC_page to MDC_DM transition case as an example, but the MDC_8way to MDC_DM and MDC_page to MDC_8way transitions are similar in principle.

Figure 16 represents the case where a MDC_page row is reconfigured into MDC_DM mode. MDC_page is a four-way set associative page cache with two sets in the row. The set boundaries of the MDC_page is represented by the dashed lines in the figure. There are four pages of 14 cache blocks that can be mapped into the direct mapped MDC_DM cache. As the internal blocks of a page are contiguous, they are mapped to adjacent tag and data blocks of the MDC_DM mode. This

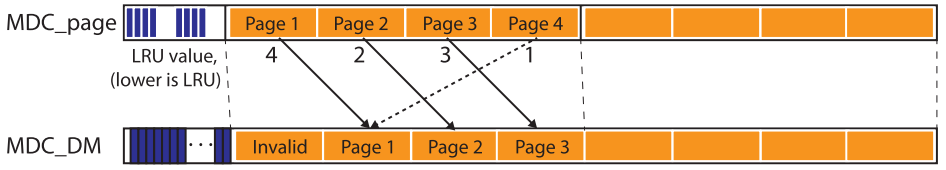


Fig. 16. Mapping from MDC_page to MDC_DM mode.

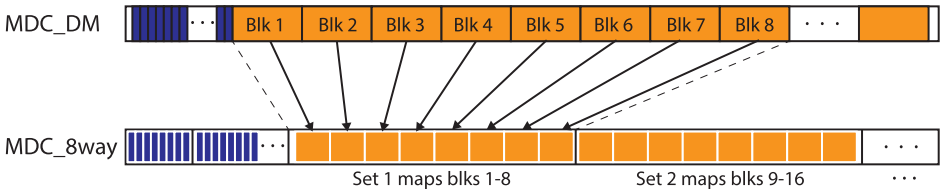


Fig. 17. Mapping from MDC_DM to MDC_8way mode.

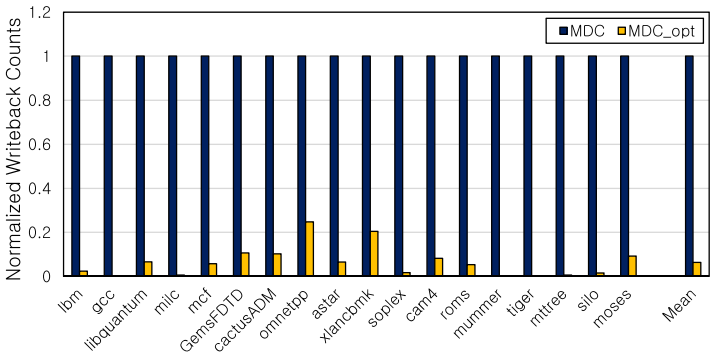


Fig. 18. Write-back reduction with optimizations enabled by DRAM modification (fastHBM-DDR).

optimization relocates the tag and data to match the layout of the new MDC_DM mode. However, the address mappings of the old and new modes are not equal; a subset of blocks can be evicted, if the address mappings have conflicts. In the figure, the blocks in Page 4 are dropped due to the mapping conflicts. When such a conflict occurs, victims are selected by their LRU priority.

Thirdly, a support for non-contiguous column access to a row can retain data within a row for the reconfiguration from MDC_DM to MDC_8way. This support requires changes in DRAM to allow a column read/write burst to access non-contiguous bits in a row. For this optimization, MDC_DM and MDC_8way are reorganized so that all tags are clustered to the front of the row. In MDC_DM mode, the non-contiguous column access can retrieve the tag and data in a single burst. As a related approach with practical implementation, the DRAM cache in Intel Knights Landing stores tags in ECC for parallel access to tags and data (Sodani et al. 2016). In the design, the stacked DRAM has additional lanes for ECC, and the tag in ECC lanes and data can be read at the same time. Figure 17 illustrates a transition from MDC_DM to MDC_8way, with the support. As shown in the figure, MDC_DM mapping is directly transformed into MDC_8way mapping without actual data movement within a row.

To show the effectiveness of the within-row reconfiguration, we measure the potential write-back reduction by the optimization. Figure 18 presents the normalized writebacks caused by flush operations during a reconfiguration. As shown in Figure 18, the optimization achieves a significant writeback reduction compared to MDC by 93.6% on average. Although MDC does not require

frequent reconfiguration, the reconfiguration can cause a temporary performance degradation due to the writebacks generated by the reconfiguration. The reduced writeback traffic of MDC_opt can almost eliminate the reconfiguration overhead. The lower reconfiguration cost can enable fine-grained reconfigurations, and thus it can better cope with rapid phase changes. Further investigating the optimization will be our future work.

6.2 OS Awareness

OS can make use of multiple regions, 32 in this work, to maximize the system performance by mapping applications of similar behavior to the same regions. If OS can recognize similar memory behaviors, or if the user specifies similarities among executed programs, OS can allocate physical memory based on the DRAM cache region, so that each region serves applications with similar memory behaviors. Each region will then find the best DRAM cache configuration for the applications mapped to the region, offering the best DRAM cache performance to the applications. Such region-aware placement can provide better performance improvement and isolation among applications or virtual machines in consolidated systems.

7 RELATED WORK

With the increasing importance of data-intensive workloads and memory heterogeneity due to the advent of new memory technologies, there have been several studies of how to use fast and high bandwidth memory more efficiently on hybrid memory systems.

Hardware-Based Hybrid Memory: In HW-based hybrid memory, a HW controller tracks data mapping across heterogeneous memory components, and quickly migrates critical data to fast memory. The prior studies proposed one of two main uses of fast memory: DRAM cache or part of memory. The main challenge for using fast memory as such DRAM cache is reducing tag area overhead and tag access latency. Alloy (Qureshi and Loh 2012), LH (Loh and Hill 2011), Footprint (Jevdjic et al. 2013), and Unison (Jevdjic et al. 2014) caches proposed optimized cache structures and mechanisms with different block sizes and associativities. Tagless cache (Lee et al. 2015) has no extra tags to eliminate its overheads for tag lookups, by storing the mapping information in TLBs. However, it requires HW/SW modifications and can migrate at page granularity.

Using fast memory as a part of main memory has the benefit of increasing available memory capacity (Chou et al. 2014; Sim et al. 2014). However, migrations based on hot/cold page detection incur more bandwidth consumption since pages need to be swapped between the different memories, while DRAM caches write-back only dirty blocks. In the HW-based part of memory approaches, simple restricted address remapping is employed to avoid increasing HW complexity. CAMEO (Chou et al. 2014) and PoM (Sim et al. 2014) proposed a simple address remapping mechanism constructing groups of fast and slow memory segments at a ratio of 1:n. This mapping restriction can offer fast address translation for physical address to actual memory address and reduce the overhead for the remapping table, although it can cause conflict misses.

Software-Based Hybrid Memory: For SW-based hybrid memory, OS uses the memory management mechanism based on virtual memory. The granularity of SW hybrid memory inherits the page granularity used by the virtual memory. Using virtual memory for hybrid memory mapping can cause overheads involving interrupt handler invocation, page table update, and TLB flush. Nonetheless, such software-based hybrid memory has become a practical approach since it can be applicable to modern systems without any HW modification. Meswani et al. proposed heterogeneous memory architecture (HMA) (Meswani et al. 2015) for hybrid memory with die-stacked and off-chip memories. They placed fast and slow memory to a single flat memory space, and investigated an adaptive migration algorithm using history-based or first touch promotion schemes.

Thermostat (Agarwal and Wenisch 2017) classifies hot/cold pages effectively considering the 2MB huge page support, aiming efficient and accurate data promotion to fast memory for such huge pages. A recent study on disaggregated memory provides a rack-scale memory extension via remote direct memory access (Koh et al. 2019).

8 CONCLUSION

An efficient hybrid memory architecture should be flexible in responding to diverse hybrid memory organizations and data access patterns of workloads. However, for DRAM cache approaches for such hybrid memory systems, this article showed that no single fixed DRAM cache design works best across different memory configurations and applications. By taking advantage of the flexibility of tag and data layout in DRAM caches, we proposed a novel reconfigurable DRAM cache architecture, MDC, which applies the best cache configuration dynamically. MDC supports three configuration modes consisting of MDC_DM, MDC_8way, and MDC_page. Using a periodic sampling technique, it chooses and reconfigures to the best performing cache configuration. By choosing the best one dynamically, our evaluation showed that MDC outperforms three fixed DRAM cache designs by 16.7%, 20.2%, and 29.8%.

REFERENCES

Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. 631–644.

Kursad Albayraktaroglu, Aamer Jaleel, Xue Wu, Manoj Franklin, Bruce Jacob, Chau-Wen Tseng, and Donald Yeung. 2005. BioBench: A benchmark suite of bioinformatics applications. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'15)*. 2–9.

Mike Amidi. 2016. NVDIMM-X deliver DRAM performance at NAND capacity *(Flash Memory Summit'16)*.

W. Cheong, C. Yoon, S. Woo, K. Han, D. Kim, C. Lee, Y. Choi, S. Kim, D. Kang, G. Yu, J. Kim, J. Park, K. Song, K. Park, S. Cho, H. Oh, D. D. G. Lee, J. Choi, and J. Jeong. 2018. A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time. In *IEEE International Solid - State Circuits Conference (ISSCC'18)*. 338–340.

Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2014. 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 (MICRO-47)*. 1–12.

Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2015. BEAR: Techniques for mitigating bandwidth bloat in gigascale DRAM caches. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 198–210.

Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2016. CANDY: Enabling coherent DRAM caches for multi-node systems. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-49)*. 1–13.

John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17.

Cheng-Chieh Huang and Vijay Nagarajan. 2014. ATCache: Reducing DRAM cache latency via a small SRAM tag cache. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT'14)*. 51–60.

Hakbeom Jang, Yongjun Lee, Jongwon Kim, Youngsok Kim, Jangwoo Kim, Jinkyu Jeong, and Jae W. Lee. 2016. Efficient footprint caching for tagless DRAM caches. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. 237–248.

Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison cache: A scalable and effective die-stacked DRAM cache. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. 25–37.

Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. 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 (ISCA'13)*. 404–415.

Xiaowei Jiang, Niti Madan, Li Zhao, Mike Upton, Ravishankar Iyer, Srihari Makineni, Donald Newell, Yan Solihin, and Rajeev// Balasubramonian. 2010. CHOP: Adaptive filter-based DRAM caching for CMP server platforms. In *The 16th International Symposium on High-Performance Computer Architecture (HPCA'10)*. 1–12.

Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS design for heterogeneous memory management in datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. 521–534.

Harshad Kasture and Daniel Sánchez. 2016. Tailbench: A benchmark suite and evaluation methodology for latency-critical applications. In *IEEE International Symposium on Workload Characterization (IISWC'16)*. IEEE Computer Society, 3–12.

Kwangwon Koh, Kangho Kim, Seunghyub Jeon, and Jaehyuk Huh. 2019. Disaggregated cloud memory with elastic block management. *IEEE Trans. Comput.* 68, 1 (2019), 39–52.

D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong. 2014. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'14)*. 432–433.

Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W. Lee. 2015. A fully associative, tagless DRAM cache. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. 211–222.

Gabriel H. Loh. 2009. Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*. 201–212.

Gabriel H. Loh and Mark D. Hill. 2011. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. 454–464.

Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 126–136.

Micron 2016. *576Mb: x18, x36 RLDRAM 3*. Micron.

Kyle J. Nesbit, Nidhi Aggarwal, James Laudon, and James E. Smith. 2006. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*. 208–222.

J. Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *IEEE Hot Chips 23 Symposium (HCS'11)*. 1–24.

Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. 2008. Set-dueling-controlled adaptive insertion for high-performance caching. *IEEE Micro* 28, 1 (Jan. 2008), 91–98.

Moinuddin K. Qureshi and Gabe H. Loh. 2012. 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 (MICRO-45)*. 235–246.

Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. 24–33.

Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.* 10, 1 (Jan. 2011), 16–19.

Samsung Electronics 2017. *4Gb E-die DDR4 SDRAM*. Samsung Electronics. Rev 1.6.

Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. 475–486.

Jaewoong Sim, Alaa R. Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent hardware management of stacked DRAM as part of memory. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. 13–24.

Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. 2016. Knights landing: Second-generation intel xeon phi product. *IEEE Micro* 36, 2 (March 2016), 34–46.

spec.org 2017. *SPEC CPU2017 Documentation*. Retrieved on Oct. 1, 2018 from https://www.spec.org/cpu2017/Docs/.

Po-An Tsai, Nathan Beckmann, and Daniel Sanchez. 2017. Jenga: Software-defined cache hierarchies. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. 652–665.

Stavros Volos, Djordje Jevdjic, Babak Falsafi, and Boris Grot. 2017. Fat caches for scale-out servers. *IEEE Micro* 37, 2 (Mar. 2017), 90–103.

H.-S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. 2010. Phase change memory. *Proc. IEEE* 98, 12 (Dec. 2010), 2201–2227.

Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas. 2017. Banshee: Bandwidth-efficient DRAM caching via software/hardware cooperation. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*. 1–14.

Received October 2018; revised May 2019; accepted May 2019