IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 17 July 2023, accepted 20 August 2023, date of publication 25 August 2023, date of current version 30 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3308591

RESEARCH ARTICLE

Page-Size Aware Buddy Allocator With Unaligned Range Supports for TLB Coalescing

TRAN DAI DUONG AND JAE YOUNG HUR, (Member, IEEE)
Department of Electronic Engineering, Jeju National University, Jeju-si 63243, South Korea

Corresponding author: Jae Young Hur (jaeyoung.hur@jejunu.ac.kr)

This work was supported by the Research Grant of Jeju National University, in 2022.

ABSTRACT It is well known that the traditional page-based address translation scheme has limited translation look-aside buffer (TLB) reach and page-table walk overheads. A TLB coalescing scheme reduces these problems by representing an address range in a TLB entry. However, the conventional physical memory allocator has the power-of-2 block size and the address alignment restrictions. As a result, it is difficult to utilize diverse contiguities in memory and exploit the capability of TLB coalescing. To alleviate these issues, in the context of eager paging for I/O devices, we propose the flexible physical memory allocator that can represent unaligned ranges within the page sizes defined in the machine architecture. Combined with TLB coalescing, the presented scheme can efficiently utilize the contiguity in memory and reduce page-table walks. Considering the binary buddy allocator as a baseline, we present an algorithm, a design, analyses, a case study, an implementation, and evaluations. The experimental results indicate the presented scheme can improve memory utilization, TLB performance, and system performance.

INDEX TERMS Memory management, allocation, architecture, translation look-aside buffer, performance.

I. INTRODUCTION

Modern system-on-chip (SoC) typically accommodates memory management units (MMUs) to support virtual memory and protection. An MMU translates a virtual address into a physical address. The mapping information to translate the address is stored in a translation look-aside buffer (TLB) entry. If a TLB is miss, MMU conducts a page-table walk (PTW) to load the mapping information from a page table in memory. A traditional TLB entry can serve a single page translation and the number of TLB entries is limited. Accordingly, TLB misses often frequently occur when the address pattern has low locality. A conventional method to reduce TLB misses is to use multiple page sizes defined in the underlying architecture. However, the traditional method does not fully utilize diverse contiguities other than page sizes. To improve TLB performance, a number of TLB coalescing schemes are proposed. In [1], [2], [3], [4], [5], [6], [7], and [8], a various-sized address range is represented in a single TLB entry. Then the TLB entry can serve the larger range than a single page. Subsequently, TLB coalescing can reduce page-table walks and improve TLB reach.

The modern operating system (OS) dynamically allocates physical memory typically in block level. As an example, in the *malloc* function of the C standard library, the buddy algorithm efficiently allocates a set of contiguous free physical pages in a block [9], [10]. However, the conventional physical memory allocator has two restrictions. First, a block size should be power-of-2 pages. Second, the address of a block should be aligned with a block size. Though these restrictions make the implementation efficient, OS often handles contiguous physical space as fragmented blocks. As a result, the size and the alignment restrictions can degrade memory utilization. On the other hand, the conventional allocator is oblivious to the underlying machine architecture. Then the algorithm conducts many (often unnecessary) split operations to maintain the blocks while their sizes are not used in the hardware. Subsequently, it is difficult for the system to fully exploit the capability of an allocator and the advantage of TLB coalescing. To alleviate these issues, we propose the flexible allocator that can handle unaligned ranges. The proposed allocator is architecture specific in that

The associate editor coordinating the review of this manuscript and approving it for publication was Young Jin Chun.

it is customized for the page sizes defined in the architecture. The presented approach combines the effectiveness of TLB coalescing and the efficiency of the legacy buddy allocator. The main contributions of this paper are:

- The architecture-specific allocation algorithm called page buddy system (PBS) is presented. PBS inherently maintains physical memory in ranges within page sizes. PBS alleviates the power-of-2 size and the address alignment restrictions.
- The TLB coalescing scheme that utilizes PBS is presented. The presented scheme supports unaligned large-sized blocks.
- An algorithm, a design, analyses, a case study, an implementation, and performance evaluations are presented.

This paper is organized as follows. In Section II, related work is described. In Section III, conventional designs are described. In Section IV, the proposed design is presented. In Section V, experimental results are presented. Finally, conclusion is drawn in Section VI.

II. RELATED WORK

A. TLB COALESCING

In [1], the information on the number of contiguous pages is represented in a page table. In [2] and [3], multi-page mapping approaches are presented. In [2] and [3], a small set of pages (for example, 8 or 16) is packed into a TLB entry. This limits the ability to coalesce when a block size is large. In [4], the special range table is implemented by modifying system calls. Additionally, the redundant TLB hardware design is presented. In [5], the hybrid TLB coalescing scheme is presented. In software, anchor entries are added to the page table. In hardware, MMU handles the anchor page-table walks together with the regular page-table walk. In [6], the block-level TLB coalescing is presented. The design in [6] exploits the fact that the buddy allocator efficiently allocates power-of-2 sized blocks. In [7], the page-table compaction technique for TLB coalescing is proposed. In [7], for given memory allocation, OS packs multiple distinct blocks in adjacent page-table entries. In [8], the TLB coalescing scheme that supports page migration between flash memory and DRAM in the hybrid memory system is presented. In [8], the unaligned range migration is supported using the inverted page table. Our work is similar to [1], [2], [3], [4], [5], [6], [7], and [8] in that various address ranges can be represented in a TLB entry. However, the designs in [1], [2], [3], [4], [5], [6], [7], and [8] highly rely on the contiguity that the buddy allocator provides. In contrast, we present the allocation algorithm that implements the range concept. Unlike [4], [5], and [8], our design does not require additional TLB hardware components, separate special OS processes, the additional range table, or the inverted page table.

B. MULTIPLE PAGE SIZES

In [11], the in-place coalescer to transparently coalesce small pages into a large page without data movement is presented. In [12], to increase TLB reach, the design to map a large region to a TLB entry is presented. In [12], OS can promote pages into a superpage when reserved pages are accessed. In [13], the design to treat non-contiguous pages (due to retirement) as contiguous and construct large pages is presented. In [14], the TLB design to efficiently support multiple page sizes is presented. In [11], [12], [13], and [14], it is difficult to utilize the contiguity other than defined page sizes. Our work differs from [11], [12], [13], and [14] in that the system utilizes diverse contiguities by adding the allocation information in a page table.

C. IOMMU AND MEMORY ALLOCATION

In [15], the input/output MMU (IOMMU) that operates partitioned data tiles is presented. The design in [15] utilizes the shared pages among the accelerators to reduce page-table walk overheads. In [16], the page-table walk coalescing scheme is presented. The design in [16] coalesces page-table walks using neighborhood-aware addresses. In [17], the region-based physical memory management scheme robust to fragmentation in mobile systems is presented. In the anti-fragmentation approach in [17], grouped pages with the same lifetime are stored in the regions. In [18], the throughput-oriented memory allocator for GPU is presented. The allocator in [18] supports concurrent programming and synchronization primitives for multiple threads running in GPU. The allocators in [17] and [18] are based on the legacy buddy algorithm that has size and alignment restrictions, whereas our design supports unaligned ranges. In [19], the unified TLB and the page-table cache are presented. In [19], page-table cache entries are stored with TLB entries to reduce TLB misses. In [20], page-table walk requests are rescheduled to reduce GPU stalls. Unlike [15], [16], [17], [18], [19], and [20], our design coalesces page-table entries to increase the address range that a TLB entry covers.

D. OS SUPPORT

In [21], the OS service that allocates large contiguous chunks (possibly throughout the workload's lifetime) by coalescing scattered physical frames is presented. In [22], to reduce memory fragmentation, the compaction technique that migrates movable pages is presented. In [23], the compressor that requires a single heap pass to compact the entire heap is presented. In [24], concurrent real-time garbage collection algorithms are presented. They provide partial compaction support to deal with fragmentation issues. Our TLB coalescing scheme can benefit from the defragmentation methods in [21], [22], [23], and [24] in that OS additionally can exploit the higher contiguity in memory.

E. CONTIGUOUS MEMORY ALLOCATION

In general, a range concept in an allocator is not a new idea. As an example, the virtual memory allocator (VMA) should support unaligned ranges because a chunk requested by an application can vary [25]. VMA typically uses the binary search tree structure that requires logarithmic time complexity. In this work, we present the range implementation in the physical memory allocator. An implication in physical memory allocation is that a range can be fragmented, whereas a chunk in VMA should be contiguous. In [26] and [27], for physical memory, the contiguous memory allocator (CMA) that supports unaligned ranges is presented. Our work differs from CMA [26], [27] in the following ways. First, CMA is designed only for direct memory access (DMA) devices and their (reserved) memory zone, whereas our work can be generally used in normal system (heap) memory. Second, CMA provides fully contiguous physical memory space as requested by an application, whereas a range in our work can be fragmented. In [28], the memory management scheme that can efficiently rent the contiguous reserved memory from inactive I/O devices is presented. In our system, there is no requirement to reserve memory space for an I/O device.

III. BACKGROUND

The traditional MMU architecture and the TLB coalescing scheme [1], [7] are reviewed. Then the conventional memory allocation [4], [9] and the issues are described.

A. TRADITIONAL MMU ARCHITECTURE

In Fig. 1(a1), an application requests five pages. OS allocates virtual page numbers (VPNs) 4-8 to the application. OS finds free pages in physical memory and allocates them in block level. In Fig. 1(a1), two physical blocks (Block$_0$ and Block$_1$) are allocated. In this work, VPN and PPN indicate 4KB-sized page numbers. When a block is allocated, OS configures a page table and stores it in main memory. In a page-table entry (PTE), OS represents the mapping between a VPN and a PPN. Fig. 1(a3) depicts the PTE format in the 32-bit architecture. In Fig. 1(b), when the application runs, the master accesses memory with a virtual address. The master refers to a processor or an I/O device that initiates memory accesses. A VPN that the master requests is called a demanding VPN. To obtain its demanding PPN, MMU conducts a page-table walk, acquires a PTE, and stores the PTE in a TLB entry. Two issues of the traditional design are the following. First, TLB size is usually small and TLB reach (the address space that a TLB covers) is accordingly limited. Second, a TLB entry handles only a single page mapping. Subsequently, certain page-table walks are undesirably required.

B. TLB COALESCING

TLB coalescing is a scheme that maps an address range (instead of a page) into a TLB entry. To implement TLB coalescing, when memory is allocated, OS can add on the contiguity information to page-table entries. Then MMU exploits the information. Fig. 2 depicts the *page-table contiguity ascending descending* (PCAD) scheme [1], [7]. Table 1 shows design parameters [1]. Suppose the page-table walk for VPN 5 acquires PTE$_1$, where the demanding PPN is 17. In PTE$_1$, *Ascend* is 2. This means two ascending PPNs next to PTE$_1$ are contiguous. In PTE$_1$, *Descend* is 1. This means

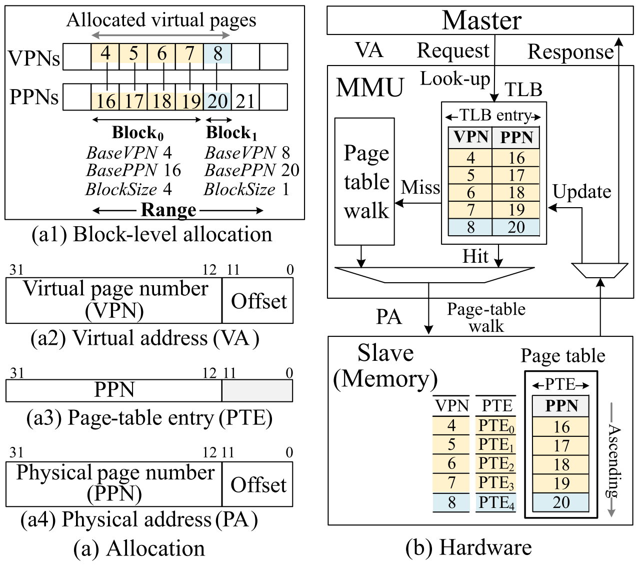

Allocated virtual pages
VPNs 4 5 6 7 8
PPNs 16 17 18 19 20 21
Block0 BaseVPN 4 BasePPN 16 BlockSize 4
Block1 BaseVPN 8 BasePPN 20 BlockSize 1
Range
(a1) Block-level allocation
31 12 11 0
Virtual page number (VPN) | Offset
(a2) Virtual address (VA)
PPN
(a3) Page-table entry (PTE)
Physical page number (PPN) | Offset
(a4) Physical address (PA)
(a) Allocation
Master
VA Request Response
MMU Look-up TLB
TLB entry VPN PPN 4 16 5 17 6 18 7 19 8 20
Page table walk
Miss
Update
Hit
PA
Page-table walk
Slave (Memory)
Page table
VPN PTE PTE0 PTE1 PTE2 PTE3 PTE4
PPN 16 17 18 19 20
Ascending
(b) Hardware

FIGURE 1. Traditional MMU architecture [1], [7].

one descending PPN next to PTE$_1$ is contiguous. In Fig. 2, the contiguity information refers to *Ascend* and *Descend*. These two values represent the number of contiguous pages in a block. Then, using the formula in Table 1, *BaseVPN* is 4 (= 5 - 1), *BasePPN* is 16 (= 17 - 1), and *BlockSize* is 4 (= 2 + 1 + 1). The contiguity information represents entire Block$_0$ and it is coalesced into a single TLB entry. Later when the master accesses any VPN in Block$_0$ in any order, the TLB entry can serve address translation for the entire block. In Fig. 2, only two TLB entries are required and page-table walks can be accordingly reduced. For comparison, the traditional design requires five TLB entries. Therefore, TLB coalescing can improve TLB utilization and reduce page-table walks. It is noted that a block size in a TLB entry is not necessary to be power-of-2 pages.

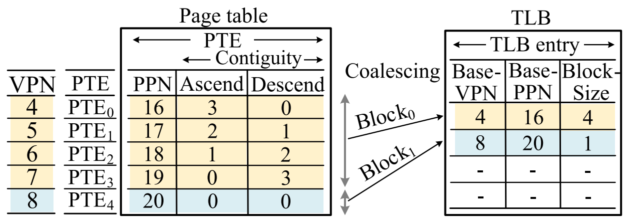


FIGURE 2. The PCAD TLB coalescing scheme [1], [7].

C. PHYSICAL MEMORY ALLOCATION

In this section, the legacy allocator is described. We consider the binary buddy system (BBS) that provides fast allocation [9], [29]. Three rules to allocate a block are the following:

1) A block size (*BlockSize*) should be power-of-2 pages.
2) The address of a block should be aligned with a block size. This means the starting PPN of a block should be *aligned* with a block size. In other words, *BasePPN* should be multiple of *BlockSize*.

TABLE 1. Main design parameters [1].

Parameters	Description	Unit
Ascend	The number of ascending contiguous pages next to the demanding PPN	Page
Descend	The number of descending contiguous pages next to the demanding PPN	Page
BaseVPN	The first VPN in a block = Demanding VPN $-$ *Descend*	
BasePPN	The first PPN in a block = Demanding PPN $-$ *Descend*	
BlockSize	The number of contiguous pages in a block = *Ascend* $+$ *Descend* $+$ 1	Page

3) A block is split into and merged from two buddy blocks of identical size.

In this work, *size* denotes the number of 4KB pages. BBS manages a freelist per block size. The freelist is the data structure to manage dynamic memory allocation, typically implemented in the associative array of linked lists. An array index i is called an order. Freelist[i] contains a list of block nodes of size 2^i. BBS finds the largest block that is smaller than or equal to the requested size. If a block of the desired size 2^i is not available (or freelist[i] is empty), BBS splits a large block into two buddies with identical sizes, until BBS finds a free block of the desired size. Then BBS allocates this block and deletes it in the freelist. Later when OS reclaims this block, BBS scans the freelist to find its buddy. A buddy is easily identified by the bit-wise *XOR* operation between *BasePPN* and *BlockSize*. If the buddy is found, BBS efficiently merges those two blocks and adds the merged block in the freelist. In the typical BBS implementation, freelist nodes are ordered by PPNs. Fig. 3 depicts an example, where the requested size is five pages.

1) Initially, a range of six free contiguous physical pages is available. In this work, the *range* is defined by any contiguous pages. A single range is treated as a set of power-of-2 sized distinct blocks. In Fig. 3, the range is treated as two blocks of sizes four and two. These are represented in freelist[1] and freelist[2]. The node in freelist[2] indicates that *BasePPN* is 16 and *BlockSize* is 4.
2) In Step 1, BBS finds a free block of four pages. As the desired block is found in freelist[2], BBS allocates Block_0. Then BBS finds a block of the remaining one page.
3) In Step 2, BBS splits the block in freelist[1] into two blocks of identical size. These two blocks are buddies and are represented in freelist[0].
4) In Step 3, BBS picks the head node in freelist[0] and allocates Block_1.

In this way, two blocks are efficiently allocated to the application. The algorithm complexity to allocate a block is $O(1)$ since only a head node in the freelist is accessed. However, a main issue is that it is difficult to represent the contiguity other than power-of-2. Accordingly, the system usually handles a contiguous range as multiple fragmented blocks, which degrades memory utilization.

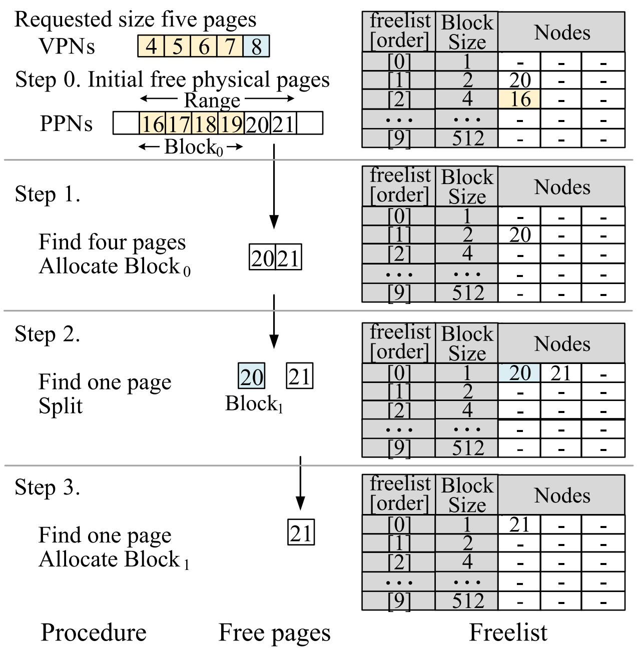


FIGURE 3. Allocation in binary buddy system (BBS).

IV. PROPOSED DESIGN

A. OVERVIEW

We present the range supports in the architecture-specific allocator and its utilization in TLB coalescing. The aim is to fully exploit the advantages of TLB coalescing in architecture and contiguity in memory. The design goal is to alleviate the size and the alignment restrictions without complex hardware and software overheads. Our general approach is to customize the allocation algorithm to better exploit the page sizes defined in the underlying hardware architecture. To do this, we relate the freelist orders to the page sizes and make block sizes flexible. The main novel features of the page buddy system (PBS) are:

1) PBS inherently maintains physical memory in range level.
2) PBS maintains the freelists for page sizes.
3) Using PBS, the TLB coalescing scheme can support unaligned large block.

B. PAGE BUDDY SYSTEM

To implement PBS, we modify the legacy buddy algorithm [4], [9]. Fig. 4 depicts the algorithm. Similar to BBS, PBS finds a free block enough to satisfy the requested size. To do this, PBS checks whether a request is satisfied from an order in the freelist. If a block of the desired size is not available, PBS splits a block in the higher order into 2 blocks, until PBS finds a free block of the desired size. However, in Fig. 4, PBS differs from BBS in the following ways:

1) *Line 4:* PBS maintains the orders for the page sizes.
2) *Line 10:* A block can be split into different sizes.
3) *Lines 12-14:* PBS splits a head block if its block size is larger than the desired size.

4) *Lines 15 and 22:* A block size is not necessary to be power-of-2.
5) *Line 20:* PBS adds the block size information in PTEs.

The parameter *Granularity* in line 4 denotes the distance between adjacent orders. *Granularity* is a configurable parameter and it varies with the targeted architecture. As an example, in ARM v7 architecture, page sizes are 4KB (order 0), 64KB (order 4), 1MB (order 8), and 16MB (order 12). Then *Granularity* is 4. In ARM v8 architecture with a 4KB granule, page sizes are 4KB (order 0), 2MB (order 9), and 1GB (order 18). Then *Granularity* is 9. If *Granularity* is configured by 1 and the block size is 2^{order} pages, PBS is compatible with BBS.

```
Page buddy allocation algorithm
 Inputs: (1) requested size, (2) freelist, (3) Granularity
 Output: Page table
 /* freelist[i] is a list of blocks */
 /* MAX_ORDER is the maximum order of freelists. */
1. while (requested size > 0) do
2.     /* Allocate a block */
3.     High order has block = no
4.     for (i = MAX_ORDER - 1; i >= 0; i -= Granularity) do
5.         if ( (freelist[i] > 0) or (High order has block == yes) ) do
6.             High order has block = yes
7.             if (2^i <= requested size) do
8.                 if (freelist[i] has no block) do
9.                     Pick a block from higher-order freelist
10.                    Split the block into a block of
                       BlockSize and the other block of
                       remaining size.
11.                end if
12.                if (block size of the head block at freelist[i]
                      is larger than requested size) do
13.                    Split the head block into a block of
                       BlockSize and the other block of
                       remaining size
14.                end if
15.                Allocate a block of BlockSize
16.            end if
17.            /* Set page table */
18.            for all pages of the allocated block do
19.                Construct and set PTE
20.                Add contiguity information in the PTE
21.            end for
22.            requested size -= BlockSize
23.            break
24.        end if
25.    end for
26. end while
```

FIGURE 4. **Allocation algorithm in page buddy system (PBS).**

C. FREELIST

Similar to BBS, PBS accesses freelist head nodes to allocate a block. A freelist node is ordered by PPNs similar to BBS. Then PBS maintains the algorithm efficiency of BBS. The PBS freelist differs from BBS in the following ways:

1) A freelist node represents a range with various sizes.
2) A node has a tuple (a base PPN, a block size).
3) A base PPN is not necessary to be aligned with a block size.

When a block is deallocated, PBS examines any ascending and descending buddies. If buddies are found, PBS merges them into the larger block. PBS has the following advantages over BBS:

1) The number of freelist orders can be reduced.
2) When there are certain contiguities in memory, PBS can reduce the number of nodes in the freelist. Then, PBS can reduce the freelist search space, the footprint of an allocator, and fragmentation. As a TLB entry can represent the larger block size, TLB hardware resources can be better utilized.

Fig. 5 depicts an example. Freelist[0] covers 4KB pages. A node in freelist[0] contains the block sizes of 1 to 15. Freelist[4] covers 64KB pages. A node in freelist[4] contains the block size of the multiple of 16. Freelist[8] covers 1MB pages. A node in freelist[8] contains the block size of the multiple of 256.

1) Initially, the range of six free contiguous pages is available. This is represented by the tuple (16, 6) in freelist[0]. The node indicates *BasePPN* is 16 and *BlockSize* is 6.
2) In Step 1, PBS finds a free block of the desired five pages. PBS splits the block into two blocks of different sizes. These blocks are represented in two nodes, (16, 5) and (21, 1) in freelist[0].
3) In Step 2, PBS picks the head node in freelist[0] and allocates Block_0.

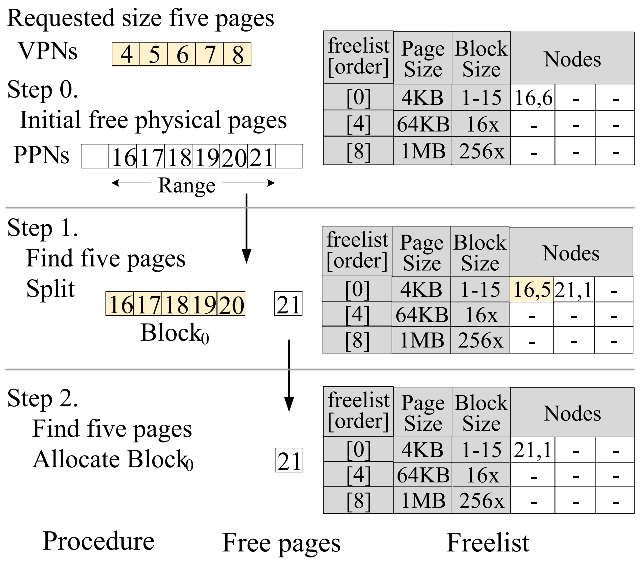
As a result, a single block is allocated in two steps. For comparison, BBS allocates two blocks in three steps. The freelist node can be implemented in an integer-type variable. In practice, a tuple (a base PPN, a block size) can fit in a single integer. Therefore, the memory footprint overhead to implement the freelist node can be avoided.

D. SUPPORT FOR LARGE-SIZED BLOCKS

Traditionally, to use large page sizes, both *BaseVPN* and *BasePPN* should be aligned with *BlockSize* [31]. This restriction makes the system difficult to use large page sizes. In PBS, the address alignment restriction is significantly relaxed. Fig. 6 depicts an example to allocate 1MB or 256 pages. Suppose free pages of PPNs 1 to 256 are available. Fig. 6(a) depicts the BBS freelist where 9 nodes are required. In the traditional hardware, up to 256 page-table walks and 256 TLB entries can be required. If TLB coalescing [1] is supported, 9 page-table walks can be required. Fig. 6(b) depicts the PBS freelist for ARM v7 where three nodes are required. Fig. 6(c) depicts the PBS freelist for ARM v8 where a single node is required.

E. TLB COALESCING USING PBS

The presented allocator is highly orthogonal to underlying hardware and TLB coalescing. However, the system leverages the TLB coalescing logic that the hardware accommodates. The TLB coalescing scheme can benefit from the flexibility of PBS. Fig. 7 depicts an example using the PCAD TLB coalescing [1] and PBS. Suppose the page-table walk

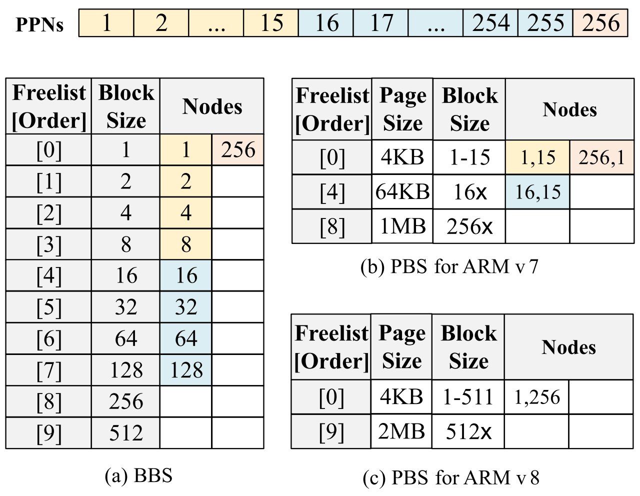


freelist [order]	Page Size	Block Size	Nodes		
[0]	4KB	1-15	16,6	-	-
[4]	64KB	16x	-	-	-
[8]	1MB	256x	-	-	-

freelist [order]	Page Size	Block Size	Nodes		
[0]	4KB	1-15	16,5	21,1	-
[4]	64KB	16x	-	-	-
[8]	1MB	256x	-	-	-

freelist [order]	Page Size	Block Size	Nodes		
[0]	4KB	1-15	21,1	-	-
[4]	64KB	16x	-	-	-
[8]	1MB	256x	-	-	-

FIGURE 5. **Allocation in PBS for ARM v7 architecture.**


PPNs 1 2 ... 15 16 17 ... 254 255 256

Freelist [Order]	Block Size	Nodes	
[0]	1	1	256
[1]	2	2	
[2]	4	4	
[3]	8	8	
[4]	16	16	
[5]	32	32	
[6]	64	64	
[7]	128	128	
[8]	256		
[9]	512		

(a) BBS

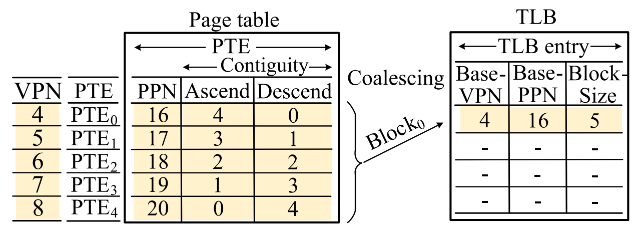
Freelist [Order]	Page Size	Block Size	Nodes	
[0]	4KB	1-15	1,15	256,1
[4]	64KB	16x	16,15	
[8]	1MB	256x		

(b) PBS for ARM v 7

Freelist [Order]	Page Size	Block Size	Nodes	
[0]	4KB	1-511	1,256	
[9]	2MB	512x		

(c) PBS for ARM v 8

FIGURE 6. **Large-sized block allocation. The requested size is 256 pages.**



VPN	PTE	PPN	Ascend	Descend
4	PTE_0	16	4	0
5	PTE_1	17	3	1
6	PTE_2	18	2	2
7	PTE_3	19	1	3
8	PTE_4	20	0	4

Base-VPN	Base-PPN	Block-Size
4	16	5
-	-	-
-	-	-
-	-	-

FIGURE 7. **PCAD TLB coalescing using PBS.**

for the demanding VPN 5 acquires PTE_1. In PTE_1, the demanding PPN is 17, *Ascend* is 3, and *Descend* is 1. Using the formula previously shown in Table 1, *BaseVPN* is 4, *BasePPN* is 16, and *BlockSize* is 5. This represents entire $Block_0$ and is coalesced into a single TLB entry. For comparison, in BBS, two TLB entries are required as previously depicted in Fig. 2. Thus PBS can further improve TLB utilization and accordingly reduce page-table walks in hardware.

F. CASE STUDY

Fig. 8 depicts various mapping examples, where 4 pages are requested and 8 pages are available. An arrow indicates a block and a circle indicates a block number. Table 2 shows the analysis.

1) In the case of (a), two ranges are available. In this optimistic case, a single range represents a single block. Then both BBS and PBS allocate two blocks ⓪-①.
2) In the case of (b), there are two ranges. BBS treats a range as fragmented blocks. Accordingly, BBS allocates two blocks ① and ④. On the other hand, PBS allocates one block ⓪.
3) In the case of (c), there are four ranges. BBS allocates four blocks ⓪-③. On the other hand, PBS allocates two blocks ⓪-①.
4) In the case of (d), there are three ranges. BBS allocates two blocks ① and ③. On the other hand, PBS allocates one block ⓪.

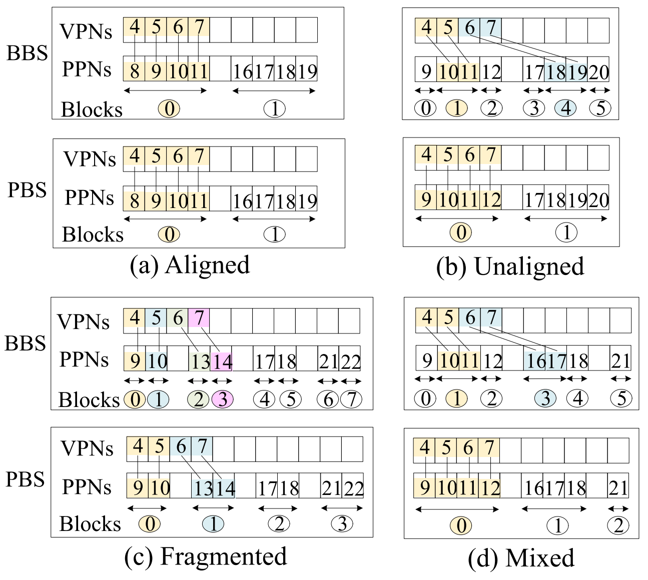


FIGURE 8. **Case study. The requested size is four pages.**

In this way, PBS can reduce the number of freelist nodes, reduce fragmentation, and improve memory utilization. In hardware, when combined with TLB coalescing, PBS can improve TLB reach and performance.

TABLE 2. **Case study for Fig. 8. Lower is better.**

Case	Allocator				Hardware		
	Number of freelist nodes		Number of mapped blocks		Number of TLB entries (Number of PTWs)		
	BBS	PBS	BBS	PBS	BBS + Trad.	BBS + PCAD	PBS + PCAD
(a)	2	2	1	1	4	1	1
(b)	6	2	2	1	4	2	1
(c)	8	4	4	2	4	4	2
(d)	6	3	2	1	4	2	1

G. DEMAND PAGING AND EAGER PAGING

The presented approach supports both eager paging and demand paging [4], [30]. In the eager-paging mode, memory can be allocated in its entirety before an application starts execution [4]. In this mode, the requested size can be the chunk size as much as specified by the application. This mode is often employed in the special-purpose OS and DMA devices. The standard BBS supports aligned block allocation, whereas PBS supports both unaligned and aligned block allocation. PBS can be further beneficial when combined with TLB coalescing. In this paper, we mainly focus on the eager-paging mode. On the other hand, in the demand-paging mode, the requested size is fixed at one. Memory is allocated in the single-page granularity which is typically used by default in the general-purpose system. Demand paging is utilized to enhance memory allocation efficiency and increase the level of multiprogramming in OS [30]. PBS is backward compatible with BBS as PBS inherently supports page-level allocation.

In this paper, we primarily target a DMA I/O device that runs high-bandwidth (streaming image processing) applications in an embedded system. PBS can be implemented together with TLB coalescing as an add-on feature to the legacy system. A potential use case is to utilize BBS for general-purpose applications running on the CPU and utilize PBS for special-purpose workloads executed by the DMA I/O devices. In this way, the presented scheme can coexist with the legacy system. It is noted that PBS maintains the functionality and the efficiency of BBS. Accordingly, the presented PBS-based system allows virtual memory allocation in the page or the superpage level.

V. EXPERIMENTAL RESULTS

A. EVALUATION METHOD

Fig. 9 depicts the system organization. We use the simulation environment of [1]. Table 3 shows the configuration. The virtual and physical addresses are 32 bits wide. The ARM v7 address translation architecture [31] is used. The interface of a component operates with AXI bus protocol [31]. The camera, the 2D data accelerator, and the display controller are DMA masters to which IOMMUs are connected. The PBS-based system uses the PCAD TLB coalescing scheme [1] previously described in Sections III-B and IV-E. In eager paging, all pages within a block can be coalesced to a single TLB entry. In demand paging, the pages within a block need to be split to multiple TLB entries potentially due to the limitation of the PCAD TLB coalescing scheme. A page size is regular 4KB. We implemented the allocation algorithms in C/C++ and integrated them in the system. In BBS, MAX_ORDER is 9, which means 9 freelists are implemented. In PBS, 3 freelists for orders 0, 4, and 8 are implemented. Our evaluation primarily focuses on assessing the performance of the eager-paging mode, specifically for high-bandwidth applications executed in DMA devices with IOMMUs. In the context of a special-purpose embedded system, these applications are sensitive to frequent page faults, which can significantly impact their performance. Therefore, our evaluation aims to understand and analyze the behavior and efficiency of the eager-paging mode in such scenarios.

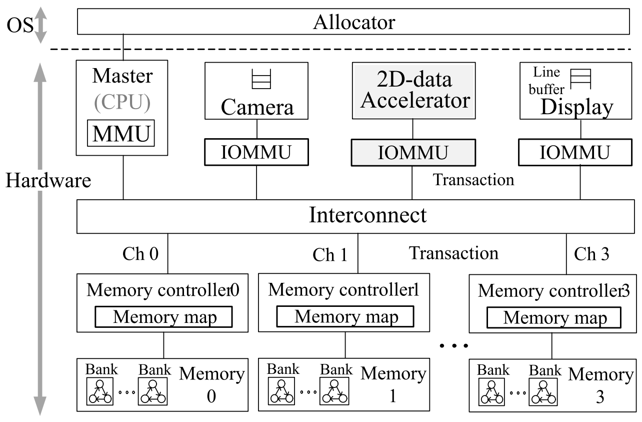


FIGURE 9. System organization.

TABLE 3. System configuration.

Components	Item	Configuration
IOMMU	TLB	32 entries, Fully associative cache
	Replacement	Least Recently Used (LRU)
Interconnect	Data width	128 bits
	Arbitration	Round-robin
	Transaction size	64 bytes
	Multiple outstanding	Max. 16
Memory Controller	Mapping	Row, Bank, Col, Channel, Col
	Request queue	16 entries
Memory (DRAM)	Model	DDR3-800
	Timing	t_{CL}-t_{RCD}-t_{RP} = 5 - 5 - 5
	Channels	4
	Scheduling	Bank-hit first
	Banks	4

Table 4 shows workloads and the associated operations. In this work, we focus on workloads that are commonly found in modern mobile devices. These workloads run in DMA devices, require high throughput, and often utilize eager paging. We specifically select these use cases to evaluate the effectiveness of the presented add-on feature, which involves multi-page allocation combined with TLB coalescing, in enhancing the performance of DMA devices. We categorize the workloads into two types. Category-I workloads access memory in the linear manner and have high address locality. Category-II workloads access memory in the non-linear manner and can have low address locality. In *rotated preview*, the camera captures an image in the raster-scan order and conducts the rotation. Then the display controller reads the image in the raster-scan order and displays the rotated image [1]. In *rotated display*, the display controller reads the rotated image and displays the image. In *matrix multiplication*, the accelerator reads a matrix in the horizontal direction and the other matrix in the block-level vertical direction. Then the accelerator writes the matrix in the horizontal direction. In *matrix transpose*, matrix rows and columns are interchanged. The accelerator reads the matrix

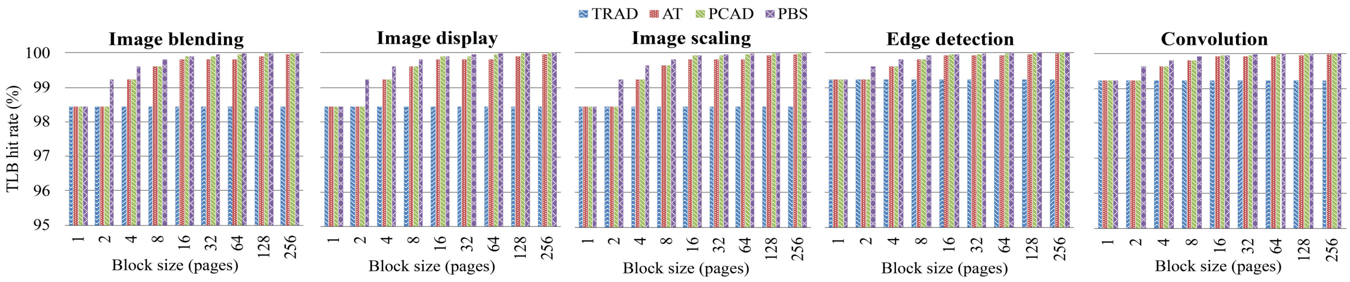


FIGURE 10. MMU performance of category-I workloads.

TABLE 4. Workloads.

	Workloads	Component	Operation	Access
I	Camera preview	Camera	Write	Raster scan
		Display	Read	Raster scan
	Image scaling x1.5	Camera	Write	Raster scan
		Scaler	Read, Write	Raster scan
		Display	Read	Raster scan
	Image blending	Blender	Read, Read, Write	Raster scan
	Edge detection	Accelerator	Read	Block hor.,
			Write	Hor.
	Convolution	Accelerator	Read	Block hor.,
			Write	Hor.
II	Rotated preview	Camera	Write	Vertical
		Display	Read	Raster scan
	Rotated display	Display	Read	Vertical
	Matrix multiplication	Accelerator	Read	Hor.
			Read	Block ver.
			Write	Hor.
	Matrix transpose	Accelerator	Read	Block hor.,
			Write	Block ver.

in the block-level horizontal direction. Then the accelerator writes the matrix in the block-level vertical direction. We implemented traffic generators to represent these memory access behaviors. An image size is 1280 x 720 pixels in which a pixel is 4-byte sized RGB format. In this case, the requested size is 900 pages. A matrix size is 600 x 600 elements in which an element is 4-byte sized float type.

To evaluate MMU and system performance, two experiments are conducted. Then OS allocation time and freelist size are evaluated. We consider the traditional design (denoted by TRAD), PCAD TLB coalescing with BBS [1] (PCAD), and AT TLB coalescing with BBS [5] (AT) as references. Our presented scheme with the PCAD TLB coalescing is denoted by PBS. Figs. 11-21 depict the performance results versus various fragmentation levels. In Figs. 11-21, the x-axis denotes free block sizes that indicate the fragmentation levels in the physical memory. When the block size is 1, all free blocks are one page sized and fully fragmented. When a block size increases, the fragmentation decreases and the contiguity increases. When the block size is 256, huge-sized free blocks are available. Physical page numbers of the blocks are randomly generated.

B. MMU PERFORMANCE

To evaluate the MMU performance, we measure TLB hit rates. Fig. 10 depicts the results of category-I workloads, where TLB hit rates of all designs are higher than 98% and are sufficiently high. In this case, though PBS performs better than references, the improvement is less than 2% and is insignificant. This is because the raster-scan traffic patterns have high address localities. Figs. 11-14 depict the results of category-II workloads, where traffics have low address localities. The results are summarized by the following:

1) Full fragmentation (block size 1): All designs have the same TLB hit rates.
2) Small and medium block sizes (2 to 64): PBS performs significantly better than the reference. PBS is up to 22% better than PCAD and up to 30% better than AT. This is because PBS reduces the size and the alignment restrictions.
3) Large block sizes (128 to 256): PBS is up to 2x better than TRAD. PBS is comparable to PCAD and AT. This is because large-sized blocks are allocated.

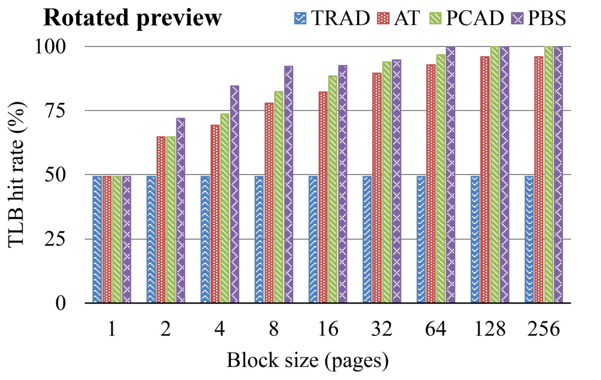


FIGURE 11. MMU performance of rotated preview.

C. SYSTEM PERFORMANCE

To evaluate the system performance, we measure execution cycles to run the workloads. In category-I workloads, the performance differences between all designs are less than 2% and are insignificant. This is because the TLB hit rates of all designs are sufficiently high as depicted in Fig. 10. In category-II workloads, PBS significantly improves performance. Figs. 15-18 depict the results summarized by the following:

1) Full fragmentation (block size 1): The PBS performance is same as TRAD and PCAD. PBS is up to 52% better than AT. This is because AT maintains two types (regular and anchor) of page-table walks.

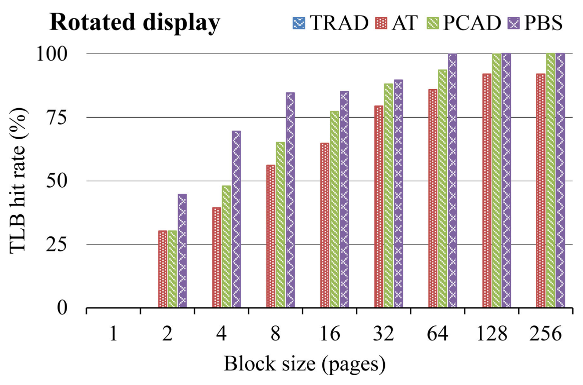


FIGURE 12. **MMU performance of rotated display.**

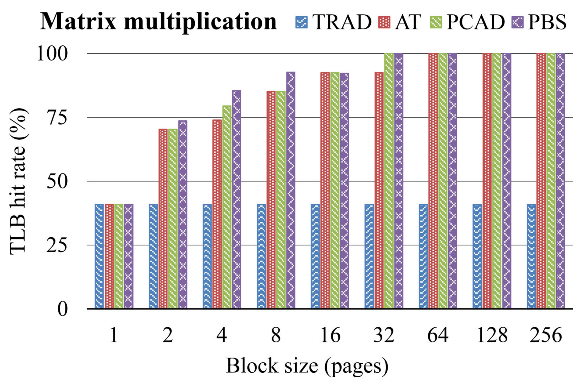


FIGURE 13. **MMU performance of matrix multiplication.**

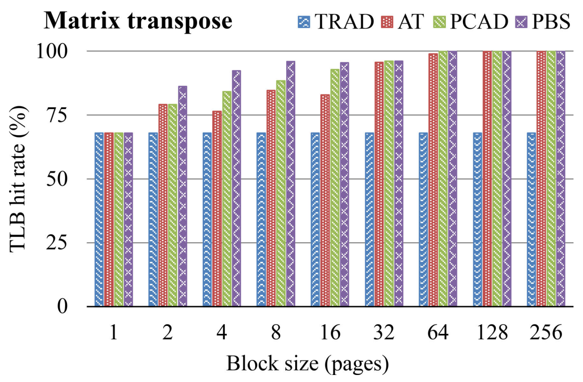


FIGURE 14. **MMU performance of matrix transpose.**

2) Small and medium block sizes (2 to 64): PBS performs significantly better than references. PBS is up to 30% better than PCAD, up to 67% better than AT, and up to 78% better than TRAD. In the medium block sizes, AT can coalesce multiple PTEs and reduce page-table walks.
3) Large block sizes (128 to 256): PBS performance is close to PCAD. PBS is up to 45% better than AT and up to 79% better than TRAD.

D. ALLOCATION PERFORMANCE

To evaluate the allocation performance, we measure the time to allocate 900 pages. We consider BBS as a reference.

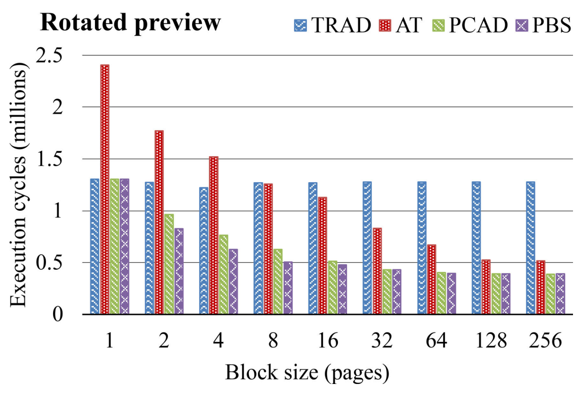


FIGURE 15. **System performance of rotated preview.**

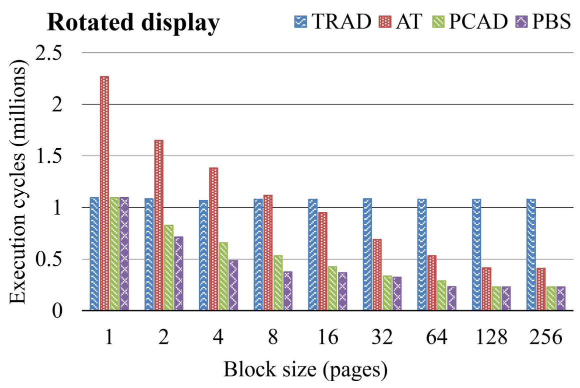


FIGURE 16. **System performance of rotated display.**

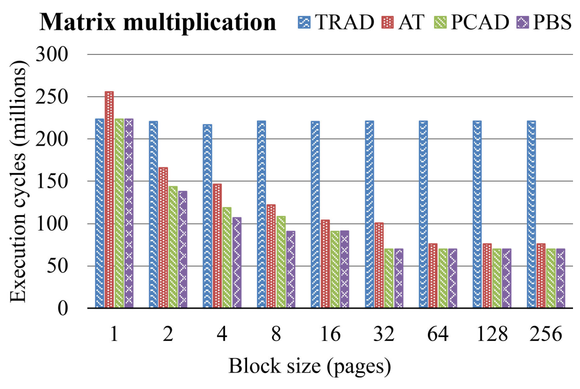


FIGURE 17. **System performance of matrix multiplication.**

1) EAGER PAGING

Fig. 19 depicts the results for eager-paged virtual memory. When a block size is small (1 to 16), PBS performs up to 25% better than BBS. This is because PBS traverses less orders and can allocate larger blocks. When the block size increases, PBS often performs worse than BBS. When the block size is 256, PBS is up to 9% worse than BBS. This is mainly because of implementation overheads to maintain various block sizes, ascending and descending buddies, and the contiguity information. Additionally, when large-sized blocks are available, both PBS and BBS allocate the large blocks. In Fig. 19, PBS is on average 8% better than BBS.

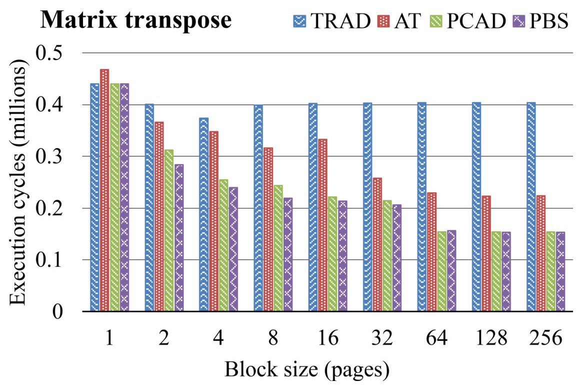


FIGURE 18. System performance of matrix transpose.

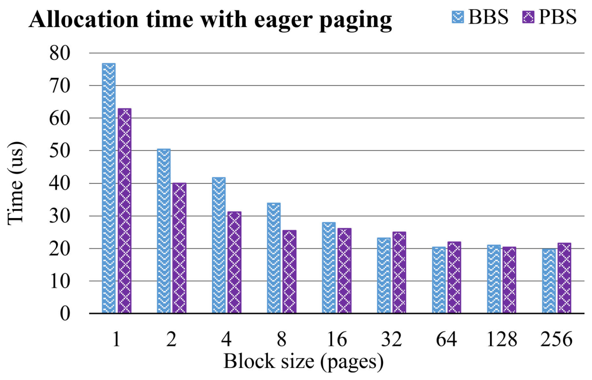


FIGURE 19. Allocation performance in eager paging.

2) DEMAND PAGING

To assess the relative allocation performance in the demand-paging mode, we fixed the request size at one. Then the memory is allocated in the single-page granularity, which is typical for demand paging. Fig. 20 depicts the standalone allocation time. As a result, PBS is on average 17% better than BBS. This is because BBS scans up to 9 orders to allocate a single page, whereas PBS scans up to 3 orders. In the system simulation environment, a secondary storage model and a page-fault handling routine are not included.

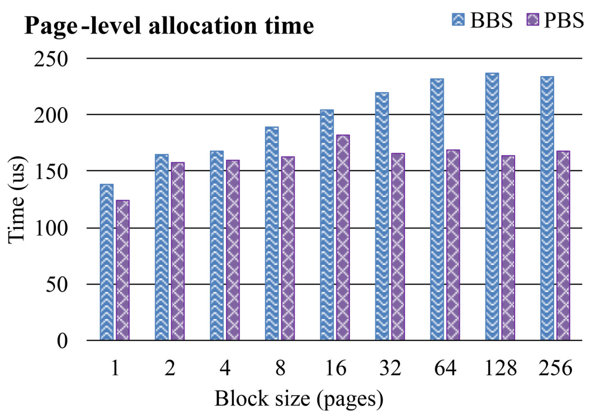


FIGURE 20. Relative allocation performance in demand paging.

Accordingly, the latencies associated with page swapping and the page-fault handler are not accounted for in Fig. 20. We leave the performance evaluation in the demand-paging mode using those models for future work.

E. FREELIST SIZE

To evaluate allocator size and memory utilization, we measure the number of freelist nodes. There are 1024 physical pages in total. Fig. 21 depicts the results. When the block size is 1 (or all blocks are fully fragmented), PBS and BBS has the identical number of freelist nodes. However, when a block size (or contiguity) increases, PBS significantly reduces the freelist nodes. In Fig. 21, PBS reduces the freelist nodes by 46%. This is because PBS reduces the size and the alignment restrictions. This suggests that PBS can reduce allocator size, reduce fragmentation, and improve memory utilization.

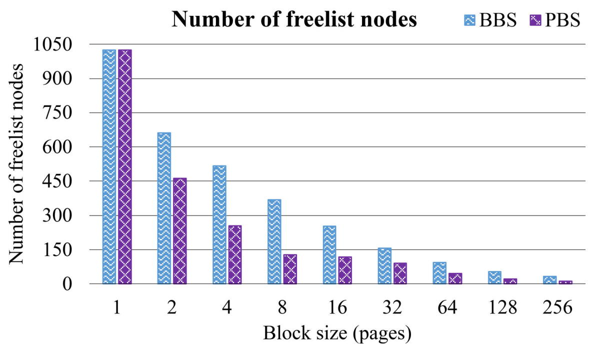


FIGURE 21. Freelist size.

VI. CONCLUSION

The flexible memory allocator customized for page sizes defined in the architecture is presented. The presented allocator can manage various-sized and unaligned blocks with insignificant overheads. The main advantage is the improved memory utilization. In the context of eager-paged virtual memory, when TLB coalescing is used and the traffic pattern exhibits low address locality, the presented scheme can provide better TLB utilization, reduced page-table walks, and improved performance. There are additional implementation overheads to maintain various block sizes, ascending and descending buddies, and the contiguity information. These implementation overheads can be traded for improved resource utilization, flexibility, and performance.

ACKNOWLEDGMENT

The authors would like to thank Kyong Ho Cho in Samsung Electronics for helpful discussions and also would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] J. Y. Hur, "Contiguity representation in page table for memory management units," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 1, pp. 147–158, Jan. 2019.

[2] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced large-reach TLBs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Vancouver, BC, Canada, Dec. 2012, pp. 258–269.

[3] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh, "Increasing TLB reach by exploiting clustering in page translations," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Orlando, FL, USA, Feb. 2014, pp. 558–567.

[4] 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 *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Portland, OR, USA, Jun. 2015, pp. 66–78.

[5] C. H. Park, T. Heo, J. Jeong, and J. Huh, "Hybrid TLB coalescing: Improving TLB translation coverage under diverse fragmented memory allocations," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 444–456.

[6] J. Y. Hur, "Block level TLB coalescing for buddy memory allocator," *IEICE Trans. Inf. Syst.*, vol. 102, no. 10, pp. 2043–2046, Oct. 2019.

[7] J. Y. Hur and J. Kong, "Page table compaction for TLB coalescing," *IEEE Access*, vol. 8, pp. 104814–104829, 2020.

[8] X. Wang, H. Liu, X. Liao, H. Jin, and Y. Zhang, "TLB coalescing for multi-grained page migration in hybrid memory systems," *IEEE Access*, vol. 8, pp. 66304–66314, 2020.

[9] K. Knowlton, "A fast storage allocator," *Commun. ACM*, vol. 8, no. 10, pp. 623–625, Oct. 1965.

[10] J. Evans, "A scalable concurrent malloc(3) implementation for FreeBSD," in *Proc. Tech. BSD Conf. (BSDCan)*, Ottawa, ON, Canada, May 2006, pp. 1–14.

[11] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: A GPU memory manager with application-transparent support for multiple page sizes," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Boston, MA, USA, Oct. 2017, pp. 136–150.

[12] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, pp. 89–104, Dec. 2002.

[13] Y. Du, M. Zhou, B. R. Childers, D. Mossé, and R. Melhem, "Supporting superpages in non-contiguous physical memory," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Burlingame, CA, USA, Feb. 2015, pp. 223–234.

[14] G. Cox and A. Bhattacharjee, "Efficient address translation for architectures with multiple page sizes," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Xian, China, Apr. 2017, pp. 435–448.

[15] Y. Hao, Z. Fang, G. Reinman, and J. Cong, "Supporting address translation for accelerator-centric architectures," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2017, pp. 37–48.

[16] S. Shin, M. LeBeane, Y. Solihin, and A. Basu, "Neighborhood-aware address translation for irregular GPU applications," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Fukuoka, Japan, Oct. 2018, pp. 352–363.

[17] S.-H. Kim, S. Kwon, J.-S. Kim, and J. Jeong, "Controlling physical memory fragmentation in mobile systems," in *Proc. Int. Symp. Memory Manage.*, Portland, OR, USA, Jun. 2015, pp. 1–14.

[18] I. Gelado and M. Garland, "Throughput-oriented GPU memory allocation," in *Proc. 24th Symp. Princ. Pract. Parallel Program.*, Washington, DC, USA, Feb. 2019, pp. 27–37.

[19] Z. Ma, Y. Tan, H. Jiang, Z. Yan, D. Liu, X. Chen, Q. Zhuge, E. H. Sha, and C. Wang, "Unified-TP: A unified TLB and page table cache structure for efficient address translation," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Hartford, CT, USA, Oct. 2020, pp. 255–262.

[20] S. Shin, G. Cox, M. Oskin, G. H. Loh, Y. Solihin, A. Bhattacharjee, and A. Basu, "Scheduling page table walks for irregular GPU applications," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Los Angeles, CA, USA, Jun. 2018, pp. 180–192.

[21] Z. Yan, D. Nellans, D. Lustig, and A. Bhattacharjee, "Translation ranger: Operating system support for contiguity-aware TLBs," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Phoenix, AZ, USA, Jun. 2019, pp. 698–710.

[22] M. Gorman and A. Whitcroft, "Supporting the allocation of large contiguous regions of memory," in *Proc. Ottawa Linux Symp. (OLS)*, Ottawa, ON, Canada, Jun. 2007, pp. 141–152.

[23] H. Kermany and E. Petrank, "The compressor: Concurrent, incremental, and parallel compaction," in *Proc. 27th ACM SIGPLAN Conf. Program. Language Design Implement.*, Ottawa, ON, Canada, Jun. 2006, pp. 354–363.

[24] F. Pizlo, E. Petrank, and B. Steensgaard, "A study of concurrent real-time garbage collectors," in *Proc. 29th ACM SIGPLAN Conf. Program. Language Design Implement.*, Phoenix, AZ, USA, Jun. 2008, pp. 33–44.

[25] D. Bovet and M. Cesati, *Understanding the Linux Kernel*, 3rd ed. Sebastopol, CA, USA: O'Reilly Associates, 2005.

[26] M. Nazarewicz, "Contiguous memory allocator," in *Proc. LinuxCon Eur.* Barcelona, Spain: Linux Foundation, Nov. 2012, pp. 1–46.

[27] S. Park, M. Kim, and H. Y. Yeom, "GCMA: Guaranteed contiguous memory allocator," *IEEE Trans. Comput.*, vol. 68, no. 3, pp. 390–401, Mar. 2019.

[28] J. Jeong, H. Kim, J. Hwang, J. Lee, and S. Maeng, "Rigorous rental memory management for embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1, pp. 1–21, Mar. 2013.

[29] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," in *Proc. Int. Workshop Memory Manage. (IWMM)*, Kinross, U.K., Sep. 1995, pp. 1–78.

[30] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

[31] ARM Ltd. *ARM Architecture Reference Manual*. Accessed: May 20, 2023. [Online]. Available: http://www.arm.com

TRAN DAI DUONG received the B.S. degree in computer engineering from the University of Information Technology, Vietnam National University Ho Chi Minh City, Vietnam, in 2016. He is currently pursuing the integrated master's and Ph.D. degree with the Department of Electronic Engineering, Jeju National University, South Korea. From 2015 to 2017, he was a Hardware Design Engineer with Renesas Design Vietnam Company Ltd., Vietnam. His research interests include computer architecture and digital logic design.

JAE YOUNG HUR (Member, IEEE) received the B.S. degree in electronics engineering from Cheju National University, South Korea, in 1995, the M.S. degree in electronics engineering from Sogang University, South Korea, in 1998, the M.S. degree in electronics engineering from the Munich University of Technology, Germany, in 2002, and the Ph.D. degree in computer engineering from the Delft University of Technology, The Netherlands, in 2011. He was an Engineer, from 1999 to 2000, and a Senior Engineer, from 2008 to 2016, with the Semiconductor Division, Samsung Electronics Ltd., South Korea. Currently, he is an Assistant Professor with the Department of Electronic Engineering, Jeju National University, South Korea. His research interests include embedded system architecture, VLSI design, and reconfigurable computing.