Improving Address Translation in Tagless DRAM Cache by Caching PTE Pages

Osang Kwon, Yongho Lee, and Seokin Hong
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Republic of Korea
{*osang915, jhyn205, seokin*}*@skku.edu*

***Abstract*—This paper proposes a novel caching mechanism for PTE pages to enhance the Tagless DRAM Cache architecture and improve address translation in large on-package DRAM caches. Existing OS-managed DRAM cache architectures have achieved significant performance improvements by focusing on efficient tag management. However, prior studies primarily update PTEs after caching data pages without directly accessing them from the DRAM cache, leading to performance degradation during page walks.**

To address this, we propose a method to simultaneously cache data pages and PTE pages in the DRAM cache. This approach reduces address translation latency and cache access delays. Additionally, we introduce a shootdown mechanism to maintain PTE and page walk cache consistency in multi-core systems, ensuring all cores access the latest information for shared pages. Experimental results show that caching PTE pages reduces address translation overhead by up to 33.3% compared to traditional OS-managed tagless DRAM caches, improving overall program execution time by an average of 10.5%. This method effectively mitigates bottlenecks caused by address translation.

***Index Terms*—DRAM Cache, Virtual Memory, Page Table**

I. Introduction

Modern applications demand vast memory capacity and high bandwidth, which cannot be satisfied by a single type of memory. To address the memory wall problem, many studies propose heterogeneous memory systems that combine DRAM with large-capacity non-volatile memory [20], [35]. These systems use high-speed memory as a cache while employing large non-volatile memory as primary storage for rapid access. Researchers are increasingly focusing on efficient memory access methods tailored to the data characteristics in heterogeneous memory systems. As a result, heterogeneous memory systems are emerging as promising solutions for addressing major memory bottlenecks in next-generation computing [24], [31].

Recently, die-stacked DRAM technology has been widely adopted by major processor manufacturers to enhance the capacity and energy efficiency of memory systems [2], [32]. While this technology offers significant benefits, it is unlikely to replace physical memory entirely, especially for applications requiring large memory capacities, such as graph analytics, in-memory databases, genome analysis, and large AI models. To address these limitations, researchers have proposed using die-stacked DRAM effectively as DRAM caches [13], [14]. A key challenge in DRAM cache design is minimizing the overhead associated with cache tag management. For example, a 1GB DRAM cache requires approximately 128MB of tag storage. Storing tags in SRAM can reduce read overhead but becomes impractical due to the large tag size. Alternatively, tags can be stored in DRAM alongside the data, but this introduces additional latency due to tag read operations [8], [12]. To overcome these challenges, OS-based tag management techniques have been proposed, commonly referred to as Tagless DRAM Cache (TDC) [10], [17], [21]. TDC exposes on-package DRAM to the OS, allowing pages to be copied from DRAM to the DRAM cache. By storing DRAM cache addresses in Page Table Entries (PTEs) and Translation Lookaside Buffers (TLBs), TDC eliminates the overhead of additional tag reads. Recent research [17], [18] shows that hardware support can transform blocking operations in TDC's page copying process into non-blocking operations, further optimizing performance.

Despite progress in tag management and blocking operation optimization, challenges remain. Virtual memory address translation, integral to TDC's functionality, has been identified as a significant bottleneck, accounting for up to 30% of application execution time in some workloads [28]. To address this, some studies have proposed new page table structures or modified TLB designs to improve address translation [7], [19], [33]. In TDC, address translation maps virtual memory addresses to DRAM cache addresses instead of physical memory addresses. However, the lookup process still relies on physical addresses for accessing the SRAM cache and main memory. This reliance on off-chip memory for PTE lookups exacerbates performance degradation, particularly in workloads with large memory footprints or random memory access patterns, due to frequent TLB misses [29].

This paper emphasizes the need to cache not only normal data pages but also PTE pages to achieve fast tag access in TDC. As illustrated in Figure 1, our approach signifi-

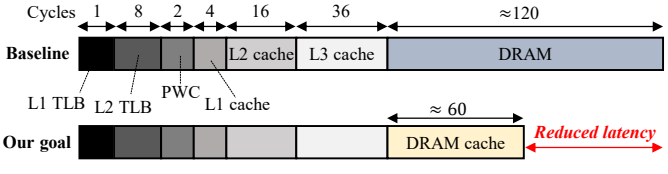

Fig. 1. Address translation latency for Tagless DRAM cache. Our goal is to enable PTE fetch in DRAM cache to reduce latency.

978-3-9826741-0-0/DATE25/© 2025 EDAA

cantly reduces latency by fetching PTEs directly through the DRAM cache. Frequent TLB misses often require repeated PTE lookups, and searching for PTEs in off-chip memory (DRAM) can lead to substantial overhead. By caching PTE pages from DRAM to DC, we aim to minimize this latency. In modern systems with multi-level page tables [9], the challenges of address translation are further exacerbated, making the proposed DRAM cache approach even more beneficial. Additionally, to address potential coherency issues arising from caching PTEs in the DRAM cache, we introduce selective caching mechanisms to reduce unnecessary overhead. This method reduces address translation overhead by an average of 33.3% and improves program execution time by an average of 10.5%.

II. Background and Motivation

A. DRAM cache

Extensive research has been conducted on systems that leverage heterogeneous memory to address the bandwidth and capacity challenges posed by single main memory architectures. Various memory types, such as non-volatile memory (NVM) and high-bandwidth memory (HBM), which differ significantly from traditional main memory, have been explored. In this study, we focus on systems that integrate high-bandwidth on-package DRAM, which provides substantial performance benefits. Specifically, we analyze systems that utilize on-package DRAM as a DRAM cache. DRAM caches can be classified into two types based on how they manage tags: hardware-based and software-based approaches, both of which will be discussed.

Hardware-based dram caches: HW-based DRAM caches apply MSHR (Miss Status Holding Register)-based miss handling, similar to traditional cache hierarchy structures. By processing memory requests concurrently, even when multiple misses occur, the DRAM cache can adopt a non-blocking structure by recording these misses in the MSHRs. This hardware-based non-blocking miss handling provides a performance advantage by enabling parallel miss tracking without needing to wait for multiple requests [17]. By offloading miss handling to the MSHR, the DRAM cache can handle other memory requests instead of waiting for the missed requests to complete.

However, this approach does not always guarantee performance benefits. The fundamental challenge in adopting DRAM caches lies in managing metadata [22], [30]. This metadata includes cache tags, valid bits, and dirty bits, which are crucial for ensuring the accuracy of cache data. Updating metadata such as valid or dirty bits in DRAM, or checking cache tags to process memory requests, can lead to significant performance degradation. In DRAM caches, just as in typical cache operations, the hit or miss of a request must be determined through tag checks, which requires fetching these tags from DRAM. Even if tags and data can be fetched in a single burst, the need for multiple bursts to verify tags makes it challenging to implement a set-associative structure. Ultimately, the additional memory bandwidth required for metadata verification and updates can result in performance degradation.

Software-based dram caches: In contrast to the MSHR-based miss handling design, a DRAM cache managed by the OS, where tags are handled by the OS, was proposed in previous research [10], [21], [27]. In OS-managed designs, data pages are copied between the on-package DRAM cache (DC) and main memory at the granularity of pages. This approach is often referred to as a "Tagless DRAM cache" because the system does not check tags within the DRAM cache itself. Instead, the presence of data in the DRAM cache is determined by the PTE.

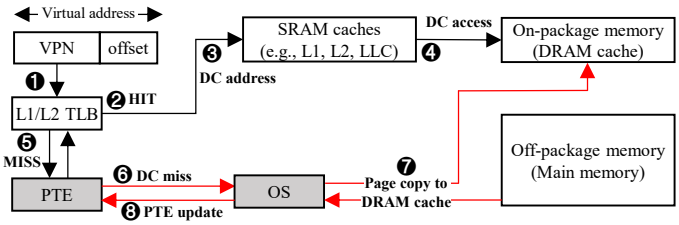

Fig. 2. OS-managed DRAM cache (Tagless DRAM cache).

Figure 2 illustrates the operation of a Tagless DRAM cache. We depict the various actions performed when fetching a data page. First, each process manages memory independently using virtual memory. The virtual memory must be translated into a cache address (CA) through the TLB (❶). If there is a TLB hit (❷), the system can directly access the SRAM cache using the translated CA (❸). If the data cannot be found in the SRAM cache, it can be retrieved from the DC instead of accessing the main memory (❹). However, if a TLB miss occurs, the PTE must be checked for address translation (❺). Although Figure 2 shows a single PTE, in reality, PTEs are organized in a multi-level structure, and the process of checking multiple PTE levels is referred to as a page walk. If the CA cannot be found after checking the PTE, the page does not exist in the DC, requiring OS intervention for miss handling (❻). During this handling, data needs to be copied from main memory to the DRAM cache (❼), and associated metadata and the PTE must be updated (❽). Afterward, memory access can be attempted again using the CA.

B. Prior work

The primary limitation of OS-based DRAM caches is the need to stall applications during tag miss handling. Fundamentally, OS-based architectures rely on the TLB and PTE rather than direct tag checking. If address translation through the TLB and PTE successfully maps a VA to a CA, the data should already be accessible at the CA, indicating that the data copy to the DRAM cache has been completed. For this reason, handling a DRAM cache miss requires stalling the application during the PTE update and data copying process, leading to significant performance degradation. This behavior is referred to as a "blocking cache," where a single DC miss can result in a performance penalty of several thousand cycles. Recent research [17] proposes a method to mitigate these stalls by decoupling the PTE update and data copy processes. In this method, the PTE is updated first, allowing the application to resume execution, while the data copy is offloaded to back-end hardware. The hardware then verifies the presence of data at

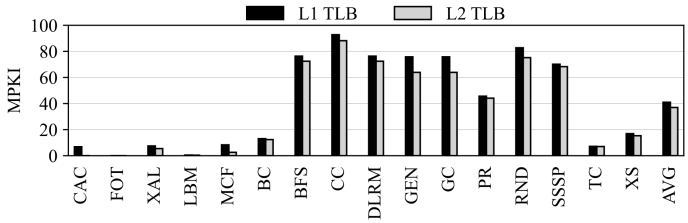

Fig. 3. L1 TLB and L2 TLB Miss Per Killo Instructions (MPKI).

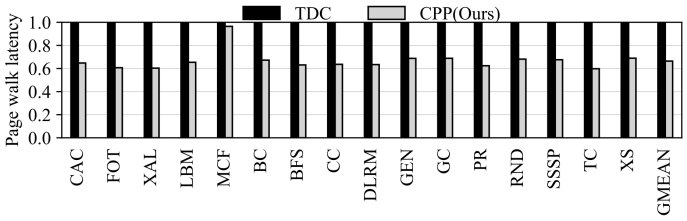

Fig. 4. Page Walk Latency based on PTE lookup location (Normalized to DRAM access).

the time of actual DRAM cache access, significantly reducing the latency caused by DRAM cache miss handling.

C. Comparative Analysis of TLB Performance

Among various DRAM cache architectures, OS-based tag management has shown promising results in addressing the drawbacks of miss handling in recent studies. While we adopt the strengths of OS-based management, we also analyze the fundamental limitations of DRAM cache structures that rely on TLB and PTE. In virtual memory systems, the process of translating a VA to a PA through the TLB and PTE has always been necessary. However, this process presents slightly different challenges in DC systems. In current DC systems, when a data page is copied from main memory to the DC, the corresponding PTE metadata is updated. However, the PTE must still access the main memory address. Even if the data page resides in the DC, the data access latency can vary significantly depending on the address translation process.

We analyze the address translation latency involved in DC address translation within DC systems. Modern workloads often exhibit large data sizes and high address randomness, which frequently cause TLB misses. While various studies have explored ways to improve TLB performance, we focus on analyzing this issue within the basic TLB structure. Figure 3 shows the Misses Per Kilo Instructions (MPKI) for L1 TLB and L2 TLB across various workloads. The workloads used in our experiments and the experimental setup are detailed in Table II. We selected memory-intensive workloads from the SPEC benchmark suite (SPEC) and also included large workloads with significant memory footprints, such as graph analysis and recommendation models. There is a significant variation in TLB MPKI depending on the workload, and TLB-sensitive workloads require frequent page walks. On average, SPEC workloads exhibit an STLB MPKI of 1.69, while non-SPEC workloads show a much higher average MPKI of around 53. We analyze the impact of frequent TLB misses and resulting PTE accesses on DC performance and ultimately focus our follow-up analysis on workloads with high TLB sensitivity.

D. Page Walk in off-package DRAM

Address translation is a crucial operation for data access in virtual memory systems. Even when using a DC, the core task remains the same, with the only difference being the translation to a CA via a page walk. Radix tree-based multi-level page tables trigger sequential memory accesses, which can account for up to 30% of total program execution time [11], [15]. In DC systems, if most address translations are resolved at the TLB level, the impact of PTE read latency could be minimal. However, as shown in Figure 3, TLB misses can vary significantly across workloads, making page walks an essential consideration. To address this translation overhead, we propose improving the PTE read process. Currently, data is copied from DRAM to the DC, and copying PTEs alongside the data could significantly reduce page walk latency. Figure 4 shows the difference in page walk latency when PTEs are accessed from DRAM versus the DC, assuming identical operations in the Memory Management Unit (MMU) cache and cache hierarchy. We denoted the baseline of the Tagless DRAM cache as TDC, and the mechanism that we propose is denoted as CPP. While the difference is negligible when PTE access frequency in DRAM is low due to high locality, on average, accessing PTEs from the DRAM cache yields approximately a 33.3% latency reduction. We emphasize the importance of PTE lookup in DC and propose a new mechanism that copies PTEs to the DC along with the data.

III. Caching PTE page

A. Overview: PTE page caching

We highlight the limitations of PTE management in existing OS-managed DRAM cache architectures and propose a new mechanism. In traditional systems, page copies from DRAM to the DRAM cache are followed by PTE updates in DRAM, enabling efficient access to the DRAM cache without tags. However, as address translation has become a significant bottleneck, we propose a **Caching PTE Pages** mechanism, where both the data page and the PTE page are copied to the DRAM cache during this process. Figure 5 illustrates the traditional flow of OS-managed DRAM cache and the additional steps introduced by Caching PTE Pages (CPP). We focus on the process of translating a VA to a CA and provide a detailed diagram of this process. When a TLB miss occurs and the VA cannot be converted to a CA, a PTE lookup is triggered (❶). This lookup traverses multiple levels, with some levels cached in the Page Walk Cache (PWC) [3]. Fundamentally, however, each level must still be accessed through either the SRAM cache or DRAM. This entire process is known as a page walk (❷). The page walk begins by locating the base address in a register at the highest level and traversing each level's PTE until the final PTE is found. The final PTE determines whether the requested page exists in the DRAM cache. If the page resides in DRAM, it is copied to the DRAM cache, and

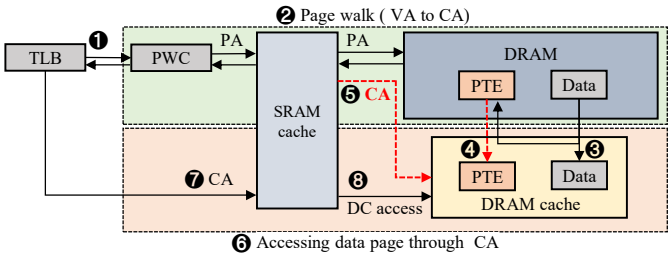

Fig. 5. Overview of Caching PTE pages (CPP). This figure illustrate Address translation from VA to DC address and data page access through DC address.

the PTE is updated accordingly (❷). The PTE then provides the CA, which is updated in the upper-level caches (PWC and TLB).

To mitigate the latency caused by frequent TLB misses and repeated page walks, we propose copying the PTE page alongside the data page to enable direct CA-based PTE access (❸). The copied PTE's address is updated in the higher-level PTE. Once both the data and PTE pages are copied, subsequent page walks can quickly access the PTE via the CA (❹). The process of accessing data through the translated CA remains the same as in traditional methods (❺): the CA is used to request the required data from the SRAM cache (❻), and if the requested page is not found in the SRAM cache, the DC is accessed to retrieve the data (❼).

B. Caching Selectively: Only What Matters

When caching a data page from DRAM to the DRAM cache, we ensure that the related metadata and PTE are updated. For instance, in the case of the PTE, the address in the last-level PTE entry must be updated from the PA to the CA. Similarly, when caching the PTE, it is crucial to record the updated address in the higher-level PTE to prevent incorrect address translation. During this process, invalid entries in the PWC that store outdated PTEs require shootdowns. To minimize the overhead caused by frequent shootdowns, we do not copy all levels of PTEs to the DRAM cache, but selectively copy only specific levels.

Figure 6 shows the criteria used to determine which PTEs should be copied. PTEs are organized into multiple levels, and the PWC can cache PTEs at each level except for the last level. Through experiments, we evaluated the hit rate of each level's page walk cache. The higher-level (L4, L3) PWCs generally show high hit rates, making it unnecessary to copy these entries to the DRAM cache. However, the L2 PWC, despite being

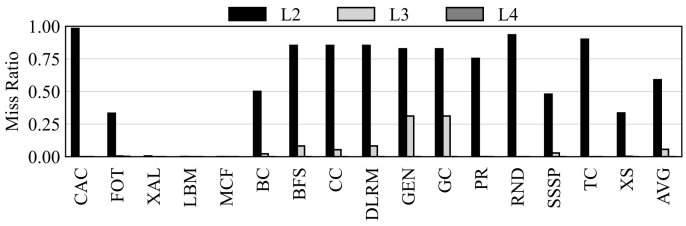

Fig. 6. Cache miss ratio for multi-level Page Walk Cache. L4 is the root-level cache.

cached, shows an average miss rate of 59% and a maximum of 98%, making it a significant target for caching. Additionally, the last level (L1) is not cached within the PWC, so copying it to the DRAM cache is essential. Based on these observations, we adopt a strategy that caches only the last-level PTE and the level directly above it (L2) in the DRAM cache. This approach optimizes performance while minimizing overhead. The additional overhead introduced by PTE copying will be analyzed in the next section.

C. Multi-core support : Page Walk Cache coherence

Maintaining the consistency of private caches across cores in a multi-core system is crucial, especially when shared data is updated. In such cases, all cores must have access to the latest data. Among the components, we focus on the consistency of the TLB and PWC. For example, if a page is evicted from the DRAM cache to DRAM, and the TLB still holds a valid entry with the evicted page's CA, this could lead to incorrect page accesses. During the page copy or eviction process, the TLB must be flushed. In multi-core systems, this process involves flushing the relevant entries from all cores, which is known as TLB shootdown [1], [4]. TLB shootdown can introduce significant overhead due to the need to wait for responses from multiple cores.

Similarly, **Caching PTE pages** (CPP) can cause consistency issues with PTE addresses, similar to those encountered with the TLB. To address this, we propose two solutions. The first approach is to avoid PTE page caching. Since PTEs are copied rather than migrated, page walks can still proceed using the PA stored in the PWC. However, this method has limitations as it requires waiting for the PWC entry to be replaced before the PTE can be fetched from the DRAM cache. Furthermore, if the PTE page is selected as a victim in the DRAM cache, it becomes difficult to apply the same approach.

Therefore, we adopt the second approach: maintaining the consistency of the PWC. Just as TLB consistency is crucial when caching data pages, the same principle applies to the PWC. Located in the MMU, the PWC serves as a private cache for each core. When PTE information or location is updated, we ensure consistency by performing a PWC shootdown to keep the entries up-to-date, much like a TLB shootdown. Although frequent shootdowns can introduce overhead, the advantages of PTE caching outweigh the drawbacks because a single caching operation can cover a large address space. For example, a single PTE page can cover at least 2MB of address space at the last level, and up to 1GB at higher levels. Since each entry handles a large address space, PWC shootdowns do not significantly impact overall program execution (as we will demonstrate through experiments). This method enables accurate and reliable caching of PTE pages to the DRAM cache, making it a robust solution.

IV. Experimental Evaluation

A. Experimental Methodology

We model the proposed **Caching PTE pages** mechanism based on the Sniper [6] simulator. Table I summarizes the

system configuration used in our experiments. The L1 and L2 caches within each core are private, while the L3 cache is a shared resource among all cores. To incorporate a virtual memory system, we extend the Sniper [16] simulator, adding private L1/L2 TLBs and multi-level page walk caches for each core. In addition, we configure a heterogeneous memory system consisting of HBM and DDR. The modeled memory system includes 512MB of in-package DRAM and 16GB of external main memory, both shared by all cores.

Based on the TLB MPKI shown in Figure 3, our objective is to ensure that workloads with low TLB MPKI do not experience any performance degradation due to our PTE management techniques, while achieving performance improvements in TLB-sensitive workloads.

We manage the DRAM cache using an OS-managed tagless DRAM cache [10] and adopt a non-blocking page copying mechanism, as proposed in recent studies [17], as the baseline. Using this tagless DRAM cache (TDC), we evaluate the performance of the **Caching PTE pages** (CPP) mechanism, which aims to improve address translation efficiency in the DRAM cache. For comparison, we also adopt a baseline system (BASE) that uses only main memory to provide the lower bound of the DRAM cache performance and evaluate our improvements.

B. Performance

Figure 7 compares the performance of BASE, which does not use a DRAM cache (DC), with TDC and our proposed CPP.

TABLE I
SIMULATED SYSTEM CONFIGURATION

Component	Parameter
CPU	4-way Out of Order, 2.66GHz
L1 I/D-TLB	64-entry, 4-way, 1-cycle
L2 TLB	1,536-entry, 12-way, 8-cycle,
Page Walk Cache	3-level Split PWC, 4-way; PGD: 16-entry; PUD: 16-entry; PMD: 32-entry; 2-cycle
L1 I/D-cache	32KB, 4-way, LRU, 4-cycle
L2 Cache	512KB, 16-way, LRU, 16-cycle
LLC	8MB, 16-way, LRU, 36-cycle
DRAM cache On-package DRAM	HBM2, 2GHz, 128-bit, 64-byte burst size, 16 banks/rank, 1 rank/channel, 8 channels, 4KB/row, open-page policy, tRCD-tCAS-tRP-tRAS 7-7-7-17
Main memory Off-package DRAM	DDR4-3200, 64-bit, 64-byte burst size, 16 banks/rank, 2 rank/channel, 2 channels, 4KB/row, open-page policy, tRCD-tCAS-tRP-tRAS 14-14-14-34

TABLE II
WORKLOADS

Suite	Workloads	Memory
SPEC2017 [5]	507.cactuBSSN_r (CAC), 549.fotonik3d_r (FOT), 523.xalancbmk_r (XAL), 519.lbm_r (LBM), 505.mcf_r (MCF)	477MB ∼ 6721MB
GraphBIG [25]	Betweeness Centrality (BC), Breadth-first search (BFS), Connected components (CC), Graph coloring (GC), PageRank (PR), Triangle counting (TC), Shortest-path (SSSP)	8GB
GenomicsBench [34]	K-length substring of DNA sequence counting (GEN)	33GB
HPCC [23]	Giga updates per second (RND)	10GB
XSBench [36]	Monte Carlo neutron transport (XS)	9GB
DLRM [26]	Sparse-length sum (DLRM)	10.3GB

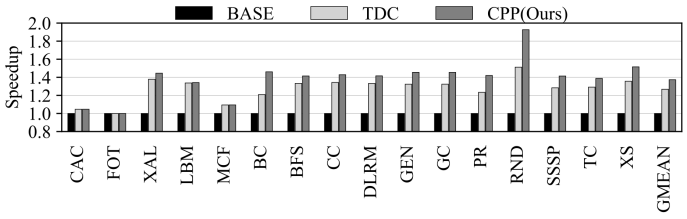

Fig. 7. Performance of baseline (BASE), Tagless Dram Cache (TDC), and Caching PTE Pages (CPP) (normalized to the BASE).

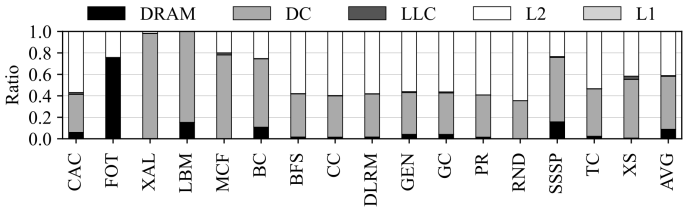

Fig. 8. Fetching location breakdown of memory requests for page table entries.

We evaluate performance using IPC, normalized based on the performance of BASE. In the case of non-blocking TDC, there is an average performance improvement of 26.7% compared to the BASE system. For workloads that exhibit high locality in the SRAM cache (e.g., CAC, FOT), the performance improvement from DC is relatively small compared to BASE. However, in most workloads, DC provides a significant performance improvement, with a maximum enhancement of 51%. CPP further enhances performance by significantly reducing address translation overhead, as shown in Figure 4 for the existing TDC. By caching selected PTEs in the DC, CPP achieves an average performance improvement of 10.5% and a maximum improvement of 41.4%. In some workloads with low TLB MPKI, no performance improvement is observed. However, this result occurs in workloads where address translation is not a major bottleneck. Importantly, even in low TLB MPKI workloads, CPP does not cause any additional performance degradation.

C. Cache characterization

Figure 8 shows the details of memory requests related to address translation in CPP. This figure illustrates how many address translation requests, which were previously fetched from DRAM (main memory) in the existing TDC, are now converted to DC requests in CPP. PTEs are initially fetched from main memory and then cached in the DC, enabling efficient searches. On average, 41.8% of PTEs are read from the SRAM cache, while the remaining requests must access memory (DRAM or DC). CPP handles 84.7% of the 58.1% memory requests via the DC, reducing page walk latency by 33%. Additionally, access to the SRAM cache is achieved using the cache address (DC address). The proportion of fetches from the L1 cache and LLC is relatively small, while 41.1% of translation information is retrieved from the L2 cache. As a result, CPP effectively retrieves PTEs through the SRAM cache

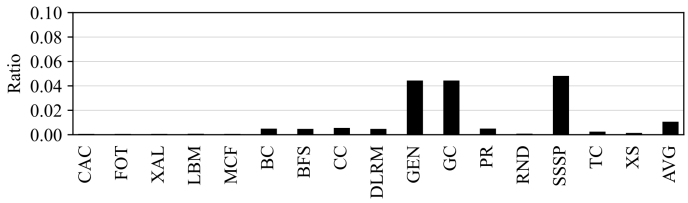

Fig. 9. Ratio of Page Walk Cache(PWC) Shootdowns.

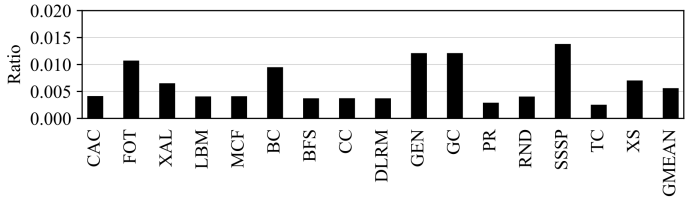

Fig. 10. Increased page copy ratio due to PTE page caching.

and significantly reduces main memory dependency by utilizing the DC.

D. Additional CPP overhead analysis

Page Walk Cache: In CPP, as explained in Section III-C, coherency handling for the PWC is necessary due to PTE page caching. Figure 9 shows the frequency of PWC shootdowns during program execution. Proper handling is required when PTEs are newly cached or evicted from the DC. We measure and present the frequency of PWC shootdowns among the total memory requests. On average, PWC shootdowns account for only 1% of memory requests, with a maximum occurrence of 4.7%. This indicates that shootdowns occur at a very low frequency. Since PTE pages cover a wide address range, they are infrequently replaced, enabling significant performance improvements with minimal overhead.

Page copy: We also evaluate the amount of page copying induced by caching from main memory to the DC. Figure 10 shows the percentage increase in page copying due to PTE pages among the total pages cached in the DC. Our experimental results indicate that page copying has increased by approximately 0.54%. PTEs can store address translation information as 8-byte entries based on the default page size (4KB), representing about 0.2% of the total. While ideally, a 0.2% increase would be expected, not all entries are used even when a single PTE page is copied. Furthermore, as upper-level PTE pages are also copied to achieve a greater effect, a slight increase in page copying may occur. The results vary depending on the utilization of entries within the PTE page, leading to different outcomes for each workload. The maximum increase is approximately 1.3%, which represents an effective trade-off for reducing page walk latency.

DC miss frequency: Finally, Figure 11 shows the changes in DC misses caused by CPP. Unlike the existing TDC, which stores only data pages in the limited space of the DC, CPP additionally stores PTE pages. This may increase the likelihood of data pages being selected as victims, potentially resulting in more misses. We compare and analyze the data page MPKI

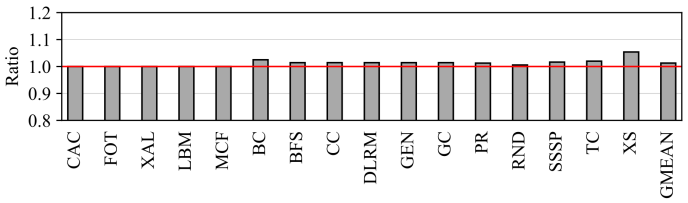

Fig. 11. Impact of Caching PTE pages on Data Page MPKI in DC (normalized for TDC).

between TDC, which does not store PTE pages, and CPP. Figure 11 is normalized based on TDC, as indicated by the red line in the figure. On average, CPP induces approximately 1.2% additional misses, with a maximum increase of 5.3%. However, as shown in Figure 8, CPP significantly improves PTE fetching from the DC, which can lead to increased misses for some data pages. Despite this slight increase in data page misses, CPP achieves an overall improvement in program execution time.

V. Conclusion

In this paper, we proposed the **Caching PTE Page (CPP)** technique to address the address translation overhead of the **Tagless DRAM Cache (TDC)**. Existing research on DRAM caches has primarily focused on effective tag management techniques, with significant attention given to TDC for blocking operations aimed at efficient page caching. However, the diversity and evolution of workloads, particularly those utilizing big data, have exposed the limitations of TLBs. To effectively mitigate the overhead of page walks caused by frequent TLB misses, CPP caches PTE pages in the DC, an approach not previously explored. Additionally, we analyzed potential consistency issues, such as page walk cache shootdowns resulting from PTE page caching, to ensure effective management of PTE pages. CPP achieves an average reduction of 33.3% in address translation overhead compared to existing TDC, resulting in a performance improvement of 10.5%.

Acknowledgment

This work was partly supported by the Ministry of Science and ICT (MSIT) under the Information Technology Research Center (ITRC) support program (No.II212052, 50%) supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP) and Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.II221170, 50%). Seokin Hong is the corresponding author.

References

[1] N. Amit, A. Tai, and M. Wei, "Don't shoot down tlb shootdowns!" in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–14.

[2] S. Anthony, "Intel unveils 72-core x86 knights landing cpu for exascale supercomputing," 2013.

[3] T. W. Barr, A. L. Cox, and S. Rixner, "Translation caching: skip, don't walk (the page table)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 48–59, 2010.

[4] D. L. Black, R. F. Rashid, D. B. Golub, and C. R. Hill, "Translation lookaside buffer consistency: A software approach," *ACM SIGARCH Computer Architecture News*, vol. 17, no. 2, pp. 113–122, 1989.

[5] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 41–42.

[6] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[7] K. Gosakan, J. Han, W. Kuszmaul, I. N. Mubarek, N. Mukherjee, K. Sriram, G. Tagliavini, E. West, M. A. Bender, A. Bhattacharjee *et al.*, "Mosaic pages: Big tlb reach with small pages," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 433–448.

[8] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014, pp. 51–60.

[9] Intel, *5-Level Paging and 5-Level EPT White Paper*, Intel, 2018.

[10] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 237–248.

[11] S. Jang, J. Park, O. Kwon, Y. Lee, and S. Hong, "Rethinking page table structure for fast address translation in gpus: A fixed-size hashed page table," in *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*, 2024, pp. 325–337.

[12] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 25–37.

[13] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 404–415, 2013.

[14] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Adaptive filter-based dram caching for cmp server platforms," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.

[15] K. Kanellopoulos, H. C. Nam, N. Bostanci, R. Bera, M. Sadrosadati, R. Kumar, D. B. Bartolini, and O. Mutlu, "Victima: Drastically increasing address translation reach by leveraging underutilized cache resources," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1178–1195.

[16] K. Kanellopoulos, K. Sgouras, and O. Mutlu, "Virtuoso: An open-source, comprehensive and modular simulation framework for virtual memory research," *arXiv preprint arXiv:2403.04635*, 2024.

[17] Y. Kim, H. Kim, and W. J. Song, "Nomad: Enabling non-blocking os-managed dram cache via tag-data decoupling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 193–205.

[18] Y. Kim and W. J. Song, "Genie cache: Non-blocking miss handling and replacement in page-table-based dram cache," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 983–996.

[19] O. Kwon, Y. Lee, J. Park, S. Jang, B. Tak, and S. Hong, "Distributed page table: Harnessing physical memory as an unbounded hashed page table," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 36–49.

[20] Y. Lee, O. Kwon, and S. Hong, "Don't open row: rethinking row buffer policy for improving performance of non-volatile memories," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 823–828.

[21] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," *ACM SIGARCH computer architecture news*, vol. 43, no. 3S, pp. 211–222, 2015.

[22] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 454–464.

[23] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, vol. 213, no. 10.1145, 2006, p. 1.

[24] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 742–755.

[25] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "Graphbig: understanding graph computing in the context of industrial solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[26] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[27] M. Oskin and G. H. Loh, "A software-managed approach to die-stacked dram," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 188–200.

[28] C. H. Park, I. Vougioukas, A. Sandberg, and D. Black-Schaffer, "Every walk's a hit: making page walks single-access cache hits," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 128–141.

[29] J. Park, O. Kwon, Y. Lee, S. Kim, G. Byeon, J. Yoon, P. J. Nair, and S. Hong, "A case for speculative address translation with rapid validation for gpus," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 278–292.

[30] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 235–246.

[31] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 85–95.

[32] I. Z. Reguly, "Comparative evaluation of bandwidth-bound applications on the intel xeon cpu max series," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1236–1244.

[33] D. Skarlatos, A. Kokolis, T. Xu, and J. Torrellas, "Elastic cuckoo page tables: Rethinking virtual memory translation for parallelism," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1093–1108.

[34] A. Subramaniyan, Y. Gu, T. Dunn, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das, "Genomicsbench: A benchmark suite for genomics," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 1–12.

[35] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong *et al.*, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 105–121.

[36] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis," *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.