GATLB: A Granularity-Aware TLB to Support Multi-Granularity Pages in Hybrid Memory System

Yujuan Tan[1,2], Yujie Xie[1], Zhulin Ma[1,*], Zhichao Yan[3], Zhichao Zhang[1], Duo Liu[1] and Xianzhang Chen[1]
[1] College of Computer Science, Chongqing University, Chongqing, China
[2] Wuhan National Laboratory for Optoelectronics, Wuhan, China
[3] HewlettPackard Enterprise, San Jose, USA

***Abstract*—The parallel hybrid memory system that combines Non-volatile Memory (NVM) and DRAM can effectively expand the memory capacity. But it puts lots of pressure on TLB due to a limited TLB capacity. The superpage technology that manages pages with a large granularity (e.g., 2MB) is usually used to improve the TLB performance. However, its coarse-grained granularity conflicts with the fine-grained page migration in the hybrid memory system, resulting in serious invalid migration and page fragmentation problems. To solve these problems, we propose to maintain the coexistence of multi-granularity pages, and design a smart TLB called $GATLB$ to support multi-granularity page management, coalesce consecutive pages and adapt to various changes in page size. Compared with the existing TLB technologies, GATLB can not only perceive page granularity to effectively expand the TLB coverage and reduce miss rate, but also provide faster address translation with a much lower overhead. Our experimental evaluations show that GATLB can expand the TLB coverage by 7.09x, reduce the TLB miss rate by 91.1%, and shorten the address translation cycle by 49.41%.**

***Index Terms*—TLB, multi-granularity pages, parallel hybrid memory**

I. Background and Motivation

Nowadays, parallel hybrid memory architecture composed of DRAM and non-volatile memory (NVM) can greatly expand the available memory to meet the ever-increasing memory requirements of modern applications. In this architecture, both DRAM and NVM are treated as main memory equally, aiming at building large-capacity hybrid memories. To avoid NVM's weakness, hot pages are usually put on the DRAM and cold pages are placed in NVM through page migration technology.

The hybrid memory architecture brings large capacity, but it also puts great pressure on address translation of the translation lookaside buffer (TLB). The rapid expand in memory capacity exacerbates the problem of TLB misses and triggers large number of long-latency page table walks, which leads to memory access performance degradation. In order to reduce the TLB miss rate, the most direct way is to increase the capacity of the TLB. However, this method is impractical because increasing the TLB size not only results in higher hardware overhead and longer look-up latency, but a small increase in TLB capacity cannot effectively reduce the TLB miss rate.

Another commonly used effective method is to expand the TLB coverage (the mapping range of each TLB entry) by leveraging superpages [1] - [3] in main memory. Compared to the increase in TLB capacity, using superpages can expand the TLB coverage by several times while slightly increasing the hardware overhead. For example, if we use 2MB superpages instead of 4KB normal pages in memory, one entry that was previously mapped to one normal page can now be mapped to 512 normal pages, which will greatly expand the TLB coverage, thereby reducing the TLB miss rate. Many studies have shown that using superpages can indeed effectively reduce the TLB miss rate [4] - [6], leading to 49.6% performance improvement [7]. Although superpages can effectively extend TLB coverage, directly replacing normal pages with superpages in the hybrid memory system will cause new problems.

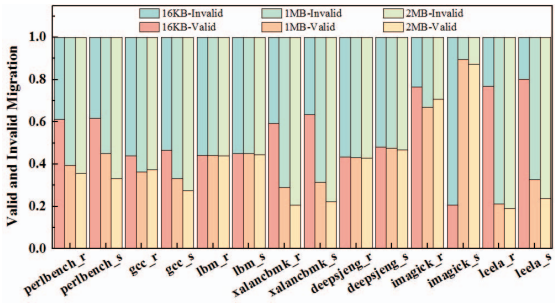
*Corresponding author: Zhulin Ma, College of Computer Science, Chongqing University, Chongqing, China. E-mail: mazhulin1993@gmail.com.

A. Superpage Issues in Hybrid Memory

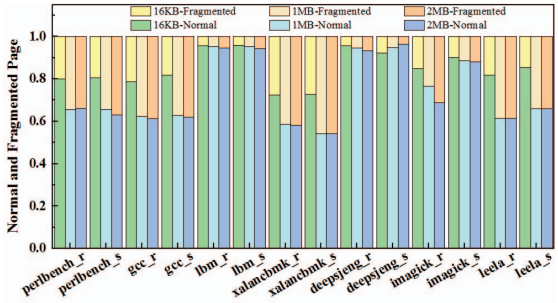
As mentioned above, page migration is used in the hybrid memory system to gain performance improvement. However, when superpages are used in this architecture, the invalid page migration and page fragmentation problems would arise.

Invalid page migration: Existing research [8] shows that the memory references of most applications are mainly concentrated on a few 4KB hot pages, while most of the remaining pages are cold pages. Moreover, these hot pages and cold pages are mixed together and distributed on the superpages. Therefore, migrating an entire superpage will bring a large number of small cold pages to DRAM, thereby wasting transmission bandwidth and valuable DRAM space. Besides, in order to make room for the migrated pages, DRAM will evict pages into NVM, and these pages may contain small hot pages that are hotter than those small cold ones migrated to DRAM, which violates the rules of putting hot data in DRAM and storing cold data in NVM. In addition, if these small hot pages migrated to NVM are accessed intensively in the future, they would be returned to DRAM again, resulting in frequent page swaps. These additional page swaps will cost a heavy system overhead [9], which far exceeds the benefit of page migrations. In this paper, we refer to this type of migration as ***invalid page migrations***. In our preliminary studies, we found that the proportion of invalid page migration is very high. As shown in Fig. 1(a), for the SPEC 2017 workload [10], the average percentage of invalid migrations caused by migrating the entire 2MB superpage is about 60.3%.

Page fragmentation: The problem of invalid migration can be solved well if we only migrate the small hot pages in a superpage, but such fine-grained page migration will destroy the continuity and integrity of superpages. Because in the parallel



(a) The invalid migrations caused by migrating superpages.



(b) The page fragmentation caused by migrating small pages.

Fig. 1: The statistic evidence of invalid migration and page fragmentation.

hybrid memory architecture, pages cannot reside in both NVM and DRAM. The origin space of the small hot pages in NVM will be reclaimed when they are migrated to DRAM. Only after all the small pages in the superpage have been reclaimed can the space be allocated to other pages. Otherwise, the empty ones scattered in the superpage caused by migration will neither be used nor allocated to other pages, resulting in ***page fragmentation*** problems. In the worst case, each superpage is a fragmented page and the available NVM space will soon be used up. Finally, it needs to swap superpages to disk to make room for new pages, resulting in unbearable overhead and performance degradation. As shown in Fig. 1(b), the proportion of fragmented pages in NVM caused by migrating small hot pages in SPEC 2017 workloads is about 26.61% on average and up to 45.69%.

The above analysis and statistical evidence clearly show that due to the conflict between fine-grained migration and coarse-grained superpage management, it is not appropriate to directly use superpages in a hybrid memory system. In addition, since different applications always require different sizes of superpages [12] and most applications use a high percentage of small pages, it is not suitable to use fixed-size superpages for all applications [11]. So in this paper, we propose the coexistence of multi-granularity pages to alleviate the incompatibility between coarse-grained superpages and fine-grained page migration in a hybrid memory system. However, when using multi-granularity pages, TLB needs to perceive different page sizes and obtain the correct physical address for each page, which brings great difficulties and challenges.

B. Challenges of TLB Support for Multi-granularity Pages

Using multi-granularity pages instead of default 4KB or 2MB pages will cause uncertainty in page size. However, in the address translation process, only by knowing the page size in advance can the TLB obtain the correct physical address. Therefore, the uncertainty of page size brings the following challenges to TLB management.

Challenge 1: ***which bits in the virtual address should be selected as the index?*** When performing an address look-up operation in a set-associative TLB, the virtual address is commonly divided into three parts, including Tag, Index and Page Offset. Index is used to locate the set in the TLB, Tag and Page Offset help to find the correct entry in the specified set and obtain the full physical address respectively. However, due to the use of multi-granularity pages, the Page Offsets corresponding to different page sizes are different [13], so the corresponding Index values are also different. Taking an 8-set 4-way set-associative TLB as an example, if the page size is 4KB, the Index value corresponds to the bits 12-14 of the virtual address. While if the page size is 2MB, the Index value corresponds the bits 21-23 of the virtual address. Therefore, when using multi-granularity pages, the uncertainty in page size makes it difficult to determine the position of the Index and it is impossible to obtain the correct physical address, resulting in invalid TLB work.

The fully-associative TLB that checks all entries for the requested address can be easily extended to support multi-granularity pages [14]. However, considering the translation speed, the fully-associative TLB can only support a limited number of entries. When the number of entries becomes large, each translation will take a long time, resulting in performance degradation. Therefore, we focus on designing a novel set-associative TLB to support multi-granularity page.

Challenge 2: ***which management granularity should we choose to use in the TLB?*** The uncertain problem of Index can be solved by setting a uniform page management granularity in the TLB. However, this will also fall into a dilemma. Specifically, the optimal page granularity varies from application to application [15] [12]. Even the same application has different page granularities at different stages [15] [11]. If the page granularity is small, then large pages need to store multiple TLB entries in different sets, resulting in very low TLB utilization and address coverage, which conflicts with the idea of using superpages to extend TLB coverage. On the contrary, if a large page size is selected as the TLB management granularity, all small pages in a large page will be mapped to the same TLB set with the same Tag and cannot be distinguished. Therefore, the correct physical page address of each small page cannot be obtained through normal address transformation.

The direct solution is to use split TLBs [8] [16] to provide a dedicated TLB for each page granularity. However, this method is only suitable for applications with few page granularities. When the number of page granularities is large, it is difficult to maintain a dedicated TLB for each page granularity in the limited TLB space. Besides, different applications have different requirements for page granularities [12]. For some applications, maintaining too many TLBs will cause some TLBs to be idle

[17] [18], resulting in low TLB space utilization.

Other existing solutions still choose normal 4KB pages as the TLB granularity, while coalescing consecutive pages near certain selected pages [15] [12] or adding dedicated TLBs to store large pages [19]. However, these methods also have limitations. First, it can only coalesce consecutive pages near certain selected pages, and cannot make full use of page continuity to effectively extend the TLB coverage. In addition, it may take two look-ups to find the correct TLB entry, one for small pages and another for large pages, which may take a long time. Second, adding a dedicated TLB with limited capacity can only store a limited number of large pages, and cannot fully support on-demand multi-granularity pages. Therefore, it is very challenging to design an efficient TLB to support the coexistence of multi-granularity pages.

Challenge 3: ***how to deal with page size changes?*** Due to page migration in the hybrid memory architecture, the page size often changes. For example, when a small page is migrated, a large page will be divided into multiple smaller pages, and multiple consecutive small pages can also form a larger page. Therefore, it's necessary to re-check the page continuity and update the corresponding TLB entries to ensure the correctness of address translations. The existing methods [15] [12] mainly rely on re-traversing the page table to check the page continuity and update the corresponding TLB entry, which is costly. So how to design a smart TLB that can adapt to frequent page size changes with low overhead is really challenging.

The above analysis and observations show that existing TLBs cannot address these challenges and manage multi-granularity pages efficiently. In this paper, we propose a new TLB called ***GATLB*** to support multi-granularity pages in the hybrid memory system, which can meet the following requirements. ***First***, GATLB can use clear Index bits to manage pages of different granularities. ***Second***, GATLB can perceive page granularity and adapt well to page granularity changes caused by page migration. ***Third***, GATLB can provide greater coverage expansion and support efficient address translation with low overhead. Our extensive experiments show that GATLB can expand the TLB coverage by 7.09x and reduce the TLB miss rate by up to 91.1% with 49.41% fewer address translation cycles.

II. Design of GATLB

The overall GATLB consist of three parts: *a unified TLB*, *entry coalescing logic* and *an address-aware allocation* method. The *unified TLB* can efficiently store TLB entries for multi-granularity pages in a unified way. *Entry coalescing logic* can perceiving and coalescing consecutive pages to expand TLB coverage. And *address-aware allocation* method can increase the page continuity in memory to facilitate the TLB entry coalescing. Below, we will introduce each part in detail.

A. Unified TLB

The unified TLB (U-TLB) is a novel set-associative TLB that can support multi-granularity page address translation. In order to manage pages of different granularity in a unified way, U-TLB uses the largest possible granularity as the management granularity according to the access characteristics of the applications, while considering both the TLB set conflict and TLB coverage. For example, for applications with large memory footprint and strong access continuity, it can use the granularity of 2MB in the TLB. In this case, assuming that an 8-way TLB with 1024 entries is used, U-TLB uses bits 21-27 of virtual address as the Index to locate the TLB set, and bits 28-63 are used as the Tag (marked as $High\_Tag$) for entry comparison, as shown in Fig. 2.

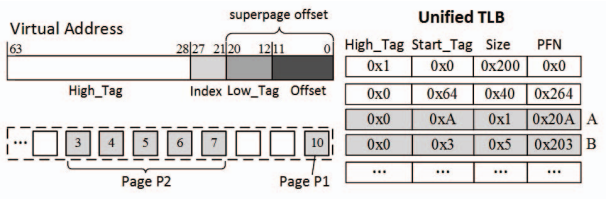


Fig. 2: An exmaple of the TLB entires in Unified TLB.

However, the above method brings a new problem, that is, small pages within one superpage will have the same Index and Tag and correspond to one TLB entry. In this case, it's impossible to distinguish which small page the TLB entry belongs to. To solve this problem, we introduce two extra fields, $Start\_Tag$ and $Size$, for each TLB entry to distinguish these small pages and identify the page granularity. For a certain granularity page **P**, $Start\_Tag$ records the bits 12-20 of the virtual address (marked as Low_Tag) for the start small page of page **P**. This is because the small pages in a superpage have the same $High\_Tag$, but their Low_Tag are different, which can be used to distinguish them. $Size$ holds the number of consecutive small pages contained in page **P**, which indirectly clarifies the page granularity of each TLB entry. In addition, for consistency, the PFN field in the TLB entry also stores the physical page frame number of the start page.

Fig. 2 shows an example, where **P1** and **P2** are two pages with different granularities in the same superpage and their information is recorded in TLB entry **A** and **B**. **P1** is a 4KB small page, thus $Start\_Tag$ of TLB entry **A** records the Low_Tag of page 10 and $Size$ stores the page granularity of 1. For **P2**, it contains five consecutive small pages (page 3 - page 7), so $Start\_Tag$ of the TLB entry **B** records the Low_Tag of the start page 3 and $Size$ stores the page granularity of 5.

Based on the above analysis, U-TLB only adds two fields to support multiple-granularity pages. Besides, it can identify the virtual and physical address range of the page through calculations based on $Start\_Tag$, PFN and $Size$ fields, which makes the TLB look-up efficient and simple. Moreover, according to the address range of the TLB entry, it can be recognized whether the pages corresponding to the two TLB entries are consecutive, which simplifies the coalescing of the TLB entries in Section II-B. Below, we will introduce the TLB look-up and update operation.

TLB look-up: Fig. 3 shows the look-up process of U-TLB. First, the Index bits are selected from the given virtual address **VA** to locate the TLB set. Then the $High\_Tag$ of **VA** will be compared with the $High\_Tag$ stored in all TLB entries in the set. If there is a match, the Low_Tag of **VA** will be determined

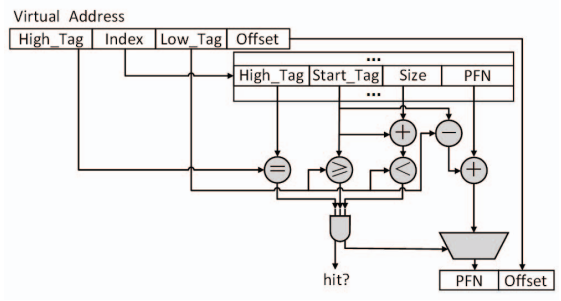


Fig. 3: The look-up process of U-TLB.

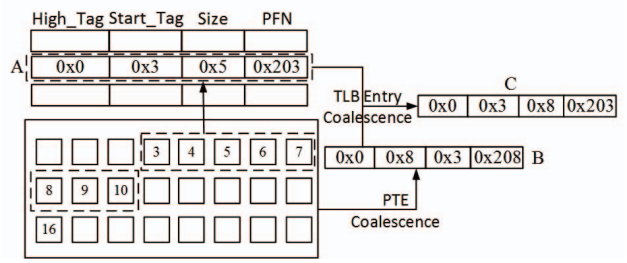


Fig. 4: An example of two-stage coalescing.

whether it is within the virtual address range of the TLB entry. Finally, we can get the required physical address if one TLB entry hits. Note that, the comparisons of the given **VA** with TLB entries in the same set will be performed in parallel to save time.

TLB update: Page migration will cause TLB update operations, and different migration methods can cause different update operations. For page migration from DRAM to NVM, the migrated page is cold, so there is no need to store its corresponding TLB entry. If it exists, the corresponding old TLB entry needs to be invalidated. For page migration from NVM to DRAM, the migrated page is considered to be hot, so it is necessary to invalidate the outdated TLB entry and insert a new TLB entry corresponding to the migrated page. It should be noted that during the migration process, large-granularity pages in NVM are likely to be divided into multiple smaller pages. In this case, it is still only necessary to insert TLB entries for the migrated pages, because pages that have not been migrated are considered cold, and these TLB entries should not be inserted to occupy the TLB space.

B. Two-stage Coalescing

Based on the unified TLB design, GATLB can support multi-granularity pages to address challenge 1 and challenge 2 described in section I-B. However, just using the unified TLB is not enough to effectively expand the TLB coverage. Therefore, in order to allow as many consecutive pages as possible to be represented by a single entry, GATLB is designed to use a ***two-stage coalescing*** method to detect and coalesce consecutive pages. The two-stage coalescing method includes two parts, page table entry (PTE) coalescing and TLB entry coalescing. Both are executed after returning the physical address and will not affect the memory access performance.

1) PTE Coalescing: PTE coalescing detects and coalesces consecutive PTEs to increase the TLB coverage. Specifically, when a TLB miss occurs, the page table is accessed, and multiple adjacent PTEs are usually prefetched into the CPU cache based on spatial locality. In this process, GATLB will check the continuity of these PTEs, coalesce consecutive PTEs whose PFN are also consecutive, and then insert a large-page granularity TLB entry to increase the TLB coverage.

2) TLB entry Coalescing: TLB entries with consecutive addresses will be mapped to the same set and occupy multiple ways, wasting TLB space. Therefore, when a new TLB entry is inserted, GATLB will detect and coalesce TLB entries with consecutive addresses. Here, GATLB uses the $Start\_Tag$, PFN and $Size$ fields stored in the TLB entries to check whether the virtual and physical addresses covered by the two TLB entries are consecutive, which can be performed with much low overhead.

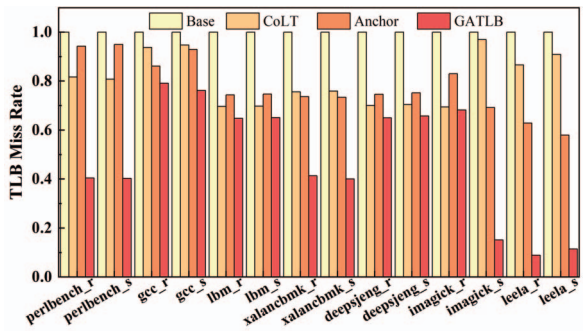
Taking the pages in Fig. 4 as an example, the TLB entry **A** currently stored in unified TLB contains five consecutive pages from page 3 to page 7. When accessing page 8, PTE coalescing will detect that pages 8 to 10 are consecutive, and then coalesce the three PTEs into a TLB entry **B** with $Start\_Tag$ being 0x8. In addition, since all these pages are in the same superpage and have the same $High\_Tag$ as entry **A**, entry **B** will be mapped to the same TLB set as entry **A**. In the TLB entry coalescing stage, entry ***A*** and entry **B** are finally coalesced into entry **C**. The old entry **A** will be replaced by the new entry **C** with a larger coverage.

C. Address-aware Allocation

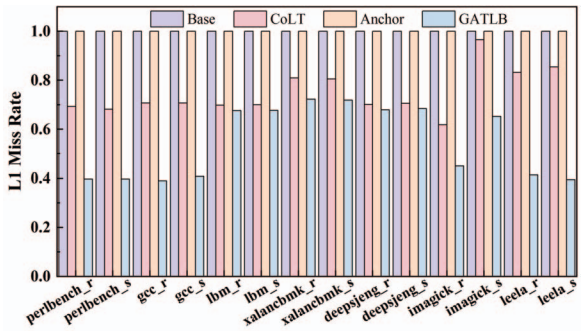
Based on the above design, GATLB can support the management of multi-granularity pages in the TLB and the coalescing of consecutive TLB entries to improve TLB coverage. However, the effectiveness of this design is constrained by the limited consecutive pages in memory. Therefore, in order to further improve the possibility of coalescing, we propose ***address-aware allocation*** to increase the proportion of consecutive pages in memory.

The basic principle of address-aware allocation is to allocate contiguous physical space for pages with consecutive virtual addresses. So when a new page is allocated, it is necessary to refer to the physical address of the allocated pages whose virtual address are consecutive with that of the new page. Fortunately, we can obtain this information by reading PTEs in the cache to guide page allocation. For example, the page with the virtual page number A is now in the physical page frame X. If the page frame X+1 is free when the new page A+1 is allocated, we can first allocate the page frame X+1 to the new page.

In NVM, considering its capacity is much larger than DRAM, we reserve NVM space for consecutive pages to further strengthen the page continuity. Specifically, we reserve 2MB superpage frame for consecutive 512 small pages within the same 2MB superpage and record the information of reserved physical superpage frame in the page table. When allocating a new page, we can easily use the information stored in the page table to calculate the final physical address reserved for



(a) The overall TLB miss rate



(b) The L1 TLB miss rate

Fig. 5: TLB Miss Rate.

the new pages. Then if the above ideal page frame allocated for new page is not available, we will re-allocate free pages according to the conventional allocation method.

III. Evaluation

A. Methodology

We simulate GATLB in the full system mode of gem5 [22] and use UIMigrate [23] as the page migration method. We set DRAM size to between 10% to 50% of the workload size, and NVM is large enough to accommodate the entire workload. Due to space limitation, here we only show the experimental results when DRAM size is 30% of the workload size.

We choose 14 classic workloads from SPEC CPU 2017 [10] and compare GATLB with three existing TLB schemes, including Base TLB and two state-of-the-art methods CoLT [19] and Anchor [15]. In GATLB, the management granularity of TLB is set to 64KB according to the size of the workloads. Base TLB uses L1 and L2 TLBs with 4KB management granularity. CoLT adds a fully-associative super TLB on the basis of normal TLBs to support pages of different granularities and there is also entry coalescing in the super TLB. Anchor selects a subset of PTEs as anchor entries, and records the information of page continuity in the anchor entries. The detailed configuration in our experiment is shown in the Table I.

B. TLB Coverage

Table II compares the TLB coverage between GATLB and the other three existing TLBs, and normalizes the results to those of Base TLB. GATLB has achieved the highest expansion of TLB coverage. Compared with Base TLB, GATLB expands the TLB coverage by 7.09x, which is 3.15 times and 4.40 times of CoT and Anchor respectively. This is because GATLB not only actively coalesces TLB entries and PTEs, but also considers making pages consecutive during allocation to increase the probability of page coalescing, which greatly expands the TLB coverage. For CoLT, it just adds an additional super TLB to expand the coverage of the TLB. The small size of the super TLB greatly limits the expansion of its coverage. While for Anchor, it relies on coalescing pages that are consecutive to the anchor entries. However, due to the limited number of the anchor entries, page continuity cannot be fully utilized to expand the TLB coverage. Therefore, because of the design constraints, the TLB coverage of CoLT and Anchor is much lower than that of GATLB.

TABLE I: TLB confguration

Schemes	**TLB Configuration**
Processor	10 cores, 2.2GHz, x86
Super TLB	8 entries, fully-associative
L1 TLB	64 entries, 4-way set-associative
L2 TLB	1024 entries, 8-way set-associative
Latency	7 cycle regular L2 hit latency [20] 8 cycle Anchor/GATLB hit latency [15] 50 cycle page table walk [21]

TABLE II: TLB coverage

	Base	CoLT	Anchor	GATLB
perlbench_r	1	1.16	1.56	4.60
perlbench_s	1	1.18	1.56	4.64
gcc_r	1	1.11	1.46	4.75
gcc_s	1	1.11	1.46	4.92
lbm_r	1	13.89	1.80	14.14
lbm_s	1	9.69	1.81	13.00
xalancbmk_r	1	1.10	1.41	3.09
xalancbmk_s	1	1.10	1.42	3.19
deepsjeng_r	1	12.62	1.80	14.36
deepsjeng_s	1	10.63	1.80	13.37
imagick_r	1	1.28	1.21	6.89
imagick_s	1	1.05	1.52	7.13
leela_r	1	1.07	1.37	2.80
leela_s	1	1.06	1.33	2.38

C. TLB Miss Rate

The TLB miss rate is a key indicator of TLB performance. Here we show two miss rates, the overall TLB miss rate and the L1 TLB miss rate. In order to highlight the improvements, all results are normalized to the result of Base TLB.

Fig. 5(a) shows the overall TLB miss rate of the four TLBs. Compared with Base TLB, GATLB can reduce the TLB miss rate by an average of 51.31% and up to 91.1%, while CoLT and Anchor only slightly reduce the TLB miss rate by 19.56% and 22.35% on average. This is because in CoLT and Anchor, they only perform very limited page coalescing to expand address coverage (see Table II), so the miss rate is much less reduced. While in GATLB, it uses two-stage coalescing technology and address-aware allocation method to make full use of page continuity and coalesce consecutive pages, thereby greatly expanding the TLB coverage and reducing the miss rate.

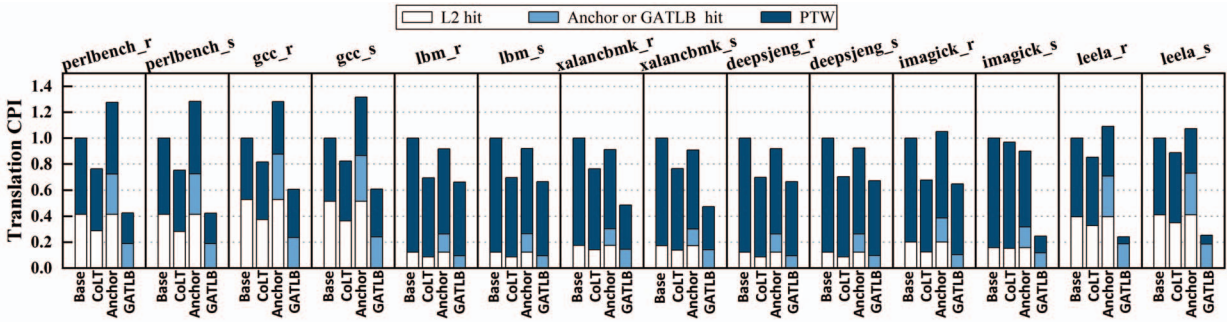


Fig. 6: The breakdown of translation CPI.

Fig. 5(b) shows the L1 TLB miss rate of the four TLBs. As seen from the results, GATLB has the lowest L1 TLB miss rate. In GATLB, entries in the L1 TLB are not coalesced to meet the low latency requirements, but many large-granularity TLB entries from L2 TLB or PTE coalescing can be inserted into L1 TLB when it misses. Therefore, the coverage of L1 TLB has been effectively expanded, thereby achieving a lower L1 TLB miss rate. Such a low miss rate shows that GATLB can complete more address translations in the fast L1 TLB, which will provide better performance than other methods. Compared with the results of Base TLB, GATLB can reduce the L1 miss rate by an average of 45.25%, while CoLT can reduce the miss rate by about 25.17%, and the miss rate of Anchor is the same as that of Base TLB due to the same L1 TLB entry management schemes between them.

D. Translation CPI

Fig. 6 compares the breakdown of the cycles spent per instruction in the address translation, normalized to those of Base TLB. Here we use the latencies showed in Table I to calculate the CPI for each method. Note that the access latency of L1 TLB is hidden as it is accessed in parallel with the cache [21], so it is omitted in the results. L2 hit in Fig. 6 denotes CPIs spent for regular L2 TLB entry hit, while Anchor or GATLB hit represents CPIs spent for anchor entry or GATLB entry hit in L2 TLB.

Compared with Base TLB, GATLB can reduce the translation cycles by an average of 49.41%, which outperforms all the other methods. This is because GATLB achieves the largest TLB coverage and has the lowest TLB miss rate compared to its competitors, resulting in the fewest number of page table walks. Thus, GATLB not only spends the least cycles on the L2 TLB but also on the page table walks, thereby achieving the lowest address translation cycles.

IV. Conclusion

Inspired by the problems of invalid migration and page fragmentation caused by the conflict between coarse-grained superpage management and fine-grained page migration, we proposed the coexistence of multi-granularity pages in a hybrid memory system, and designed GATLB to support the management of TLB entries with different page granularities. After extensive experimental evaluations, it is found that GATLB not only greatly expands the TLB coverage and reduces the TLB miss rate, but also significantly shortens the address translation cycles and can provide fast address translation.

V. Acknowledgment

This work was supported by grants from Natural Science Foundation of China No. 62072059, Open Project Program of Wuhan National Laboratory for Optoelectronics No. 2019WNLOKF009, Natural Science Foundation of Chongqing No. cstc2020jcyj-msxmX0897, the Fundamental Research Funds for the Central Universities No. 2020CDJLHZZ-050.

References

[1] M. Talluri and M. D. Hill. “Surpassing the TLB Performance of Superpages with Less Operating System Support.” *ASPLOS*, 1994.
[2] J. Navarro, et al. “Practical, Transparent Operating System Support for Superpages.” *SIGOPS*, 2002:89–104.
[3] B. Pham, et al. “Large Pages and Lightweight Memory Management in Virtualized Environments: Can You Have it Both Ways.” *MICRO*, 2015.
[4] Y. Du, et al. “Supporting Superpages in Non-Continuous Physical Memory.” *HPCA*, 2015.
[5] A. Basu, et al. “Efcient Virtual Memory for Big Memory Servers.” *ISCA*, 2013:237-248.
[6] G. Cox, and A. Bhattacharjee. “Efcient Address Translation for Architectures with Multiple Page Sizes.” *SIGARCH*, 2017:435-448.
[7] W. Korn, and M. S. Chang. “SPEC CPU2006 Sensitivity to Memory Page Sizes.” *SIGARCH*, 2007:97-101.
[8] X. Wang, et al. “Supporting Superpages and Lightweight Page Migration in Hybrid Memory Systems.” *TACO*, 2019:1-26.
[9] X. Chen, et al. “The Design of an Effcient Swap Mechanism for Hybrid DRAM-NVM Systems.” *EMSOFT*, 2016.
[10] Spec cpu 2017 benchmark. *https://www.spec.org/cpu2017/.*
[11] Y. Kwon, et al. “Coordinated and Efficient Huge Page Management with Ingens.” *OSDI*, 2016:705–721.
[12] Y Ban, et al. “Coalesced TLB to Exploit Diverse Contiguity of Memory Mapping”, *arXiv preprint arXiv:1908.08774*, 2019.
[13] M. Talluri, et al. “Tradeoffs in Supporting Two Page Sizes.” *ISCA*, 1992.
[14] I. Wienand, ”A Survey of Large-Page Support.” *University of New South Sales*, 2006:1-52.
[15] C. H. Park, et al. “Hybrid TLB Coalescing: Improving TLB Translation Coverage under Diverse Fragmented Memory Allocations.” *ISCA*, 2017:444–456.
[16] M. Papadopoulou, et al, “Prediction-Based Superpage-Friendly TLB Designs,” *HPCA*, 2014.
[17] J. Gandhi, et al. “Efficient Memory Virtualization,” *MICRO*, 2014.
[18] J. Buell, et al. “Methodology for Performance Analysis of VMware vSphere under Tier-1 Applications,” *VMWare Technical Journal*, 2013.
[19] B. Pham, et al. “Colt: Coalesced Large-Reach TLBs.” *MICRO*, 2012:258–269.
[20] Intel Corporation. Intel 64 and IA-32 Architectures Optimization Reference Manual, 2016.
[21] V. Karakostas, et al. 2016. “Energy-efficient Address Translation.” *HPCA*, 2016:631–643.
[22] N. Binkert, et al. “The gem5 simulators.” *SIGARCH*, 2011:1-7.
[23] Y. Tan, et al. “UIMigrate: Adaptive Data Migration for Hybrid Non-Volatile Memory Systems.” *DATE*, 2019.