Unified-TP: A Unified TLB and Page Table Cache Structure for Efficient Address Translation

Zhulin Ma[1], Yujuan Tan[1], Hong Jiang [2], Zhichao Yan[2], Duo Liu[1], Xianzhang Chen[1],
Qingfeng Zhuge[3] and Edwin Hsing-Mean Sha[3], Chengliang Wang[1]
[1]*College of Computer Science, Chongqing University, China*
[2]*Department of Computer Science and Engineering, University of Texas at Arlington, USA*
[3]*Department of Computer Science and Software Engineering, East China Normal University, China*
{mazhulin1993, tanyujuan, yanzhichao.hust, xzchen109, qfzhuge, edwinsha }@gmail.com,
hong.jiang@uta.edu, {liuduo,wangcl}@cqu.edu.cn

***Abstract*—To improve the performance of address translation in applications with large memory footprints, techniques, such as hugepages and HW coalescing, are proposed to increase the coverage of limited hardware translation entries by exploiting the contiguous memory allocation to lower Tanslation Lookaside Buffer (TLB) miss rate. Furthermore, Page Table Caches (PTCs) are proposed to store the upper-level page table entries to reduce the TLB miss handling latency. Both increasing TLB coverage and reducing TLB miss handling latency have proved to be effective in speeding up address translation, to a certain extent. Nevertheless, our preliminary studies suggest that the structural separation between TLBs and PTCs in existing computer systems makes these two methods less effective because they are exclusively used in TLBs and PTCs respectively. In particular, the separate structures cannot dynamically adjust their sizes according to the workloads, resulting in low resource utilization and inefficient address translation. To address these issues, we propose a unified structure, called Unified-TP, which stores PTC and TLB entries together. Besides, Our modified LRU algorithm helps identify the cold TLB and PTC entries and dynamically adjust the numbers of TLB and PTC entries to adapt to different workloads. Furthermore, we introduce a scheme of parallel search when receiving memory access requests. Our experimental results show that Unified-TP can reduce the numbers of TLB misses by an average of 35.69% and improve the performance by an average of 11.12% compared with separately structured TLBs and PTCs.**

***Index Terms*—Virtual Memory, Translation Lookaside Buffers, Page Table Caches**

I. Introduction

With the increase of memory capacity and the emergence of applications with large datasets, virtual-to-physical address translation has become a critical performance bottleneck. First, for a given page size, the number of address translation records is directly proportional to the memory capacity. Thus, the higher the memory capacity is, the larger the number of address translation records will be. Since TLBs cover only a portion of translations, increasing the number of translations will decrease the TLB hit rate. The more TLB misses, the more long-latency memory accesses for page table are required. Therefore, the performance of address translation decreases as memory capacity increases. Second, modern workloads such as graph analytics and in-memory key-value stores use large datasets and typically have non-uniform memory access patterns with weak locality [1]. Specifically, data in these workloads tend to have a long reuse distance, which means that when a data item is accessed again, it is likely that the address translation record for that data item has been deleted from TLBs. As a result, these workloads have a lower TLB hit rate than those with strong access locality.

Recent studies have generally focused on improving the performance of address translation in two ways. The first is to expand the translation coverage of TLBs. For example, prior works [2]–[9] propose using *translation contiguity* and *hugepages* to reduce TLB miss rate. The second focuses on reducing the TLB miss handling latency. For example, the studies [10]–[13] propose using *Page Table Caches* (PTCs) to store upper-level page table entries to reduce long-latency page table walks. Although these methods are efficient in speeding up address translation, they manage TLBs and PTCs separately. In fact, we believe that the structural separation between TLBs and PTCs limits the acceleration of address translation, while judiciously unifying TLBs and PTCs offers promises to further improve the performance of address translation.

Our preliminary studies found that the existing method of separating fixed-size TLBs and PTCs has limited effectiveness in accelerating address translation for two important reasons. First, for modern workloads such as graph analytics and in-memory key-value stores, data tend to have long reuse distance, which leads to high TLB miss rate and incurs heavy memory access overhead. Recent studies show that these workloads can experience up to 50% execution-time overhead due to the page table walks after TLB misses [9], [14]. In addition, although existing methods can use PTCs to cache frequently used upper-level entries of page tables, it still needs at least one memory access to get the final physical address. Therefore, in this case, if some PTC entries can be deleted to shrink the PTC space and provide more space for the TLB entries to accommodate the address translation of data with long reuse distance, it will greatly reduce the memory access overhead and improve the performance of address translation.

Second, for workloads that primarily contain sequential read and write requests (e.g., time series databases and video applications), TLBs are inefficient because there are almost no

repeat requests. But in this case, PTCs works well because of the high spatial locality of these workloads. However, despite of inefficiencies, existing methods retain the entire TLBs. In this case, if we can delete useless TLB entries to reduce TLB space and expand PTC space to store more PTC entries, it can reduce a lot of memory access overhead.

The root cause of the above two problems is the physical and logical separations of TLB and PTC structures that renders its impossible to dynamically adjust their sizes according to the workloads, resulting in low resource utilization and inefficient address translation.

Given the root cause, we propose a unified structure, called **Unified-TP**, which stores PTC and TLB entries together. By combining PTCs and TLBs into a single structure, Unified-TP can dynamically adjust the numbers of TLB and PTC entries based on the access characteristics of workloads. In addition, we design a scheme of parallel search of TLB and PTC entries to improve the search performance. Unified-TP is orthogonal to other address translation acceleration methods such as *translation contiguity* and *hugepages*. It can be used in conjunction with any of them to more effectively improve the performance of address translation.

Our main contributions are listed as follows.

- We conduct in-depth investigation and the results show that the structural separations of TLBs and PTCs prevent them from adapting to different workloads, resulting in a large number of memory accesses.
- We propose Unified-TP, which store TLBs and PTC entries in a single structure and dynamically adjusts the spaces of TLBs and PTCs based on the workload characteristics to speed up address translation.
- We conduct a comprehensive evaluation of Unified-TP. The results show that Unified-TP can reduce the numbers of TLB misses by 35.69% and improve the performance by 11.12% on average.

II. Background and Motivation

A. Background

Address translation from the virtual address space to the physical address space in memory systems is the main cause of performance degradation, especially in the case of virtualization and large working sets [5]. Researchers have generally focused on two approaches to mitigate this cost.

The first aims to lower the TLB miss rate. For example, by allocating contiguous physical pages to contiguous virtual pages, superpages or hugepages [6]–[9] have been proposed to replace hundreds to thousands of basic page translations with a single superpage translation. In addition, recent research [2]–[5] has proposed *translation contiguity* to expand the translation coverage of TLBs. More sophisticated techniques, such as TLB prefetching and scheduling [15]–[17], have also been considered to increase the TLB hit rate.

The second focuses on reducing the number of memory accesses after a TLB miss. For example, prior sudies [10]–[13] have proposed using PTCs to store the upper-level page table

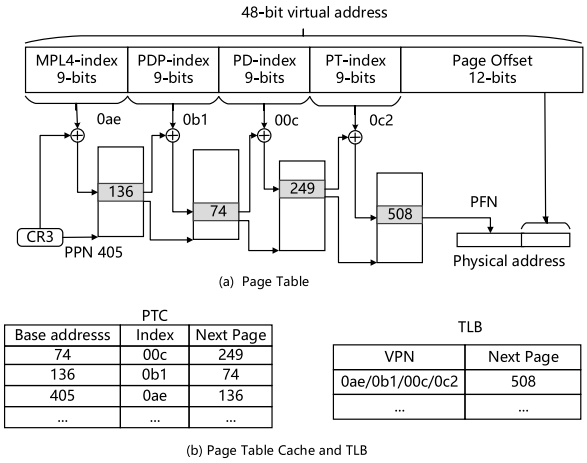

Fig. 1. An example of 64-bit x86 page table walk for virtual address (0ae, 0b1, 00c, 0c2, 035) and PTCs in AMD's processes.

entries to reduce long-latency page table walks. As shown in Fig.1, taking the x86-64 processor as an example, it uses a four-level radix tree as its page table. When performing virtual-to-physical address translation, four levels of memory accesses are required to obtain the final physical address, which can cause significant performance degradation. To avoid long memory access latency, processor architects store L4, L3 and L2 page table entries (PTEs) in PTCs to reduce the overhead of TLB misses. Fig.1 (b) presents the PTC structure in AMD' processors. Unlike TLB which is index by virtual address, PTCs in AMD' processors is indexed by physical address. And it caches PTC entries from any upper-level of the radix tree. After a TLB miss, PTC first checks to determine whether the corresponding PTC entries are stored. If so, MMU can skip one to three memory accesses to get the final physical address. Fig.1 (b) shows the PTCs state after accessing virtual address *(0ae, 0b1, 00c, 0c2, 035)*. If the MMU subsequently translates the virtual address *(0ae, 0b1, 00c, 0c2, 046)*, a L2-PTC entry hit occurs. Thus, the page walker can skip two upper-level searches and only one memory access is required to get the translation.

Both approaches have shown to improve the performance of address translation to a certain extent. However, they are used exclusively either for TLBs (to increase TLB coverage) or PTCs (to reduce TLB miss handling latency), because TLBs and PTCs are implemented in separate structures in existing computer systems. In the next subsection, we will explain the limitations of structural separation between TLBs and PTCs to motivate our proposed Unified-TP design.

B. Motivation

The physical and logical separations of TLB and PTC structures renders its impossible to dynamically adjust their sizes according to the workloads. Specifically, workloads may have different memory access behaviors and different demands on the numbers of TLB and PTC entries. However, such numbers remain fixed in separately structured TLBs and

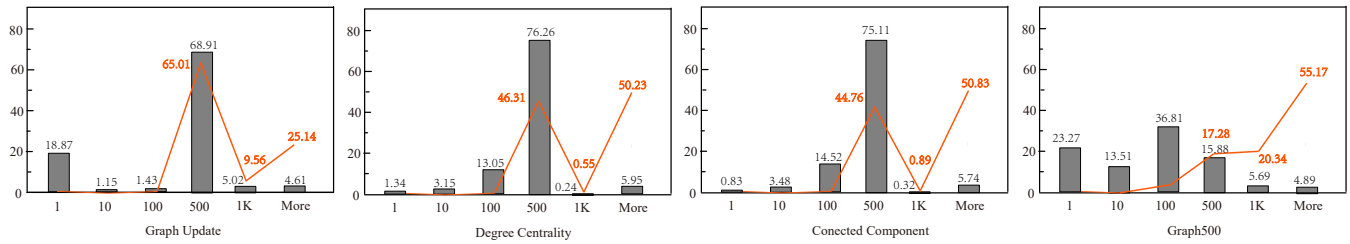

Fig. 2. Distribution of TLB misses among all pages for four graph analytics workload. The x-axis represents the ranges of TLB misses per page, i.e., $= 1; > 1\& <= 10; > 10\& <= 100; \ldots; > 1000$. The y-axis shows the percentage of pages that fall into a given range, indicated by the height of the bar, e.g., the bar on 500 in Graph Update indicates that 68.91% of pages are each missed less than or equal to 500 but more than 100 times. The orange line in each figure plots the percentage each range contributes to the total TLB misses.

PTCs even though either TLBs or PTCs may be empty or sparely populated, resulting in low cache space utilization and inefficient address translation. In this subsection, we will focus on two types of applications based on their memory access behaviors and analyze the limitations of separately structured TLBs and PTCs in face of these applications.

1) Memory accesses with long reuse distance: Many modern applications, such as graph analytics and in-memory key-value stores, can be characterized by memory pages with long reuse distance. Existing fixed-size TLBs cannot accommodate memory pages with long reuse distance, because when referring to these memory pages, their PTEs have been evicted from the TLBs due to the small capacity of TLBs, which leads to a high TLB miss rate and incurs heavy memory access overhead.

Fig. 2 shows the TLB page miss distributions among all pages for different graph analytics workloads. The x-axis represents the ranges of TLB misses per page, i.e., misses per page equal to 1, greater than 1 and less than or equal to 10, greater than 10 and less than or equal to 100, ..., greater than 1000. The y-axis shows the percentage of pages that fall into a given range, indicated by the height of the bar, e.g., the bar on 500 in Graph Update indicates that 68.91% of pages are each missed less than or equal to 500 but more than 100 times. The orange line in each figure plots the percentage each range contributes to the total TLB misses. As shown in Fig.2, about 4.61% memory pages contribute 25.14% of the total TLB misses in Graph Update and for other workloads, the percentage of the total TLB misses contributed by about 5% memory pages is up to about 50%.

The results of Fig.2 show that these frequently accessed memory pages exhibit long reuse distance behaviors, because when referring to these pages, their PTEs have been evicted from the TLBs, resulting in low TLB miss rates. Although existing methods can use PTCs to cache frequently used upper-level entries of the page table, at least one memory access is still required to get the translation record.

Therefore, due to insufficient TLB space and inflexibility of fixed-size structures, the structural separation of TLBs and PTCs in existing processor architectures causes a potential performance bottleneck for this type of applications. If some PTC entries can be deleted to shrink the PTC space and provide more space for TLB entries to accommodate the address translation of data with long reuse distance, it will greatly reduce memory access overhead and improve address translation performance.

2) Memory access with sequential pattern: The second type of applications is characterized by sequential access of memory pages. For example, sensor data is often organized along the time dimension to form a large amount of time series data for monitoring and analysis [18]. Similarly, many common applications, such as time series databases, video and senor applications, also have such memory access behavior.

When processing sequential memory accesses, there are almost no repeat requests, so the utilization of TLBs is very low. On the contrary, PTCs have a high hit rate, because most of the upper-level PTC entries of these memory pages are similar. In other words, the memory accesses have high spatial locality. For example, if the granularity of sequential reads is 4KB, most virtual addresses have the same L4, L3 and L2 PTC entries. Storing them in PTCs can reduce large number of memory accesses caused by TLB misses.

Fig.3 gives an example of 4KB granular sequential reads. Because there are few reused entries, the entries stored in the TLB cannot match subsequent requests, which results in a high TLB miss rate. PTCs store three upper-level entries of VA_c and VA_f. This is because we assume that sequential reads are conducted in two different regions that start with VA_c and VA_f. Therefore, most subsequent requests have the same L4, L3 and L2 indexes of these two addresses. In this case, because the hit rate of the PTC is high, PTC can function well, but the TLB is not fully utilized, thereby reducing the space utilization to a certain extent.

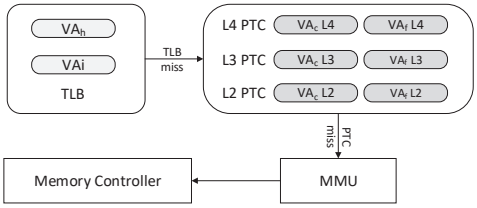

Fig. 3. An example of memory access with sequential pattern. The upper-level PTC entries of VA_c,VA_f are frequently accessed.

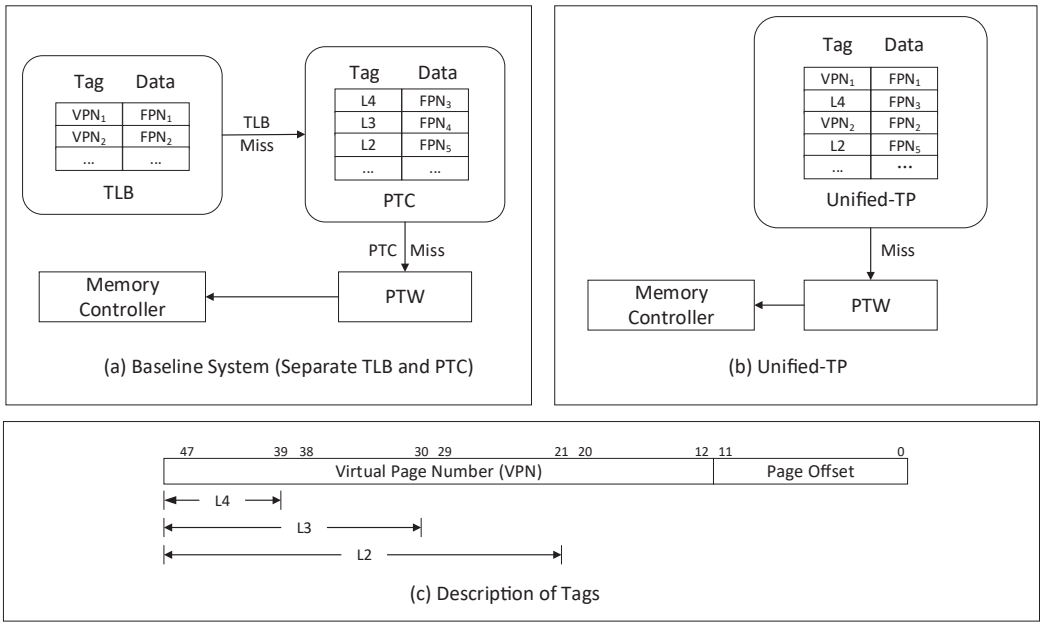

Fig. 4. Contrast between Unified-TP and baseline (conventional separately structured TLB and PTC).

The separate TLB and PTC structures do not work well in the second type of applications because TLB remains in its entirety despite of its inefficiencies. If we can delete useless TLB entries and expand PTC space to store more PTC entries, it can reduce a lot of memory access overhead.

III. Design of Unified-TP

The goal of this paper is to design an efficient address translation support for workloads with different memory access behaviors. After carefully analyzing the deficiency of separately structured TLBs and PTCs in Section II, we propose a unified TLB-PTC structure, called **Unified-TP**. In the following subsections, we are going to introduce the design and implementation of Unified-TP in detail.

A. Unified-TP Structure

To meet the requirements of dynamically adjusting the sizes of TLB and PTC structures according to the workloads, Unified-TP combines TLB and PTC into a single structure. In baseline system, the TLB and PTC are cache structures with a tag array and a data array. The TLB tag array stores virtual addresses while the PTC tag array stores physical addresses. Both the TLB data array and the PTC data array store the physical address corresponding to the virtual address translation. The TLB and PTC have similar sized tag arrays and data arrays which provide the opportunity to unify the conventional TLB and PTC structures.

Fig.4 illustrates the structure of Unified-TP and contrasts it to the baseline. In unified-TP, TLB and PTC entries are stored together. For TLB entries, they are similar to those in the baseline system. Specifically, they use *Virtual Page Number (VPN)* as the tag field and *Physical Page Number (PPN)* and protect bits (not shown in Fig.4) as the data field. For PTC entries, unlike PTCs in AMD where entries are indexed by physical addresses, they adopt the same indexing scheme (indexed by virtual address) as TLB entries to make it easier to design and implement in hardware. As shown in Fig.4 (c), the tag of a PTC entry, L2, L3, L4, is a portion of VPN. Specifically, the leftmost 9, 18 and 27 bits of VPN are used as the tag of L4, L3 and L2 PTC entries respectively, and PPN of the corresponding page tables and protect bits are stored in the data field. For simplicity, we do not show the data field when discussing about Unified-TP in the remainder of this section. Furthermore, Unified-TP mixes PTC entries from any upper-level of the page table instead of using distinct caches for each level of the tree structure (Intel's *Paging Structure Caches* [19]). This is because distinct caches maintain fixed number of entries for each level even though some levels may be empty or sparsely populated, resulting in low cache space utilization.

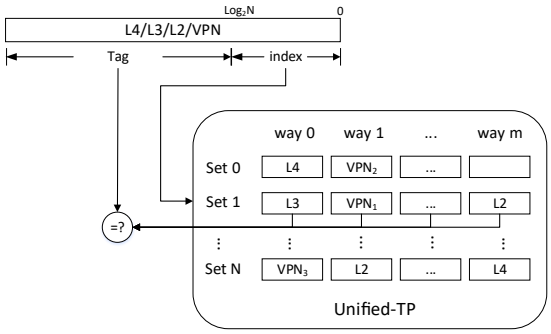

Fig. 5. Illustration of set-associative of Unified-TP.

Like the baseline system, Unified-TP can also store TLB

and PTC entries in a set-associative way. Unified-TP does not distinguish between TLB and PTC entries during an insert operation. When Unified-TP stores a TLB entry, the set selection is similar to what the baseline system does. To simplify the implementation of hardware, we treat PTC entries in the same way as we do TLB entries. Specifically, for a N-set Unified-TP, the rightmost $\log_2 N$ bits of L4/L3/L2 are used to locate the corresponding set and the rest of bits are compared with the tags stored in the ways to determine whether the entry is stored in the Unified-TP, as shown in Fig.5. The uniform scheme of set selection for TLB and PTC entries makes Unified-TP a hardware-friendly structure. This is because set selections and insertion operations are the same as conventional TLBs in the baseline system, making the Unified-TP implementation straightforward with minimum modifications to the hardware design.

We now discuss the lookup and fill operations of Unified-TP. When CPU sends a memory access request, it consults the Unified-TP to determine if the corresponding TLB or PTC entries are stored. If there is a TLB entry hit, the translation is returned to the CPU directly. Otherwise, PTW needs to walk the page table to retrieve the missing translation no matter the corresponding PTC entries hit or miss. Then the missing translation and corresponding PTC entries are inserted into Unified-TP (for further use) based on LRU replacement algorithm.

B. Parallel Search in Unified-TP

Since structural separation of TLBs and PTCs in the baseline system enables the respective circuitries of TLBs and PTCs to each separately search entries and maintain replacement policies, searching TLB and PTC entries can be executed in parallel. In contrast, Unified-TP stores all entries into a single structure, which may limit its ability to search entries in parallel. Specifically, when receiving a memory access request, the corresponding TLB entry and L2 to L4 PTC entries are checked in a sequential order if there is only one circuitry of Unified-TP. In this subsection, we propose a parallel search scheme to achieve the same effect of parallelism as the baseline system.

The basic idea of parallel search scheme is to provide more circuitries. In this way, when receiving a memory request, each circuitry check one of corresponding entry among TLB and L2 to L4 entries to accelerate the process of checking entries. However, when there are multiple entries hit, updating the replacement states of each hit entry should be carefully designed. Otherwise, data consistency issues may occur when multiple circuitries modify the replacement information simultaneously. To have a better understanding of parallel search scheme, we divide a search operation into two parts, (1) checking if the corresponding entry is stored in the Unified-TP and (2) modifying the replacement information according to the check results. Then we discuss how the parallel search scheme works in each part in the following.

First, for checking entries, Unified-TP finds the set number based on the index field and compares each way with the tag.

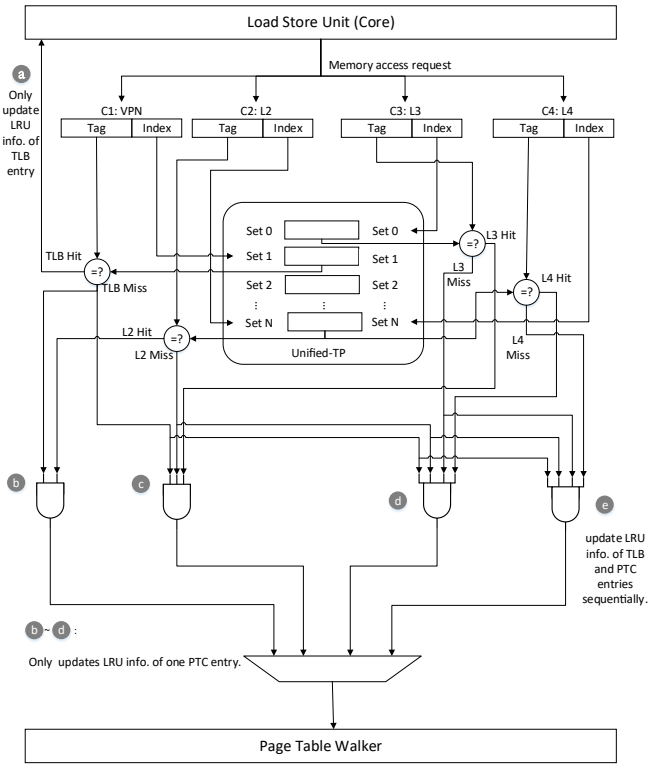

Fig. 6. Parallel search of Unified-TP.

In this part of the search operation, only read operations are involved. We provide the same number of circuitries (e.g., 4 corresponding to one TLB and three PTCs) as in the baseline system to check entries in parallel. In other words, for each entry in Unified-TP, it can be read by all the circuitries at the same time. For example, Fig.6 presents a Unified-TP with 4 circuitries (C1 to C4). When receiving a memory access request, C1 is used to check whether the corresponding TLB entry is stored in Unified-TP and C2 through C4 check the corresponding L2 through L4 PTC entries respectively. Therefore, all the circuitries can work in parallel in terms of checking entries.

Second, replacement information should be modified, which consists of updating replacement states when Unified-TP hits and inserting new entries when Unified-TP misses, according to the result of checking entries. Specifically, if the TLB entry hits, only update the replacement information of TLB entry. If the TLB entry miss and PTC entry hits, besides the replacement information of PTC entry, PTW needs to walk the page table from the corresponding level to get the translation and insert upper-level PTC and TLB entries into Unified-TP. Otherwise, PTW walks the page table from the root as baseline system does.

Not that for Unified-TP hits, multiple circuitries hits may incur data consistency issues, because multiple writes are executed to update replacement states simultaneously. To avoid this problem, among the circuitries whose check results are

hits, we choose the circuitry that checks the longest tag to update the replacement state of the corresponding entry. This is because the longer the tag is, the fewer memory accesses are required. For Unified-TP misses, PTW walks the page tables from the root to get the translation and inserts TLB and PTC entries into Unified-TP. Since page tables are accessed level by level, insert operations on TLB and PTC entries are executed sequentially, which does not incur data consistency issues.

Fig.6 illustrates how the replacement information is modified in detail. Cases ⓐ through ⓓ represent that the hit of the longest tag is from TLB to L4-PTC entries respectively. For example, ⓐ represents the case that a TLB entry hits. Even though PTC entries checked by C2 through C4 may also hit, only C1 is responsible for updating the replacement information of the corresponding TLB entry. This is because the TLB entry has the longest tag among hit entries. ⓑ represents the case that a TLB entry misses and a L2-PTC entry hits. In this case, only C2 updates the replacement information of the corresponding L2-PTC entry. Similarly, C3 and C4 update the information in cases ⓒ and ⓓ respectively. Case ⓔ represents that there is a Unified-TP miss. In this case, Unified-TP utilizes our modified replacement policy to find victims and inserts L4-PTC to TLB entries sequentially.

Based on the above analysis, Unified-TP can achieve parallel search by providing multiple circuitries and simplifying the procedure of modifying replacement information. For the baseline system, there are two differences from Unified-TP. First, TLBs and PTCs of the former have the respective circuitries to search entries and to maintain the information of replacement policy separately. Second, when there are multiple hits of TLB and PTC entries in the former (like cases ⓐ through ⓓ of the latter), all the hit structures need to update replacement information which results in more write operations than Unified-TP. Although Unified-TP reduce the number of updating the replacement information, it works efficiently with LRU algorithm to dynamically adjust the TLB and PTC sizes according to the workloads. Specifically, for the first type of applications in motivation, replacement information of TLB entries are updated frequently which stores at MRU position for further use. For the second type of applications, even though we do not insert the upper-level entries at the MRU position at first, these entries will be promoted to the MRU position when accessed again. Finally, useless TLB entries are stored in the LRU position and are more likely to be evicted in Unified-TP.

IV. Evaluation

A. Methodology

We simulated our work on a trace-based simulator that models the TLB and PTC structures. We used the Pin binary instrumentation tool [21] to generate the memory access trace as in the prior work [4]. The configuration of the hardware components is shown in the Table I.

Besides Unified-TP, we also simulate separate TLB and PTC structure as the baseline structure. The baseline structure uses distinct PTCs for each level of the page table, like Intel's *Paging Structure Cache (PSC)* [19]. Therefore, one TLB and three PTCs are included in this structure, which we refer to as *PSC-Baseline (PSC_B)*.

TABLE I
System configuration

Component	Configuratioin
TLB	64 entries, 4-way, 4-cycle access latency
PTC	32 entries, fully-associative, 10-cycle access latency
Unified-TP	96 entries, 4-way, 4-cycle access latency
PTW	50 cycle page table walk latency

To provide a quantitative evaluation, we use a wide range of benchmarks. First, we pick two kernels, which are known to be memory-intensive workloads and two applications from the PARSEC 3.0 suite [22]. In addition, we evaluate graph500, one of the most popular graph benchmarks. Furthermore, we also evaluate a big data benchmark, the GraphBIG suite [23]. GraphBIG benchmarks are based on IBM System G framework, a widely-used comprehensive industrial graph computing toolkit. GraphBIG gives us the diversity of real-world graph benchmarks that cover a variety of graph computation problems. The suite consists of three main categories: graph update, graph analysis, and social media analysis. We evaluate benchmarks from all three categories in order to span a diverse selection of modern graph workloads. A brief description of each application is specified in Table II.

B. Impact on TLB Misses

Fig.7 shows the TLB miss rate, normalized to that of the baseline structures *PSC_B*. For the PARSEC 3.0 suite including two memory intensive workloads and two applications workloads, the TLB miss rate reduction ranges from 42.95% to 60.85%. These workloads benefit from Unified-TP due to its ability to store more TLB entries than *PSC_B*. This is because Unified-TP combines TLBs and PTCs to offer an elastically larger capacity by adapting to workload characteristics with its improved replacement algorithm. Therefore, more TLB entries

TABLE II
The different workloads used in the study

Workload	Description
Canneal	Kernel from Parsec 3.0
Streamcluster	Kernel from Parsec 3.0
Blackscholes	Applications from Parsec 3.0
Swaptions	Applications from Parsec 3.0
Graph500	BFS Kernel
Graph Update (GUp)	Delete and Update Vertices
Degree Centrality (DCentr)	Social Media Analysis
Connected Component (CComp)	Graph Analysis and Traversal
Page Rank	Monte Calo Mini-App

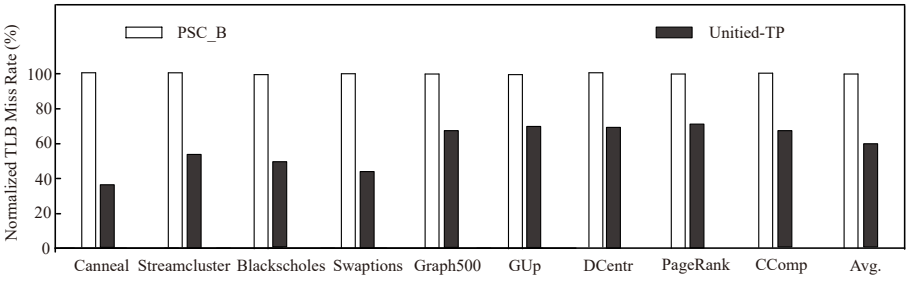

Fig. 7. TLB miss rate, normalized to that of *PSC_B* for all benchmarks.

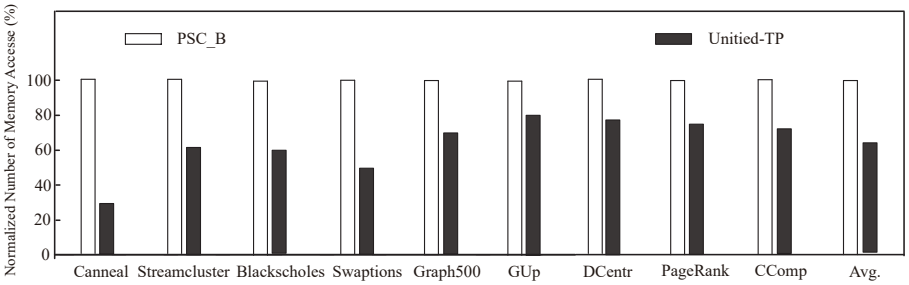

Fig. 8. Number of memory accesses to PTC entries, normalized to that of *PSC_B* for all benchmarks.

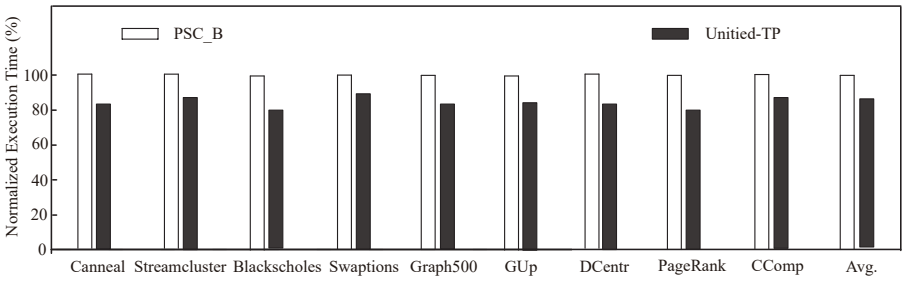

Fig. 9. Execution time, normalized to that of *PSC_B* for all benchmarks.

can be stored in Unified-TP, improving TLB hit rate. That is, when a TLB miss occurs, a new TLB entry can be inserted in Unified-TP owing to its elasticity. An old TLB entry in *PSC_B* must be evicted every time to make room for the new entry when TLB is full, possibly leading to further TLB misses. For graph workloads, the TLB miss rate reduction ranges from 21.41% to 25.85%. The TLB miss rate of these workloads is 3 times that of workloads of the PARSEC 3.0 suite because of the weak locality. Thus, Unified-TP's ability to store TLB entries is also weaker for memory pages with long reuse distance, leading to lower TLB miss reductions for these workloads. Unified-TP reduces TLB misses rate by 35.69% on average for all benchmarks.

C. Impact on Memory Accesses

Fig.8 shows the number of memory accesses to PTC entries, normalized to that of the baseline structure *PSC_B*. For workloads in PARSEC 3.0, Unified-TP reduces the number of memory accesses from 38.41% to 61.59%. For graph workloads, the reduction of memory accesses ranges from 18.79% to 28.94%. Unified-TP outperforms *PSC_B* because the latter has higher TLB miss rate than the former, which leads to more PTCs lookup operations and thus larger number of memory accesses in *PWC_B* than in Unified-TP. Moreover, PTC hits in different levels incur different numbers of memory accesses. We observe that Unified-TP has higher L1-PTC entry hit rate than *PSC_B* on most of the workloads, resulting in fewer memory accesses. In other words, most of the PTCs lookups only need one memory access in Unified-TP. In summary, Unified-TP can reduce the number of memory accesses by an average of 65.39% for all the benchmarks.

D. Impact on performance overhead

Fig. 9 shows the execution time based on the latencies in Table I of different workloads, normalized to that of the baseline structure *PSC_B*. Unified-TP achieves performance improvements ranging from 7.49% to 14.60%. The performance overhead of Unified-TP and *PSC_B* can be divided into two parts based on the TLB hits or misses. Since Unified-TP and *PSC_B* have the same overhead of per TLB hit, the performance mainly depends on the penalty of a TLB miss. For graph workloads, Unified-TP's performance improvements over *PSC_B* range from 8.06% to 14.61% because of the fewer TLB misses in the former than the latter. For workloads in the PARSEC 3.0 suite, even though the TLB miss reduction is higher than that of graph workloads, the performance improvement ranges from 7.46% to 13.71% due to the lower TLB miss rate (about one third) than the latter. In summary, Unified-TP achieves performance improvements by 11.12% on average for all benchmarks.

V. Conclusion

The emergence of memory-intensive workloads with large memory footprints increases the demand on MMU to deliver better performance in computer systems. In this paper, we investigate the impact of separately structured TLBs and PTCs in existing methods and find that they are inflexible for different requirements of workloads, thereby limiting the performance of address translation. We propose Unified-TP, which is a novel structure that can mitigate the cost of large number of memory accesses. Unified-TP unifies TLBs and PTCs to improve cache efficiency and to adapt to different workloads. Besides, we provide a parallel search scheme to improve the performance of checking entries when a memory access request is sent. We evaluate Unified-TP and separately structured TLB and PTC and our results show that Unified-TP reduces TLB misses and improve the performance by 35.69% and 11.12% on average for all the workloads, respectively.

ACKNOWLEDGMENTS

This work was supported by grants from Natural Science Foundation of Chongqing No. cstc2020jcyj-msxmX0897, the Fundamental Research Funds for the Central Universities No. 2020CDJ-LHZZ-050, Open Project Program of Wuhan National Laboratory for Optoelectronics NO. 2019WNLOKF009, National Natural Science Foundation of China NO.61672115, and 61802038, China Postdoctoral Science Foundation NO.2017M620412, and Chongqing Postdoctoral Special Science Foundation NO. XmT2018003.

References

[1] H. Elnawawy, R. B. R. Chowdhury, A. Awad, and G. T. Byrd, "Diligent tlbs: a mechanism for exploiting heterogeneity in tlb miss behavior," in *Proceedings of the ACM International Conference on Supercomputing*. ACM, 2019, pp. 195–205.

[2] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Translation ranger: operating system support for contiguity-aware tlbs," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 2019, pp. 698–710.

[3] M. Clark, "A new× 86 core architecture for the next generation of computing." in *Hot Chips Symposium*, 2016, pp. 1–19.

[4] C. H. Park, T. Heo, J. Jeong, and J. Huh, "Hybrid tlb coalescing: Improving tlb translation coverage under diverse fragmented memory allocations," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 444–456.

[5] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "Colt: Coalesced large-reach tlbs," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 258–269.

[6] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Supporting superpages in non-contiguous physical memory," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 223–234.

[7] M.-M. Papadopoulou, X. Tong, A. Seznec, and A. Moshovos, "Prediction-based superpage-friendly tlb designs," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 210–222.

[8] G. Cox and A. Bhattacharjee, "Efficient address translation for architectures with multiple page sizes," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 435–448, 2017.

[9] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Ünsal, "Redundant memory mappings for fast access to large memories," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 66–78.

[10] T. W. Barr, A. L. Cox, and S. Rixner, "Translation caching: Skip, don't walk (the page table)," in *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, 2010.

[11] ——, "Spectlb: a mechanism for speculative address translation," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, pp. 307–318.

[12] A. Bhattacharjee, "Large-reach memory management unit caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 383–394.

[13] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, "Accelerating two-dimensional page walks for virtualized systems," in *ACM SIGOPS Operating Systems Review*, vol. 42, no. 2. ACM, 2008, pp. 26–35.

[14] V. Karakostas, O. S. Unsal, M. Nemirovsky, A. Cristal, and M. Swift, "Performance analysis of the memory management unit under scale-out workloads," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014.

[15] D. Lustig, A. Bhattacharjee, and M. Martonosi, "Tlb improvements for chip multiprocessors:inter-core cooperative prefetchers and shared last-level tlbs," vol. 10, no. 1, pp. 1–38, 2013.

[16] A. Esteve, A. Ros, M. E. Gomez, A. Robles, and J. Duato, "Efficient tlb-based detection of private pages in chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 1–1, 2015.

[17] S. Shin, G. Cox, M. Oskin, G. H. Loh, S. Yan, A. Bhattacharjee, and A. Basu, "Scheduling page table walks for irregular gpu applications," 2018.

[18] J. Qiao, X. Huang, J. Wang, and R. K. Wong, "Dual-pisa: An index for aggregation operations on time series data," *Information Systems*, vol. 87, p. 101427, 2020.

[19] I. Corporation, "Tlbs, paging-structure caches and their invalidation," 2008.

[20] N. Young, "On-line caching as cache size varies." in *In SODA91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, 1991, pp. 241–250.

[21] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190–200, 2005.

[22] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[23] I. Tanase, Y. Xia, L. Nai, Y. Liu, W. Tan, J. Crawford, and C. Y. Lin, "A highly efficient runtime and graph library for large scale graph analytics," in *Workshop on Graph Data Management Experiences and Systems*, 2014.