Multimed Tools Appl (2017) 76:17937–17954
DOI 10.1007/s11042-016-3382-6

A study of partitioned DIMM tree management for multimedia server systems

Young-Kyu Kim[1] · Yong-Hwan Lee[2] · Byungin Moon[1]

Received: 28 October 2015 / Revised: 9 January 2016 / Accepted: 18 February 2016 /
Published online: 3 March 2016
© Springer Science+Business Media New York 2016

Abstract In-memory computing systems have been attracting considerable attention as a method for servicing high-quality multimedia contents. In-memory computing was intended to store entire data sets in the main memory of a computer to eliminate the need to access slow mechanical hard discs and increase the ability to process complex and large volumes of data. Prior studies have proposed a dual inline memory module (DIMM) tree architecture (DTA) as a new structure for implementing the in-memory computing system. The DTA can apply a partitioned DIMM tree policy to efficiently manage memory. However, the partitioned DIMM tree has several drawbacks, including hardware overhead resulting from additional fields in both the translation lookaside buffer (TLB) and the page table and the demand for an additional fast partition area for the fast partition page table (FPPT). To overcome these drawbacks, this paper proposes an advanced TLB management policy for the partitioned DIMM tree, DIMM tree TLB and two new partitioned DIMM tree management policies, fast-FPPT and slow-FPPT. We model the proposed policies using C language and verify them by special workloads in experiments employing a large-capacity memory system. The experimental results show how the proposed policies influence system performance and confirm that they overcome problems in the existing DTA. The simulations revealed a similarity between the performance of systems using the proposed policies and that of the existing DTA model. However, as the proposed policies demand a considerably lower hardware cost than the existing DTA model, the proposed policies are more practical.

Keywords DIMM tree architecture · Multimedia server system · In-memory computing · Big memory server · Main memory

✉ Byungin Moon
bihmoon@knu.ac.kr

[1] School of Electronics Engineering, Kyungpook National University, Daegu, Korea

[2] School of Electronic Engineering, Kumoh National Institute of Technology, Gumi, Korea

1 Introduction

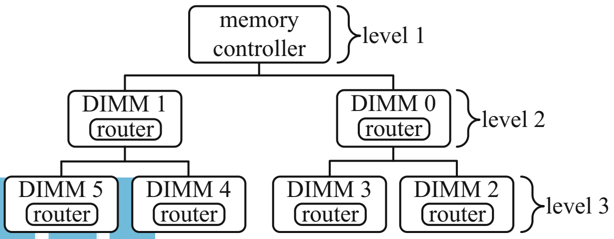
A number of recent studies have focused on systems for servicing multimedia contents. The storage performance of such systems has proven of particular interest because multimedia files have recently changed to a high capacity, high definition format [20, 24, 28]. As traditional disk-based server systems have proven inefficient because of poor disk performance, a number of related studies [3, 14, 21] have proposed using memory-based storage such as dynamic random-access memory (DRAM)-based storage. Moreover, some research has suggested employing in-memory computing technologies to address the increasing need for big-data management [29, 35]. As in-memory computing stores entire data sets in a computer's main memory, it precludes the need to access slow mechanical hard-drive I/O and increases the ability to process complex and large volumes of data [3].

Advancements in memory technology, a drastic decline in the price of memory, and the evolution of 64-bit multi-core processors have led to the development of in-memory computing technology [3]. However, traditional memory interfaces such as the multi-drop bus and point-to-point links (P2Ps) are not suitable for the in-memory computing platform because of their signal integrity and access latency. In a multi-drop bus, as the DRAM data rate increases, the number of dual inline memory modules (DIMMs) supported on the bus decreases because of a lack of signal integrity. In addition, each fully buffered DIMM (FB-DIMM) based on the P2P must buffer and repeat received signals, resulting in added latency. Therefore, the number of DIMMs on a P2P must be limited to avoid degrading throughput [5, 7, 32]. To solve the drawbacks of traditional memory interface methods and to implement a large-capacity memory system, DIMM tree architecture (DTA) has been proposed. DTA exponentially increases the number of DIMMs with each level of latency in the tree (Fig. 1) [32].

Prior studies pertaining to the DTA have addressed improvements in system performance and the implementation of an actual DTA system. In particular, Therdsteerasukdi et al. [33] suggested efficient policies for the management of a large-capacity main memory and a method for direct data transaction between two levels of DIMMs. Such policies are excellent for improving the performance of the DTA system, but they have several critical problems. Support of these policies requires modifying the TLB and the page table, resulting in hardware overhead. Furthermore, a fast partition page table (FPPT) used to implement such policies requires an additional fast partition area. Therefore, to mitigate the hardware overhead demand in the TLB, page table, and fast partition area, this paper proposes an advanced TLB management policy, referred to as DIMM tree TLB (DT-TLB) and new FPPT management policies, fast-FPPT and slow-FPPT [12].

This paper introduces the background of this research in Section 2, describes motivations for the research in Section 3, presents the proposed research methods in Section 4, explains our experiments and evaluates experimental results in Section 5, and concludes our study in Section 6.

Fig. 1 DIMM tree architecture



2 Background

The scaling capacity of the DRAM system has been the focus of several studies. Vogt [34] proposed the FB-DIMM to avoid the signal integrity problems experienced on the multi-drop bus. In their work, the FB-DIMM memory architecture employs a P2P serial interface between the memory controller and an intermediate buffer called the advanced memory buffer (AMB). Ganesh et al. [5] studied how traditional memory controller policies for scheduling and row buffer management perform on the FB-DIMM memory architecture. Jeddeloh and Keeth [10] proposed the hybrid memory cube (HMC) to improve DRAM bandwidth and energy efficiency by 3D-stacking DRAM dies on top of a logic die, and Kim et al. [11] proposed a memory-centric network in which all processor channels are connected to local HMCs and processor-to-processor communication is routed through local HMCs. However, the limited scalability of the FB-DIMM resulting from the repeat latency of the AMB calls for more study that leads to HMC commercialization.

The DTA employs a tree topology for connecting DIMMs. To support the DIMM tree, the DTA uses a special type of DIMM called a tree-DIMM (T-DIMMs) (Fig. 2), which has two external channels and one internal channel. External channels connect an upper-level DIMM to lower-level DIMMs in the DIMM tree. An internal channel drives DRAM ranks in the T-DIMM [32]. For the external channels, previous studies related to the DTA proposed the use of a multiband radio-frequency interconnect (MRF-I) to reduce T-DIMM pin overhead and improve signal integrity [31, 33]. However, because the MRF-I is an amplitude shift keying-based signal transmission technique and the MRF-I signals have sinusoidal characteristics, the signals must be converted into digital signals, executed by a parent DIMM MRF-I transceiver and a child DIMM MRF-I transceiver on the DIMM interface router (DIR) in the T-DIMM. Then, to access the ranks, a data rate converter changes the data rate and width of the converted digital signals. To process a received command, the T-DIMM must be able to choose among aborting the received command, executing that command, or forwarding it to lower-level T-DIMMs. Such operations are run by the router in the DIR (Fig. 2) [13].

Earlier research proposed a DTA using a partitioned DIMM tree policy for efficient memory management [33]. In the partitioned DIMM tree policy, T-DIMMs are separated into a fast partition composed of the T-DIMMs in the fastest level(s) of the DIMM tree and a slow partition composed of the T-DIMMs in the remaining level(s) (Fig. 3). The relationship between the two partitions is similar to that between the main memory and the hard disk

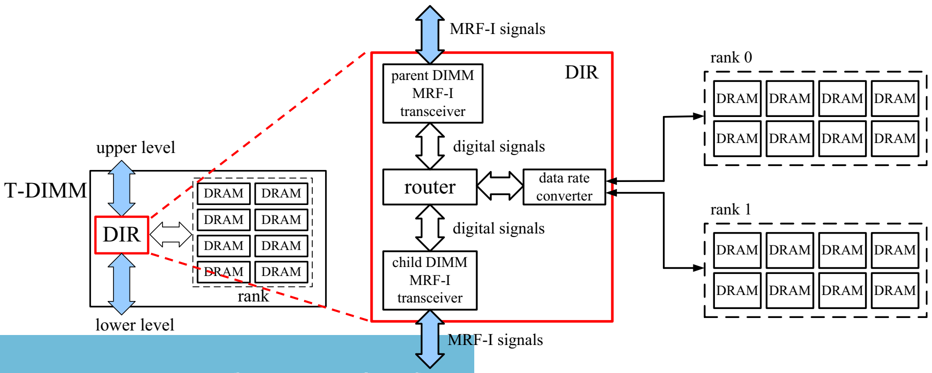


Fig. 2 T-DIMM and DIR block diagram

[30]. The fast partition acts as a cache for pages in the slow partition, which is used as the main memory. In addition, Therdsteerasukdi et al. [33] proposed a direct DIMM-to-DIMM transfer to reduce contention in the memory channel caused by the transfer of pages between the fast and slow partitions. The partitioned DIMM tree requires an FPPT that monitors the pages in the fast partition. The FPPT, whose management is similar to that of a traditional page table, is a set-associative structure to which some part of the fast or slow partition is assigned [12].

Figure 4 illustrates a four-way FPPT in the DTA using a 39-bit physical address. Access to the main memory initially requires access to the FPPT using the tag and the index of the slow partition address (physical address) to check for a hit or a miss in a fast partition. If the result is a hit, the fast partition page number is index × number of ways + way number, which is used to access the fast partition. However, if the result is a miss, the slow partition page must be uploaded into the corresponding fast partition page. If the fast partition set is already full, one of the entries in the set must be replaced by the page replacement policy [30, 33].

Recently, several studies related to the TLB have been devoted to reducing power consumption and improving performance. Such studies have influenced the design of the proposed DT-TLB in this paper. In one study, Sembrant et al. [27] proposed a tagless cache (TLC) that reduces power consumption. In their study, each TLB entry of the TLC has a collection of fields containing cache-line location information for all cache lines belonging to that page, called a cache-line location table (CLT). Each entry in the CLT contains a valid bit indicating the presence of the corresponding cache line and a way number identifying its location. Sembrant et al. [26] presented the direct-to-data (D2D) cache that locates data across the entire cache hierarchy with a single lookup. Each entry of the proposed DT-TLB is similar to that of the CLT in the TLC and the D2D cache. However, while each CLT entry has information about its corresponding cache line, each DT-TLB entry contains information about its corresponding page.

3 Motivation

The partitioned DIMM tree policy is an efficient method of improving system performance. However, it contains the following problems that result from use of the FPPT:

1) *Penalties of the modified TLB and page table*: Access to the main memory in the DTA always requires access to the FPPT to check the hit or miss in a fast partition after access of a page table for virtual to physical address translation. To solve this problem, Therdsteerasukdi et al. [33] proposed modifying the TLB and the page table. They added a flag bit and a field to each modified TLB and page table entry to specify whether the

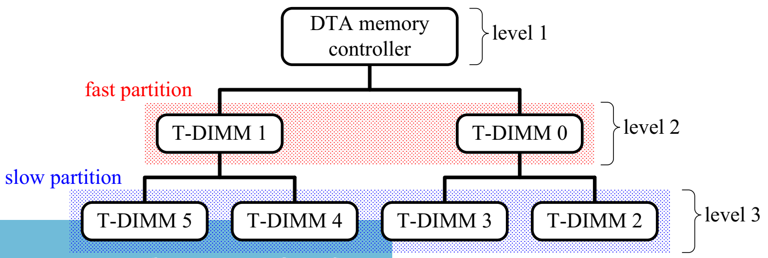


Fig. 3 Example of the fast and slow partitions in the DIMM tree architecture

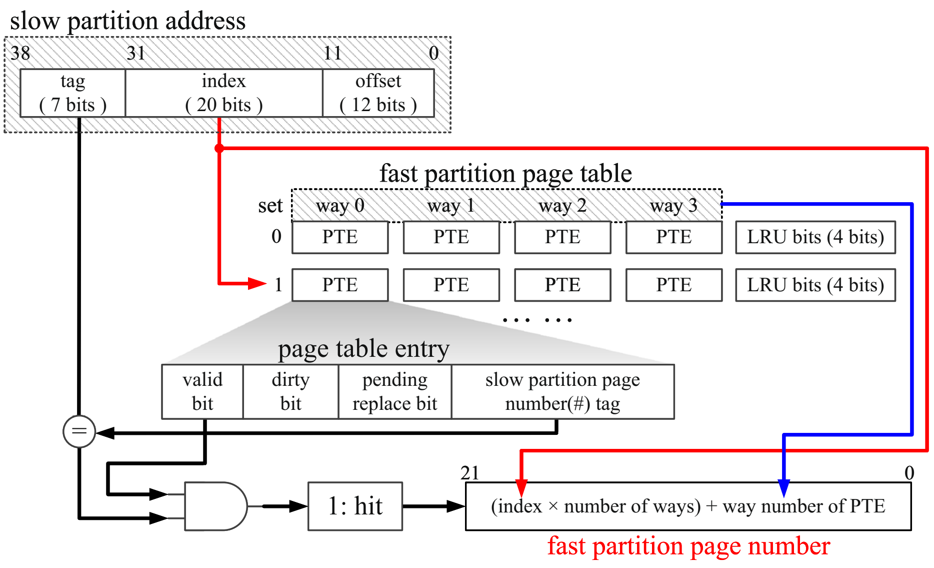
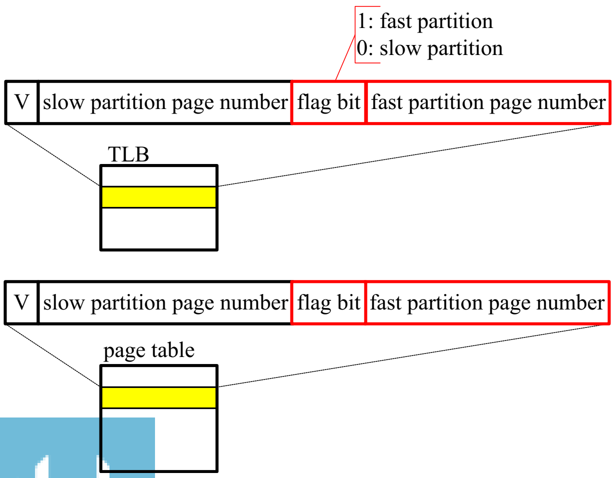


Fig. 4 Fast partition page table entry and fast partition page number

current page existed in the fast partition and to store the fast partition page number, respectively (Fig. 5). However, these modifications increased hardware costs. For example, if the DTA has a 16-GB fast partition and a 4-KB page size, each TLB entry and each page table entry must have additional one- and 22-bit fields for the flag and the fast partition page number, respectively. These additional fields cause substantial hardware overhead. To overcome these problems, this paper proposes DT-TLB, an advanced TLB management policy for the partitioned DIMM tree that does not modify the page table.

2) *Demand for an additional fast partition area*: The FPPT requires an additional fast partition area (Fig. 6). For example, if the DTA has a 16-GB fast partition, a 4-KB page size, and a 14-bit FPPT entry size (i.e., 1 bit for valid, 1 bit for dirty, 1 bit for pending replacement, 7 bits for tag, and 4 bits for LRU) [33], $2^{22} \times 14$ bits for the FPPT are

Fig. 5 TLB and page table of the existing DIMM tree architecture


1: fast partition
0: slow partition
V
slow partition page number
flag bit
fast partition page number
TLB
V
slow partition page number
flag bit
fast partition page number
page table

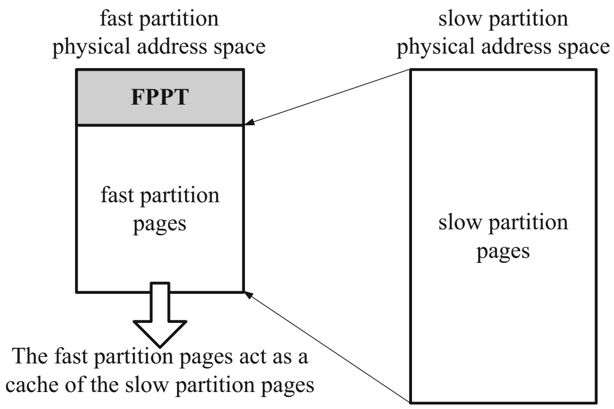

fast partition physical address space
slow partition physical address space
FPPT
fast partition pages
slow partition pages
The fast partition pages act as a cache of the slow partition pages

Fig. 6 The FPPT management policy in the existing DTA

required in the fast partition in addition to the above 16 GB. Thus, this paper proposes new FPPT management policies, fast-FPPT and slow-FPPT, and verifies that their performance without the additional fast partition area is similar to that of the existing FPPT management policy.

4 Proposed methods

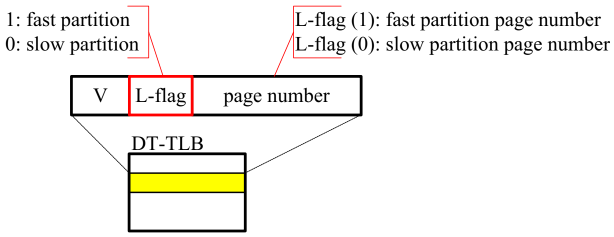
4.1 DT-TLB

To overcome the drawbacks of the TLB and page table of the existing DTA, this paper proposes a new TLB referred to as DT-TLB, which is similar to a traditional TLB [2, 22] but has an added location-flag (L-flag) bit that indicates whether the current page exists in the fast or slow partition (Fig. 7). If the L-flag is one, the current page exists in the fast partition, and the page number field in the DT-TLB entry has a fast partition page number; and if the L-flag is zero, the current page exists in the slow partition, and the page number field has a slow partition page number. The fast partition page number is assigned by the fast partition page number assign block in the DTA memory controller [9], which determines the fast partition page number from the slow partition address and the hit FPPT entry information, shown in Fig. 4.

Like the traditional TLB, the proposed DT-TLB acts as a cache of the page table that rapidly translates a virtual address into a physical address. However, it requires management of the L-flag and the page number field. Therefore, we organize the management of the DT-TLB as follows. When a DT-TLB miss occurs, uploading the corresponding page number into the DT-TLB requires accessing the page table. Then, from the index and tag of the uploaded page number, the system must check the FPPT. If the FPPT access hits, that is, the page exists in the fast partition, the L-flag of the TLB entry becomes one (Fig. 8a); otherwise, it is zero (Fig. 8b). Also, the system updates the page number field of the current DT-TLB entry as either a fast or slow partition page number according to the L-flag.

When a last-level cache (LLC) miss occurs, the cache line must be uploaded from either the fast or slow partition. The selection of the partition accessed is determined by the results of the DT-TLB check and update in the previous step (Fig. 8). If the L-flag of the corresponding DT-TLB entry is one, the system accesses the fast partition without checking the FPPT (Fig. 9a).

Fig. 7 Entry of the proposed DT-TLB


1: fast partition
0: slow partition
L-flag (1): fast partition page number
L-flag (0): slow partition page number
V
L-flag
page number
DT-TLB

However, if it is zero, the system uploads the page from the slow to fast partition, which results in updates to the FPPT and the DT-TLB entry (Fig. 9b).

If the fast partition is full, one page is evicted and a new page uploaded by the page replacement policy. When a page is evicted from the fast partition, its corresponding DT-TLB entry must also be updated. However, because the FPPT has no information with which to track the DT-TLB entry that corresponds to the evicted page, the system must search for an appropriate DT-TLB entry by using the fast partition page number of the evicted page (Fig. 10a). Page upload from the slow to fast partition follows page eviction. To offset the performance overhead resulting from DT-TLB search and update, page eviction and upload and DT-TLB search and update can occur simultaneously (Fig. 10).

The proposed DTA using DT-TLB reduces hardware overhead resulting from the modification of both the TLB and page table in the existing DTA. For example, if each fast partition page number is 22 bits long and a two-level TLB hierarchy has 576 entries, the TLB of the existing DTA requires an additional 576×23 bits; the DT-TLB, by contrast, requires only an additional 576×1 bits. In addition, if a virtual page number is 27 bits long, the page table of the existing DTA requires an additional $2^{27} \times 23$ bits; however, because the DT-TLB uses the traditional page table without modification, the proposed DTA does not require any additional bits (Fig. 11).

4.2 Fast-FPPT and slow-FPPT

To address the additional fast partition area problem identified in Section 3, this paper proposes two policies, fast-FPPT and slow-FPPT. The fast-FPPT allocates a small part of one way of the fast partition to the FPPT area (Fig. 12a). Because the fast-FPPT resides in the original fast partition (16 GB in the example presented in Section 3), it does not require an additional fast partition area, thus reducing hardware costs. However, the proposed fast-FPPT can lead to the

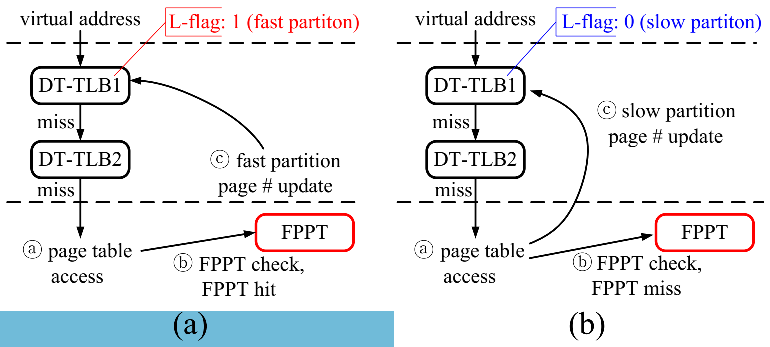


Fig. 8 DT-TLB miss and update: **a** with an FPPT hit and **b** with an FPPT miss

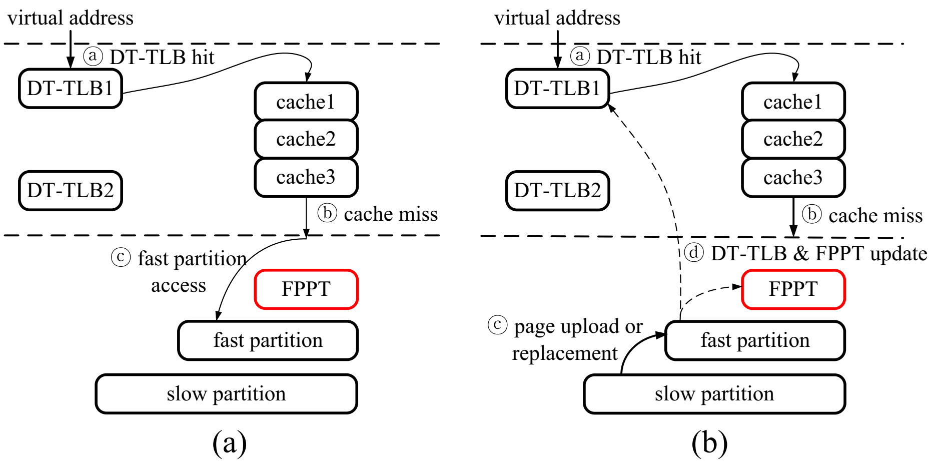


Fig. 9 Cache miss: **a** with L-flag of one and **b** with L-flag of zero

more frequent replacement of pages in the fast partition. For example, if the FPPT occupies 1024 pages of the fast partition of way 0 and it is four-way set-associative, the fast partition page sets from 0 to 1023 can use just three of the ways. As a result, the pages of the sets from 0 to 1023 can be replaced more frequently than those of the other sets. However, because the sets used by the FPPT comprise a relatively small part of the fast partition in multimedia server systems and the other sets can still use all four ways, its negative effect is negligible.

In a similar way, the slow-FPPT allocates a small part of the slow partition to the FPPT area (Fig. 12b), so it does not require an additional fast partition area either. This policy, however, could lead to inefficient performance because the access time of the slow partition is greater than that of the fast partition. However, because the FPPT in the slow partition would seldom be accessed, its negative influence on system performance is minimized. After all, the hit-rate of TLBs is generally very high [2], and most requests are processed by the L-flag of the DT-TLB without accessing the FPPT in the slow partition.

In summary, although the fast- and slow-FPPTs may negatively reduce performance, they reduce hardware costs. This negative influence, however, is expected to be negligible. Thus, to analyze the influence of the two policies on system performance, this paper models and simulates both the fast- and slow-FPPTs with the DT-TLB.

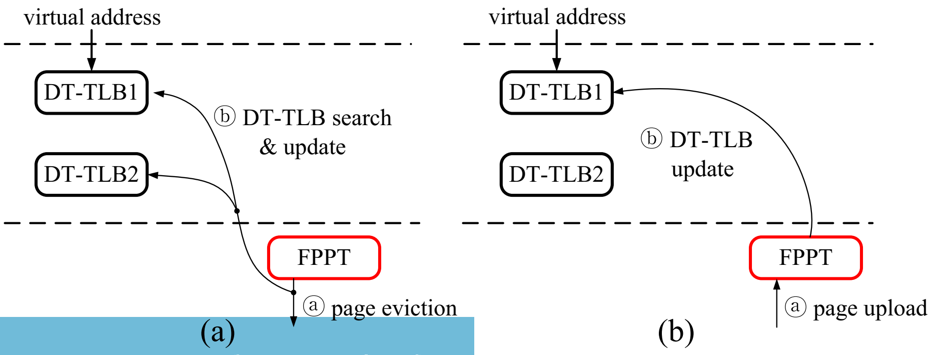


Fig. 10 FPPT replacement: **a** a page eviction and **b** a page upload

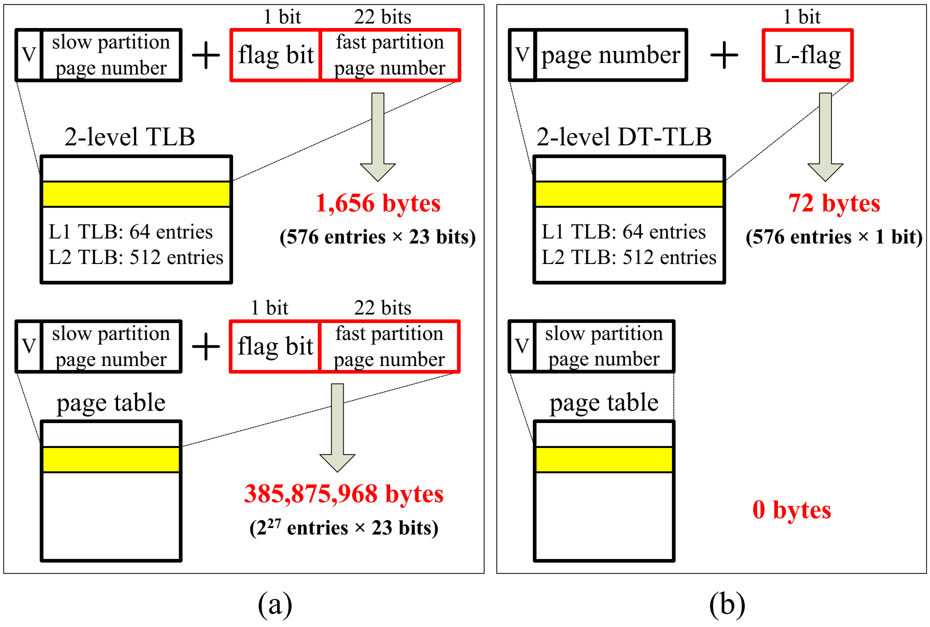


Fig. 11 Additional hardware costs of the TLB and the page table: **a** the existing DTA and **b** the proposed DTA

5 Experiments and results

5.1 Workloads and modeling

To ensure accurate operation of the proposed DT-TLB and to evaluate the effect of the proposed fast- and slow-FPPTs on system performance, this paper conducted several experiments. For experiments on a large-capacity main memory system, we prepared 14 workloads

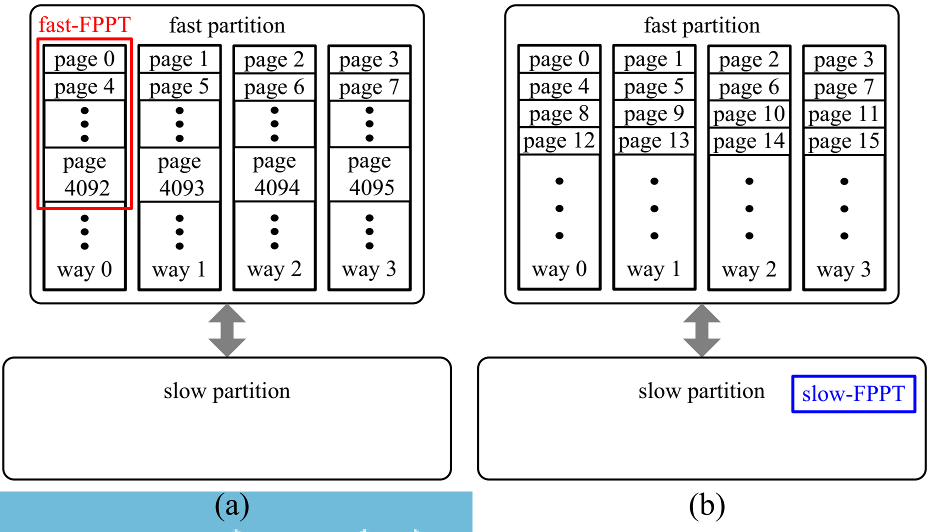


Fig. 12 Fast- and slow-FPPT: **a** FPPT in the fast partition (fast-FPPT) and **b** FPPT in the slow partition (slow-FPPT)

that we extracted from the SPEC OMP2012 benchmark by the pin tool [15, 18, 25]. Each workload had ten billion 48-bit virtual addresses generated by eight threads in a 64-bit operating system (OS). In addition, according to the locality of the extracted addresses, we classified the workloads into high, middle, and low memory-intensive workloads (Table 1). Workload_Set_1 consisted of five high memory-intensive workloads, and Workload_Set_2 had five middle memory-intensive workloads. The study did not use the four low memory-intensive workloads for the simulations because the number of main memory accesses was insufficient for verifying the performance of the DTA.

To verify the proposed DT-TLB operation and to predict the performance of the fast- and slow-FPPTs, we separately modeled the DTA systems using the fast- and slow-FPPTs with the DT-TLB through the C programming language. The DTA systems for the modeling were aimed at a high-end server computer system that supports high-quality multimedia content. Figure 13 shows the system architecture for the modeling. The system consisted of five processors, each containing a two-level DT-TLB hierarchy with a two-level cache hierarchy. The level 3 (L3) cache, external to the processors, was used as a shared cache. We modeled the two-level DT-TLBs and caches in the processors to be exclusive, and the relationship between the internal two-level caches and the L3 cache inclusive [36]. The size of the T-DIMM was 4 GB, and the DIMM tree had a branch factor and depth of four. Accordingly, the fast partition, which consisted of level 2 T-DIMMs, was 16 GB, and the two-level slow partition, which consisted of level 3 and 4 T-DIMMs, was 320 GB. The fast partition was four-way set-associative. This paper used a pseudo least recently used (PLRU) algorithm as a replacement policy for the DT-TLBs, the caches, and the FPPT, but overheads caused by implementing a particular policy such as adding PLRU bits and updating PLRU bits were excluded from the simulations. The PLRU was just one option; various replacement algorithms could be employed such as random or round-robin. Al-Zoubi et al. [1] revealed no common wisdom about the best algorithm to use. Because the slow partition used as the main memory was 320 GB, the physical address length was at least 39 bits. Because of the difficulty of generating a real physical address, we used the 39 least significant bits of the 48-bit virtual address as the physical address in the simulation. The upper nine bits of the virtual address are not suitable for the physical address because they are mostly used for the sign-extended field or address space ID [4, 6]. To compare the performance of the existing FPPT with that of the proposed DTAs, we also modeled the existing FPPT with the proposed DT-TLB. The existing FPPT had an additional fast partition area for the FPPT, and it employed the proposed DT-TLB instead of

Table 1 Workload description

Workload Set	Workloads from the SPEC OMP2012 Benchmark	Number of addresses	Description
Workload_Set_1	351.bwaves, 363.swim, 360.ilbdc 370.mgrid331, 372.smithwa,	Each benchmark has 10 billion addresses. (A total of 50 billions)	High memory-intensive workloads
Workload_Set_2	357.bt331, 359.botsspar, 362.fma3d, 367.imagick, 376.kdtree	Each benchmark has 10 billion addresses. (A total of 50 billions)	Middle memory-intensive workloads
Not used	350.md, 352.nab, 358.botsalgn, 371.applu331	Each benchmark has 10 billion addresses. (A total of 40 billions)	Low memory-intensive workloads

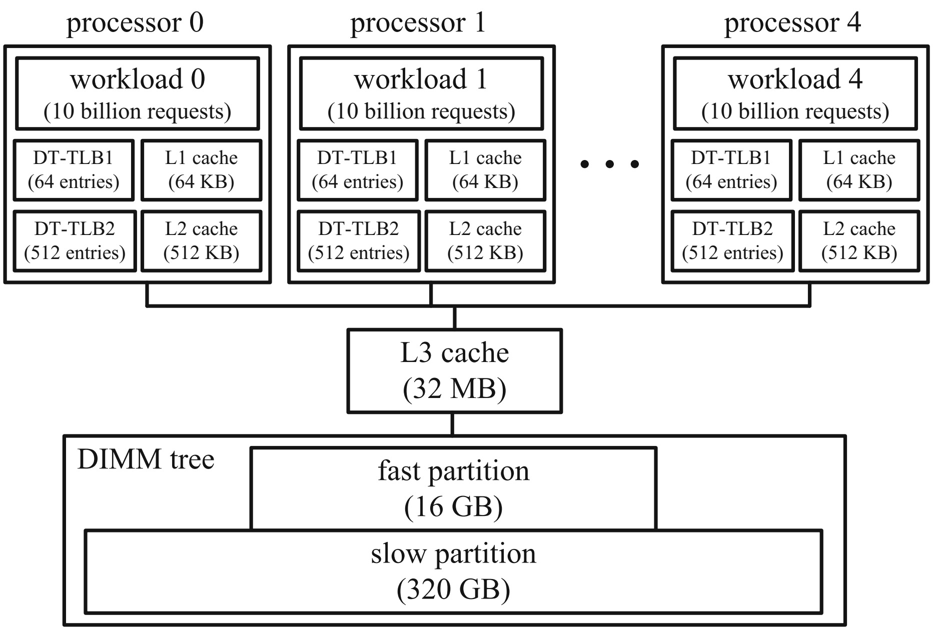


Fig. 13 System architecture using the proposed DTA

the TLB and the page table used for the existing DTA because the experiments were aimed at performance comparison of the existing FPPT and proposed FPPTs.

Table 2 Simulation parameters

Parameter		Configuration
Level 1 DT-TLB		8 ways, 64 entries, 1-clock-cycle latency
Level 2 DT-TLB		32 ways, 512 entries, 5-clock-cycle latency
Level 1 cache		8 ways, 1024 entries, 5-clock-cycle latency
Level 2 cache		16 ways, 8192 entries, 12-clock-cycle latency
Level 3 cache		32 ways, 524,288 entries, 30-clock-cycle latency
DIMM to DIMM switch time		1-clock-cycle latency
Fast partition	Level 2 T-DIMMs	16 GB, Read and write: 200-clock-cycle latency
Slow partition	Level 3 T-DIMMs	64 GB, Read: 200-clock-cycle latency + 2 DIMM to DIMM switch times Write: 200-clock-cycle latency + 1 DIMM to DIMM switch time
	Level 4 T-DIMMs	256 GB, Read: 200-clock-cycle latency + 4 DIMM to DIMM switch times Write: 200-clock-cycle latency + 2 DIMM to DIMM switch times

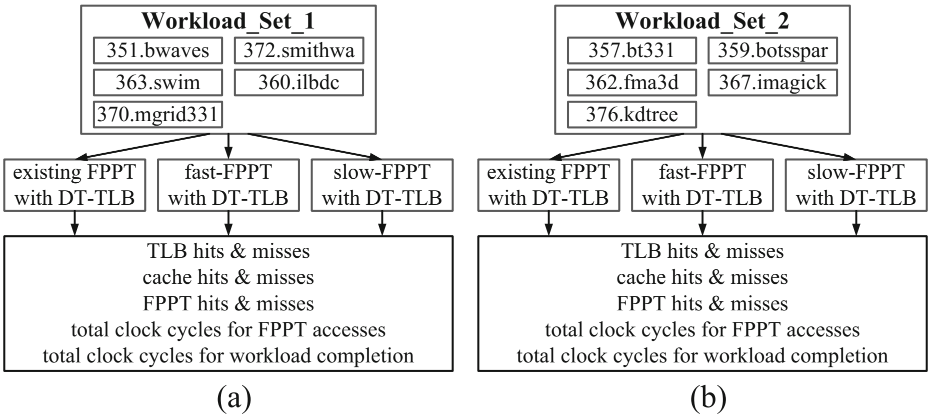


Fig. 14 Simulations of the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** for Workload_Set_1 and **b** for Workload_Set_2

We configured the parameters of the simulation similar to those of previous studies (Table 2) [4, 8, 19, 23, 27]. Every clock cycle latency means access time, except the DIMM to DIMM switch time, but the DIMM to DIMM switch time in Table 2 is the time delay for forwarding a command between two levels of T-DIMMs [13].

5.2 Simulation and results

We executed Workload_Set_1 and Workload_Set_2 in Table 1 in each of the three modeled systems and compared the simulation results. Each system was able to simultaneously run five workloads in the workload set using five processors. The L3 cache and the FPPT were shared by five processors, and a mutex function [16, 17] was used for processor synchronization; the processor scheduling depended on the OS executing the simulation. Using this multi-thread-based simulation, we obtained the simulation conditions of real systems such as the natural contentions of shared resources and a large main memory load. Then we counted the hits and misses of the TLBs, the caches, and the FPPT by processing all of the requests of each workload in the modeled systems and further calculated the total clock cycles for accessing the FPPT and processing all the requests (Fig. 14). Each system processed a total of 50 billion

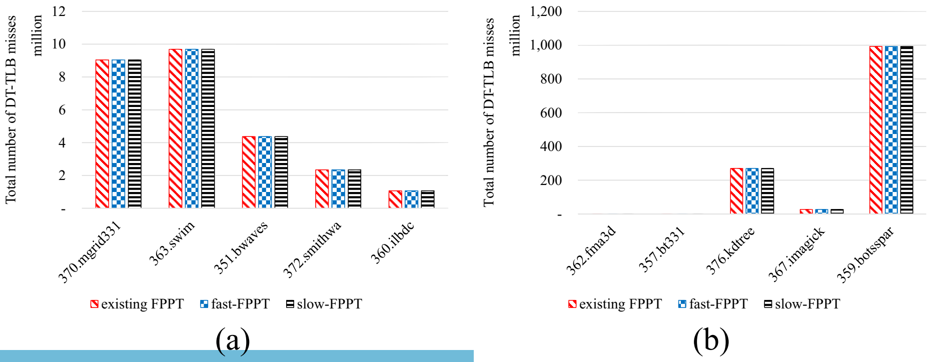


Fig. 15 Numbers of DT-TLB misses in the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** the results of Workload_Set_1 and **b** the results of Workload_Set_2

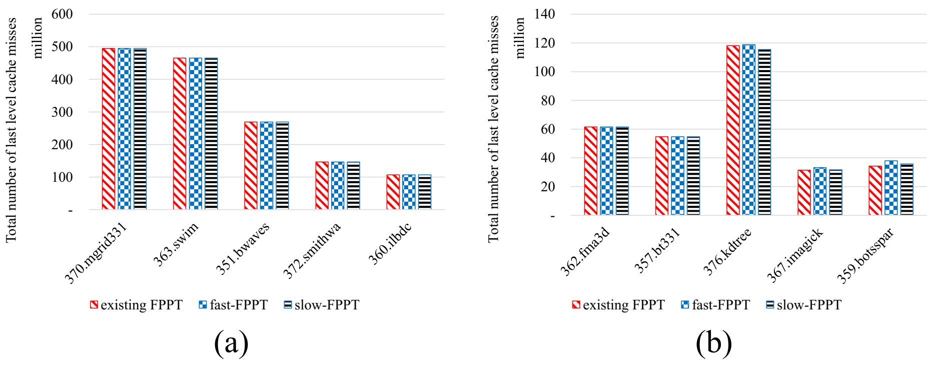


Fig. 16 Numbers of LLC misses in the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** the results of Workload_Set_1 and **b** the results of Workload_Set_2

requests because Workload_Set_1 and Workload_Set_2 each consisted of five workloads, each with ten billion requests. Figures 15, 16, 17, 18 and 19 show the simulation results of the existing FPPT with the DT-TLB, the fast-FPPT with DT-TLB, and the slow-FPPT with DT-TLB models. We analyzed the factors affecting system performance, such as number of TLB misses, number of cache misses, and number of FPPT misses. We also compare the system performance of the three models using the total number of clock cycles for the FPPT accesses and the number of clock cycles for processing the ten billion requests in each workload.

Figure 15 shows the counts of the DT-TLB misses for processing the ten billion requests in each workload. The DT-TLB misses significantly impacted the performance of the three FPPTs using the DT-TLB because when a DT-TLB miss occurred, the FPPT had to be checked before the DT-TLB was updated after the page table walk (Fig. 8). Particularly, the numbers of misses of the 376.kdtree and 359.botsspar in Workload_Set_2, which had low spatial and temporal locality, were larger than those of the other workloads. Figure 16 shows the counts of the LLC misses during the execution of the two workload sets. Because Workload_Set_1 contained high memory-intensive workloads overall, its miss counts were higher than those of Workload_Set_2.

Figure 17 shows the counts of the FPPT misses in processing the workloads. An FPPT miss was determined by the L-flag value of the corresponding DT-TLB entry after an LLC miss

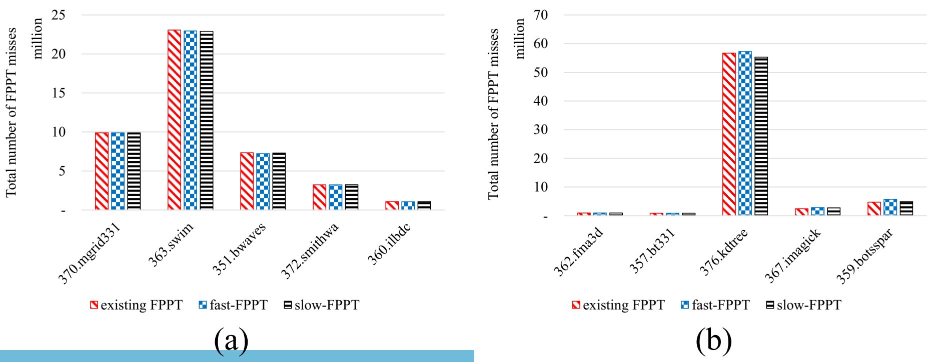


Fig. 17 Numbers of FPPT misses in the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** the results of Workload_Set_1 and **b** the results of Workload_Set_2

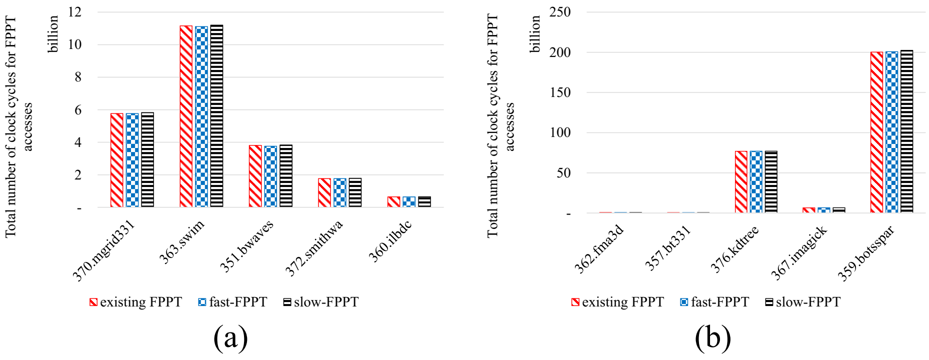


Fig. 18 Numbers of clock cycles required for accessing the FPPT in the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** the results of Workload_Set_1 and **b** the results of Workload_Set_2

(Figs. 8 and 9). The counts of FPPT misses in the three models were similar at high memory-intensive workloads (Fig. 17a), but the counts of the FPPT misses in the fast-FPPT were higher than those in the other models at 362.fma3d, 376.kdtree, 367.imagick, and 359.botsspar of Workload_Set_2, which were middle memory-intensive and had a high locality of memory references (Fig. 17b). These results show that the hit rate of the fast-FPPT was lower than the rates of the other models at workloads with a high locality of memory references because the fast-FPPT can use fewer fast partition pages than the other two models.

Figure 18 shows the numbers of clock cycles required for accessing the FPPT during the processing of the workload requests. These results were influenced by the counts of DT-TLB misses and FPPT misses at the workloads. Particularly, FPPT misses tended to increase the number of clock cycles for FPPT accesses because they required additional FPPT accesses in order to update the fast partition. Overall, the numbers of clock cycles of the slow-FPPT were slightly larger than those of the other models because the slow-FPPT required more clock cycles for each FPPT access than the other models. The difference, however, was negligible, and the results of all the models were almost the same.

Figure 19 shows the total numbers of clock cycles required for processing the ten billion requests in each workload. The results of most workloads were similar among the existing FPPT, the fast-FPPT, and the slow-FPPT. The total numbers of clock cycles at 359.botsspar

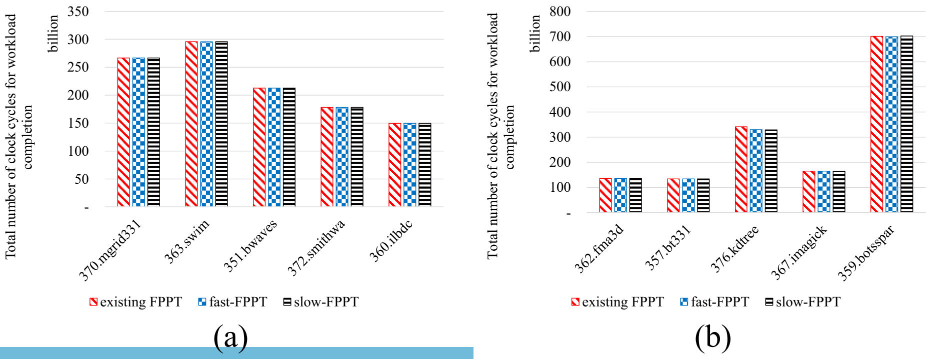


Fig. 19 Total number of clock cycles required for workload execution in the existing FPPT, the fast-FPPT, and the slow-FPPT: **a** the results of Workload_Set_1 and **b** the results of Workload_Set_2

were larger than those at other workloads because 359.botsspar required the largest number of the DT-TLB misses (Fig. 15). DT-TLB misses led to FPPT checks demanding accesses to either the fast or slow partition, so they had a strong negative influence on the overall performance of the DTA. For a similar reason, the total numbers of clock cycles at 376.kdtree were the second largest because 376.kdtree showed the second largest number of DT-TLB misses and the largest number of FPPT misses (Fig. 17). The overall analysis showed that system performance was primarily influenced by the DT-TLB misses, the FPPT misses, and the FPPT accesses. Because the existing FPPT, the fast-FPPT, and the slow-FPPT showed similar results with respect to these factors, as predicted in Section 4.2, the difference in their performance was negligible. Thus, although the proposed fast- and slow-FPPTs require lower hardware costs for the fast partition than the existing FPPT, their performance is virtually the same as that of the existing FPPT.

6 Conclusion

This study explored memory system architectures for a high-performance multimedia server system. The proposed DT-TLB and its detailed management policies reduced the hardware overhead of the TLB and the page table in the existing DTA. To overcome the need to add a fast partition area for the FPPT, this paper proposed new FPPT management policies, referred to as the fast-FPPT and the slow-FPPT. To verify the proposed policies, this paper modeled the existing FPPT, the fast-FPPT, and the slow-FPPT with DT-TLB and conducted simulations using special workloads designed for testing a large main memory system. The design of the simulation framework was based on a multi-processor system such as a high-end multimedia server. The simulation results demonstrated that the performance of the proposed FPPTs was similar to that of the existing FPPT, but the proposed FPPTs significantly reduced hardware costs. Through this study, we confirmed that by removing the need for additional hardware resources, the proposed DT-TLB and FPPTs could overcome the problems of the existing DTA. As a study for an in-memory computing-based multimedia server, this paper studied new hardware architectures for implementing actual DTA systems. It is expected to contribute to the implementation of a high-performance multimedia server for supporting high-quality multimedia content. Future work will involve a study of the detailed hardware architecture and power optimization of the DIMM tree and T-DIMM for actual implementation of the DIMM tree architecture.

Acknowledgments This investigation was financially supported by Semiconductor Industry Collaborative Project between Kyungpook National University and Samsung Electronics Co. Ltd.

References

1. Al-Zoubi H, Milenkovic A, Milenkovic M (2004) Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite. Proceedings of the 42nd annual Southeast Regional Conference (ACM-SE 42), Huntsville, AL, USA, p 267–72
2. Bhattacharjee A, Martonosi M (2009) Characterizing the TLB behavior of emerging parallel workloads on chip multiprocessors. Proceedings of Parallel Architectures and Compilation Techniques, 2009. PACT '09 18th International Conference on, Releigh, NC, USA, p 29–40

3. Brocke JV, Debortoli S, Müller O, Reuter N (2014) How in-memory technology can create business value: insights from the hilti case. Commun Assoc Inf Syst 34(1):151–167
4. Du Y, Zhou M, Childers BR, Mossé D, Melhem R (2015) Supporting superpages in non-contiguous physical memory. Proceedings of High Performance Computer Architecture (HPCA), 2015 I.E. 21st International Symposium on, Burlingame, CA, USA, p 223–34
5. Ganesh B, Jaleel A, Wang D, Jacob B (2007) Fully-buffered DIMM Memory architectures: understanding mechanisms, overheads and scaling. Proceedings of High Performance Computer Architecture (HPCA), 2007 I.E. 13th International Symposium on, Scottsdale, AZ, USA, p 109–20
6. Jacob B, Mudge T (1998) Virtual memory in contemporary microprocessors. IEEE Micro 18(4):60–75
7. Jacob B, Ng SW, Wang DT (2010) Memory system: cache, DRAM, disk. Morgan Kaufmann
8. Jaleel A, Nuzman J, Moga A, Steely Jr. SC, Emer J (2015) High performing cache hierarchies for server workloads: relaxing inclusion to capture the latency benefits of exclusive caches. Proceedings of High Performance Computer Architecture (HPCA), 2015 I.E. 21st International Symposium on, Burlingame, CA, USA, p 343–53
9. Jang YJ, Kim YK, Ahn T, Moon B (2014) A memory controller for the DIMM tree architecture. Proceedings of the Eight International Conference on Advanced Engineering Computing and Application in Sciences, Rome, Italy, p 86–90
10. Jeddeloh J, Keeth B (2012) Hybrid memory cube new DRAM architecture increases density and performance. Proceedings of VLSI Technology (VLSIT), 2012 Symposium on, Honolulu, HI, USA, p 87–8
11. Kim G, Kim J, Ahm JH, Kim J (2013) Memory-centric system interconnect design with hybrid memory cubes. Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13). IEEE Press, Piscataway, NJ, USA, p 145–56
12. Kim YK, Lee YH, Moon B (2015) A study on fast partition page table management for the DIMM tree architecture. Proceedings of the International Conference on Intelligent Information Technology and International Workshop on Advanced Computing and Multimedia Technology, Guam, USA, p 29–39
13. Kim YK, Moon B (2015) A T-DIMM ID based command routing method for the DIMM tree architecture. Int J Control Autom Syst 8(2):43–54
14. Lechtenbörger J, Vossen G, Zeier A, Krüger J, Müller J, Lehner W, Kossmann D, Fabian B, Günther O, Winter R (2011) In-memory databases in business information systems. Bus Inf Syst Eng 3(6):389–395
15. Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K (2005) Pin: building customized program analysis tools with dynamic instrumentation. Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05), Chicago, IL, USA 40(6):190–200
16. MSDN, CreateMutex function, https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms682411(v=vs.85).aspx
17. MSDN, WaitForSingleObject function, https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms687032(v=vs.85).aspx
18. Müller MS, Baron J, Brantley WC, Feng H, Hackenberg D, Henschel R, Jost G, Molka D, Parrott C, Robichaux J, Shelepugin P, Waveren MV, Whitney B, Kumaran K (2012) SPEC OMP2012: an application benchmark suite for parallel systems using OpenMP. Proceedings of 8th International Workshop on OpenMP, IWOMP 2012, Rome, Italy, p 223–36
19. Muralimanohar N, Balasubramonian R, Jouppi NP (2009) CACTI 6.0: a tool to understand large caches. University of Utah and Hewlett Packard Laboratories, Tech. Rep
20. Nieh J, Yang SJ (2000) Measuring the multimedia performance of server-based computing. Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video, Chapel Hill, NC, USA, p 55–64
21. Ousterhout J, Agrawal P, Erickson D, Kozyrakis C, Leverich J, Mazières D, Mitra S, Narayanan A, Parulkar G, Rosenblum M, Rumle SM, Stratmann E, Stutsman R (2011) The case for RAMClouds: scalable high-performance storage entirely in DRAM. Commun ACM 54(7):121–130
22. Pham B, Bhattacharjee A, Eckert Y, Loh GH (2014) Increasing TLB reach by exploiting clustering in page translations. Proceedings of High Performance Computer Architecture (HPCA), 2014 I.E. 20th International Symposium on, Orlando, FL, USA, p 15–9
23. Qureshi MK, Srinivasan V, Rivers JA (2009) Scalable high performance main memory system using phase-change memory technology. Proceedings of 36th Annual International Symposium on Computer Architecture (ISCA '09), Austin, TX, USA, p 24–33
24. Ramesh B, Savitha N, Manjunath AE (2013) Mobile applications in multimedia cloud computing. Int J Comput Technol Appl 4(1):97–103
25. Saito H, Gaertner G, Jones W, Eigenmann R, Iwashita H, Lieberman R, Waveren MV, Whitney B (2003) Large system performance of SPEC OMP benchmark suites. Int J Parallel Prog 31(3):197–209

26. Sembrant A, Hagersten E, Black-Schaffer D (2014) 2014. The Direct-to-Data (D2D) cache: navigating the cache hierarchy with a single lookup. Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA '14), IEEE Press, Piscataway, NJ, USA 42(3), p 133–44
27. Sembrant A, Hagersten E, Black-Shaffer D (2013) TLC: a tag-less cache for reducing dynamic first level cache energy. Proceedings of 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46). ACM, Davis, CA, USA, p 49–61
28. Shirale S, Patmas M, Gunjal P, Rane D (2015) A online multimedia data processing on cloud and hadoop platform. Int J Comput Technol Electron Eng 5(2):21–24
29. Shoro AG, Soomro TR (2015) Big data analysis: Ap apsrk perspective. Global J Comput Sci Technol: C, Softw Data Eng 15(1):7–14
30. Silberschatz A, Calvin PB, Gagne G (2012) Operating system concepts, 9th edn. Addison-Wesley
31. Therdsteerasukdi K, Byun G, Cong J, Chang MF, Reinman G (2012) Utilizing RF-I and intelligent scheduling for better throughput/watt in a mobile GPU memory system. ACM Trans Archit Code Optim (TACO) 8(4):51
32. Therdsteerasukdi K, Byun GS, Ir J, Reinman G, Cong J, Chang MF (2011) The DIMM tree architecture: a high bandwidth and scalable memory system. Proceedings of 2011 I.E. 29th International Conference on Computer Design (ICCD), Amherst, MA, USA, p 388–95
33. Therdsteerasukdi K, Byun GS, Ir J, Reinman G, Cong J, Chang MF (2012) Utilizing radio-frequency interconnect for a many-DIMM DRAM system. IEEE J Emerging Sel Top Circuits Syst 2(2):210–227
34. Vogt P (2004) Fully buffered DIMM (FB-DIMM) server memory architecture: capacity, performance, reliability, and longevity. Intel Developer Forum, San Francisco, NC, USA: Session OSAS008
35. Zhang H, Chen G, Ooi BC, Tan KL, Zhang M (2015) In-memory big data management and processing: a survey. IEEE Trans Knowl Data Eng 27(7):1920–1948
36. Zheng Y, Davis BT, Jordan M (2004) Performance evaluation of exclusive cache hierarchies. Proceedings of Performance Analysis of Systems and Software, 2004 I.E. International Symposium on - ISPASS, Austin, TX, USA, p 86–96

Young-Kyu Kim is a Ph.D. student in the School of Electrical Engineering, Kyungpook National University at Daegu in Korea. He received the B.S degree in Department of Electronic Engineering in Gyeongju University in 2005 and M.S. degree in Electrical Engineering & Computer Science in Kyungpook National University in 2010. His research interests are in the following areas: SoC (system on a chip), computer architecture, and multiprocessor system.

Yong-Hwan Lee is currently a professor in the School of Electronic Engineering, Kumoh National Institute of Technology at Gumi in Korea. He received the B.S. degree, M.S. degree and Ph.D. degree in Electronic Engineering in Yonsei University in 1993, 1995 and 1997 respectively. He spent four years as a Senior Researcher in Hynix Semiconductor Inc., and also spent two years as a Senior Researcher in Samsung Electronics. His research interests are in the following areas: SoC, embedded system and software, and microprocessor architecture.

Byungin Moon is currently a associate professor in the School of Electronics Engineering, Kyungpook National University at Daegu in Korea. He received the B.S. degree and M.S. degree in Electronic Engineering in Yonsei University in 1995 and 1997, respectively. He received a Ph.D. degree in Electrical & Electronic Engineering in the same university in 2002. He spent two years as a Senior Researcher in Hynix Semiconductor Inc., and also worked as a Senior Researcher in Yonsei University for one year. His research interests are in the following areas: SoC, computer architecture, and vision processor.