An Empirical Evaluation of PTE Coalescing

Eliot H. Solomon
Rice University
Houston, Texas, USA
ehs3@rice.edu

Yufeng Zhou
Rice University
Houston, Texas, USA
yufengz@rice.edu

Alan L. Cox
Rice University
Houston, Texas, USA
alc@rice.edu

ABSTRACT

Superpages (also known as huge pages) are an effective technique for reducing the latency of virtual-to-physical address translation on modern processors. However, the large size of the 2 MB and 1 GB superpages supported by x86-64 processors continues to present a challenge to the operating system's ability to form superpages, given the mandatory contiguity, alignment, and attribute requirements of a superpage. Recent work proposes medium-sized superpages as a potential solution, by allowing the creation of smaller superpages where 2 MB and larger superpages have not formed or will not be possible to form. Notably, AMD processors starting with the Zen microarchitecture have offered a "PTE Coalescing" feature where the hardware opportunistically and transparently creates, from underlying consecutive and aligned 4 KB mappings in the page table, 16 KB or 32 KB mappings to be cached in the TLB. On the surface, this feature requires no modifications to the operating system or the compiler toolchain, exploiting only coincidental contiguity and alignment. Nonetheless, there are ways that either the operating system or the toolchain can be made coalescing-aware and hence make better use of PTE Coalescing. This paper first investigates undocumented aspects of PTE Coalescing, and then evaluates some operating system and toolchain optimizations which explicitly take advantage of it. We find that an operating system that is coalescing-friendly reduces L1 ITLB misses by 50%-80% compared to an operating system that is coalescing-unaware. For a Clang compilation workload, a coalescing-friendly operating system coupled with PTE Coalescing all but eliminates L2 ITLB misses. Last but not least, we evaluate the impact of granularity (16 KB vs 32 KB) on the effectiveness of PTE Coalescing. We find that reducing the coalescing granularity from 32 KB to 16 KB leads to a 1.3x-20.5x reduction in 4 KB L2 DTLB misses in a wide variety of workloads.

CCS CONCEPTS

• **Computer systems organization**; • **Software and its engineering → Virtual memory**;

KEYWORDS

virtual memory, virtual-to-physical address translation, translation look-aside buffer, superpages, page table entry coalescing

1 INTRODUCTION

Superpages (also known as huge pages) are a combined hardware and software technique used to reduce the latency of virtual-to-physical address translation on modern processors. By enabling a single entry in the processor's translation look-aside buffer (TLB) to map a large block of physical memory that consists of many base-sized pages, they expand the TLB's reach, resulting in higher TLB hit rates and improved performance for many applications.

On x86-64 CPUs, which have a base page size of 4 KB, the most common superpage sizes are 2 MB and 1 GB, corresponding to the amount of physical memory mapped by a single level two ("page directory") and level three ("page directory pointer table") page table entry respectively. Consequently, these are the sizes that operating systems like Linux through its Transparent Huge Pages (THP) feature and FreeBSD with its reservation-based approach [30] have focused on supporting [40].

However, the 2 MB and 1 GB superpages supported by x86-64 have a number of limitations. First, their relatively large size constrains the operating system's ability to form superpages for smaller objects without causing excessive physical memory fragmentation [26]. Second, these superpages can only map aligned, physically contiguous regions of memory where each constituent 4 KB page has identical attributes.

Medium-sized superpages offer a potential solution to these problems, and a variety of different strategies for their implementation have been developed. Two broad categories, implicit and explicit, exist. Implicitly created superpages are transparent to the operating system and require no modifications to it. One example of this approach is known as a coalesced TLB. Upon a TLB miss, the memory management unit (MMU) implementing a coalesced TLB examines the page table entries (PTEs) neighboring the entry needed to handle the miss. If it discovers a group of PTEs mapping contiguous physical memory, it may coalesce them into a single TLB entry.

A coalesced TLB known as CoLT was proposed by Pham et al. in 2012 [35]. Two varieties were described, CoLT-SA and CoLT-FA, with the former being targeted at set-associative TLBs and the latter at fully-associative TLBs. CoLT-SA first alters the TLB indexing scheme to ensure that consecutive virtual pages are mapped to the same set, and then augments each TLB entry with multiple valid bits, each corresponding to a base page that could potentially be included in a superpage represented by that entry. CoLT-FA, on the other hand, stores a base virtual address and a number of pages within a separate superpage TLB structure.

The CoLT design offers a number of advantages over the 2 MB and 1 GB superpages typically offered by x86-64 processors. Most notably, neither CoLT approach requires alignment, just physical contiguity. In addition, CoLT-FA allows a variable number of consecutive PTEs to be coalesced into a single entry, and CoLT-SA drops the requirement for them to be consecutive (i.e. "holes" can exist in a superpage). Subsequent work has extended CoLT to exploit so-called "clustered spatial locality" by mapping clusters of virtual pages to clusters of physical pages without even a strict contiguity requirement [34].

In contrast, the second category of medium-sized superpage implementations require explicit operating system support. An example of this category is the ARMv8-A architecture's Contiguous bit [7]. If an ARMv8-A processor is operating with a base page size of 4 KB, when the Contiguous bit is set in every PTE within a group of 16 base page PTEs that together map 64 KB of aligned and physically contiguous memory with identical attributes, the MMU's TLB is permitted to use a single 64 KB TLB entry to cache the mappings defined by those 16 PTEs. It is the responsibility of the operating system to set the Contiguous bit only when PTEs conform to this set of requirements. However, neither THP under Linux nor Superpages under FreeBSD currently utilize the Contiguous bit to create medium-sized superpages automatically.

The "PTE Coalescing" feature offered by AMD's Ryzen[1] processors can be seen as a simplified, real-world implementation of CoLT, bringing support for medium-sized superpages to the masses. A recent AMD Zen 3 optimization manual offers the most detailed publicly available description of the feature: "If a 16-Kbyte aligned block of four consecutive 4-Kbyte pages are also consecutive and 16-Kbyte aligned in physical address space and have identical page attributes, the processor may opportunistically store them in a single TLB entry resulting in increased effective capacity for both L1 and L2 DTLB and ITLB" [3]. Note however that on older Ryzen processors (Zen and Zen+) all eight PTEs contained within the same 64-byte cache line are examined for potential coalescing as a 32 KB mapping upon a TLB miss [2, 4].

However, Ryzen's PTE Coalescing is clearly less general than both CoLT techniques in that it requires alignment and does not allow for any pages within a group of 4 or 8 to be missing from the superpage mapping. Thus, the measurements from the CoLT paper that quantify the physical memory contiguity that arises coincidentally from Linux's use of a buddy allocator for physical memory management do not effectively quantify the potential for PTE Coalescing's effectiveness since those figures ignore alignment.

Furthermore, AMD's public documentation for the Ryzen family of processors provides only limited information about PTE Coalescing, and questions still remain as to the feature's exact behavior. For example, if a contiguous and aligned block of clean read-write PTEs are all accessed, will the resulting coalesced page be read-write or read-only? A read-write mapping can handle a write without further intervention, but gives up the ability for the processor to track dirty pages at a 4 KB granularity.

Although Ryzen's PTE Coalescing can create medium-sized superpages without the direct involvement of the operating system, an operating system that is aware of the requirements for PTE Coalescing can potentially help the MMU create more medium-sized superpage mappings than would otherwise be possible. This is because a superpage-aware operating system can make physical memory allocation decisions that intentionally create the alignment and contiguity required for coalescing, rather than limiting the MMU hardware to exploiting only that which is created coincidentally, as does Linux's buddy allocator. In contrast to THP under Linux, FreeBSD's reservation-based physical memory allocator speculatively reserves aligned and contiguous physical memory for regions within the heap even when they are currently too small to be mapped by a 2 MB superpage. Whether or not such regions ever grow in size and are ever mapped by a 2 MB superpage, in the meantime, the base pages that are allocated from the reservation will have the alignment and contiguity required by PTE Coalescing. In other words, PTE Coalescing and reservation-based allocation have the potential to interact to produce performance benefits greater than either in isolation at zero additional cost.

[1]We use "Ryzen" throughout the paper for simplicity and consistency to refer to AMD processors since the Zen microarchitecture generation. As a microarchitectural feature, PTE Coalescing is available in both the Ryzen and the EPYC product lines, and is not limited to Ryzen.

This paper's contributions are:

- We use a custom microbenchmark to experimentally determine several undocumented aspects of the PTE Coalescing feature's behavior.
- We evaluate how explicitly creating physical contiguity and alignment using a reservation-based allocator impacts the effectiveness of PTE Coalescing.
- We indirectly characterize the physical contiguity and alignment that is coincidentally created by a buddy allocator by contrasting its behavior with that of a reservation-based allocator.
- We evaluate how the region size reduction from 8 to 4 PTEs has impacted the effectiveness of PTE Coalescing.
- We suggest additional ways for the compiler toolchain and operating system to make better use of PTE Coalescing.

The rest of this paper is organized as follows. Section 2 provides background for the sections that follow. Section 3 describes the microbenchmark that we developed for characterizing undocumented details of the PTE Coalescing behavior of Ryzen processors and our observations from running that microbenchmark on Ryzen 2700X (Zen+) and 5900X (Zen 3) processors. Section 4 describes our methodology for evaluating the impact of PTE Coalescing on a variety of applications under Linux and FreeBSD. Section 5 presents the results of our evaluation. Section 6 discusses additional related work. Finally, Section 7 summarizes our conclusions.

2 BACKGROUND

In this section, we first describe some relevant aspects of the support for transparent superpages in Linux and FreeBSD. Both systems focus on 2 MB and 1 GB superpages, and do not, for example, utilize ARMv8-A's Contiguous bit within base page PTEs to create medium-sized superpages automatically on that architecture. We then describe some relevant aspects of the ELF executable file format.

2.1 Linux THP

Linux Transparent Huge Pages (THP) uses two complementary approaches to create superpage mappings for anonymous virtual memory, such as an application's heap. First, upon the first page fault to a virtually aligned superpage-sized region within an application's address space, if the entire region is logically accessible and aligned, contiguous physical memory is available for allocation, the page fault handler allocates physical memory for a superpage, zeroes that physical memory, and maps it as a superpage. Second, when physical memory is heavily fragmented and thus aligned, contiguous physical memory is difficult to allocate at page fault time, a kernel daemon, khugepaged, compacts physical memory to make aligned, contiguous physical memory available and promotes an application's base pages into superpages by first copying the base pages into aligned, contiguous physical memory and then mapping that physical memory as a superpage.

However, Linux does not transparently create superpage mappings for code or data that is demand paged from a regular, disk-based file system. Nonetheless, there are two different workarounds that are commonly employed to map the code and data of an executable file with superpages. As the first workaround, the user copies the executable file to a special huge page file system and runs that copy of the executable [29]. As the second workaround, during initialization, the application first copies its own code and data from their original virtual memory regions that are demand paged from the executable file to superpage-backed anonymous memory, and then the application remaps the superpage-backed memory to the original virtual memory regions for the code and data using the `mremap` system call [28]. This second workaround incurs significant overhead because "copy-and-remap" has to be repeated every time an application is started.

2.2 FreeBSD Superpages

FreeBSD uses a reservation-based allocator [30] to create the physical contiguity and alignment that is needed to transparently offer superpage support to applications. Upon the first page fault to a virtually aligned superpage-sized region, the page fault handler reserves the physically contiguous and aligned memory for a superpage, but it does not yet map the entire superpage. Initially, it only maps the base page that is required to resolve the page fault. Additionally, in contrast to Linux THP, for anonymous virtual memory, such as the application's heap, the page fault handler reserves the physical memory for a 2 MB superpage even if the entire superpage-sized region is not currently logically accessible. In effect, the page fault handler speculates that heap mappings which are not yet large enough to encompass an entire 2 MB superpage will eventually grow. Subsequent page faults to the same region will allocate physical pages from this reservation instead of the underlying buddy allocator.

When a reservation becomes fully populated, i.e., each of its constituent pages has been allocated by a page fault, the page fault handler checks to see if the pages' attributes are identical, and if they are, promotes the base page mappings to a superpage mapping in the page table. If the memory object (file) that contains the new superpage is subsequently mapped into the address space of another process, a fast path within the page fault handler will immediately create another superpage mapping rather than incrementally promoting base pages into a superpage for a second time.

In contrast to Linux THP, FreeBSD supports automatic superpage creation for code and data from any file system, thereby avoiding the overhead of either workaround used on Linux, such as the copy-and-remap approach. In contrast to anonymous memory mappings, mappings backed by a file, such as the text and read-only data sections of an executable file, are not expected to grow. Consequently, reservations are only created for a file mapping when the file exceeds 2 MB in size, and a reservation is only created for the last part of the file if the file is an exact multiple of 2 MB. Also, upon a page fault to a file-backed region, the underlying file system will typically load multiple pages, corresponding to the file system's block size. The page fault handler will map these pages, but unlike the page triggering the fault, these neighboring pages will not have the access bit set in the PTEs that map them.

If a reservation is not fully populated, it may be cheaply broken at any time to reclaim physical memory. FreeBSD attempts to avoid this as much as possible in order to preserve the amount of physically contiguous memory that is available to create superpages. In practice, reservation breaking only occurs under severe memory pressure.

2.3 The ELF Executable File Format

To load the contents of an ELF executable file into the address space of a process, the operating system reads the file's program header table. This table defines a set of memory segments, where each segment consists of adjacent sections within the file, such as code and read-only data, that are to be mapped together within the same region of the address space with the same access permissions. However, the segments are not required to start or end on a page boundary within the file or the address space. Consequently, a physical page caching a portion of the file may contain code and/or data from two different segments, and this page will be mapped twice within the address space. For example, if the physical page contains both code and read/write data, it may be mapped with execute-only permissions as part of one segment and copy-on-write permissions as part of another. Given the page granularity of virtual-to-physical mappings, the data within that page appears alongside the code with execute-only permissions in one segment and the code within that page appears alongside the read/write data but without execute permissions in the other.

This possible sharing of a physical page between two segments necessarily affects the linker's placement of segments within the address space. Consider a segment that ends at offset *off* within a page followed by a segment that begins at offset *off* within the same page. The virtual address that is the end of the first segment must be separated from the virtual address that is the start of the second by a distance that is a multiple of the page size. Otherwise, the same physical page could not be mapped at the end of the first segment and the beginning of the second. In the ELF specification, this restriction on the placement of segments is expressed as follows: the virtual address at which a segment is mapped modulo `maxpagesize` must equal the file position from which the segment is loaded modulo `maxpagesize`.

Consider an ELF executable file that is 2 MB in size, created by the LLVM project's LLD linker, and stored in a regular, disk-based file system under FreeBSD. Upon the first access to the file, which is typically to the ELF header, a reservation will be created to provide the physical memory for caching the file's contents. Recent versions of the LLD linker have a default `maxpagesize` of 4 KB, but this can be changed from the command line[2]. Also by default, executable files created by the LLD linker begin with a read-only data segment that includes the ELF header. This read-only segment is then followed by a code segment that has both read and execute permissions. Unfortunately, with a `maxpagesize` of 4 KB, while the code will be stored in physically contiguous memory from the

[2]In contrast, older versions of the LLD linker and GNU binutils ld linker had a default `maxpagesize` of 2 MB.

reservation, it may not be properly aligned for PTE Coalescing to occur.

For example, suppose that the read-only segment does not end within the file at an offset that is a multiple of 4 KB. Such an executable is illustrated in Figure 1. By default, the code segment will begin immediately after the read-only data segment in the file, and so it will not begin at an offset that is a multiple of 4 KB. The 4 KB physical page in which the boundary between the read-only data and code segments lies (page 6 in the figure) will be mapped twice, back-to-back in the virtual address space. Because of this, the virtual addresses of the second mapping as well as the subsequent mappings for the rest of the code segment will be in effect "phase shifted" by 4 KB relative to the physical addresses that they map. In other words, the virtual and physical addresses will not be congruent modulo 16 KB or 32 KB, which will make PTE Coalescing impossible within the code and subsequent segments. In the figure, when `maxpagesize` is set to 4 KB, the bracketed virtual pages are not aligned to a 16 KB region of the process's address space, so they cannot be coalesced into a 16 KB superpage even though they correspond to a physical superpage that is aligned and contiguous.

Manually setting `maxpagesize` to any power-of-two greater than or equal to the PTE Coalescing size will address this issue. In Figure 1, when the `maxpagesize` is increased to 16 KB, a 16 KB gap is left between corresponding locations in the twice-mapped physical page. This pushes the bracketed virtual pages into alignment, allowing PTE Coalescing to create a single TLB entry that maps Virtual Superpage 3 to Physical Superpage 2.

Changing the `maxpagesize` from its default of 4 KB to a larger size like 2 MB has little to no impact on the size of the executable files that result from the linking process. To verify this, we used `diff` to compare the output of `ls -l` on `/bin` and `/usr/bin` from a FreeBSD system compiled with a `maxpagesize` of 4 KB and an otherwise identical system with `maxpagesize` set to 2 MB. Not a single line changed between the two systems, indicating that no significant file size impact exists. This is because the ELF format allows the specified virtual mappings of each segment to be changed without adding any additional padding to the file.

3 PTE COALESCING

As mentioned previously, upon a TLB miss, AMD's PTE Coalescing feature will examine a collection of PTEs adjacent to the one actually needed to resolve the miss. On Zen 2 and newer processors, four aligned and contiguous PTEs (the first or second half of the cache line containing the PTE) will be inspected, while Zen+ and older processors examine eight entries (the entire cache line). If the MMU finds that these entries map aligned and contiguous physical memory and each has identical attributes, it may coalesce them into a single TLB entry.

This high-level specification leaves out details that fall into two main categories. First, it does not clearly specify how attributes that change dynamically as memory is accessed, namely the accessed and modified bits, impact the processor's decision to coalesce or not, nor does it explain how coalescing affects the processor's ability to manage these bits at a 4 KB granularity. Second, it does not explain what happens to 4 KB mappings that are already in the TLB when an overlapping coalesced mapping is created. To obtain

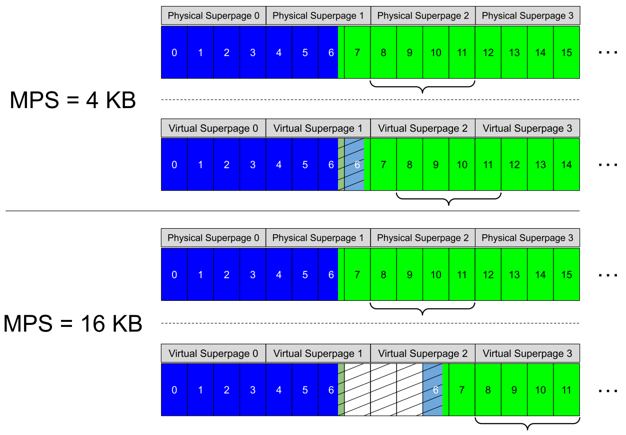

Figure 1: An illustration of an executable's read-only and code segments when mapped into a virtual address space with `maxpagesize` set to 4 KB and 16 KB. Blue shading represents read-only data, green shading represents code, empty pages represent an unmapped region in the address space, and diagonal lines indicate that a portion of the address space is not intended to be accessed.

these specifics, we take an experimental approach using a custom trace-driven microbenchmark.

3.1 Microbenchmark

The microbenchmark is designed to run on a FreeBSD system with reservation-based allocation enabled, but automatic 2 MB superpage promotion disabled. This allows the microbenchmark to allocate a large chunk of aligned, physically contiguous memory to which it can apply a variety of access patterns.

The program first requests from the operating system a 2 MB aligned region of memory that by default is 8 MB in size. It then explicitly zeroes the memory to ensure that physical pages have actually been allocated to the region, and afterwards optionally write-protects the pages. To finish preparing the memory, it clears the accessed and modified bits of each page within the region using the `madvise` system call.

The microbenchmark then allocates four performance counters (see Table 2), all related to L1 misses in the data TLB. (Unfortunately, no DTLB L1 hit counters are publicly documented by AMD.) They are:

- L2 miss at a 4 KB granularity
- L2 hit at a 4 KB granularity
- L2 miss at a coalesced page granularity
- L2 hit at a coalesced page granularity

We define a *coalescing subregion* as an aligned group of 4 or 8 pages (depending on the processor model) within the larger region of memory. A trace executed by the microbenchmark is divided into *words*, which are each composed of one or more *operations*. An operation corresponds to a read, write, or TLB invalidation of a single page within a coalescing subregion. A word is a sequence of operations that are applied to each coalescing subregion in order.

The performance counters are sampled after the execution of each word in the trace.

For example, the trace `r0 : r1 : r2 : r3` contains four words, each with a single operation. This trace instructs the program to read the first page within each subregion (first to last), then the second page within each subregion, and so on. In contrast, the trace `r0 r1 r2 r3` is made up of a single word which contains four operations. This corresponds to a sequential read of each page within each subregion, which is equivalent to a sequential read of every page within the larger region.

In order to make the performance counter values that it reports more clear, the microbenchmark offers four different looping options. *Operation repeats* cause each operation to be issued more than once. *Local repeats* apply each word to a given subregion multiple times, while *global repeats* apply each word to every subregion in order multiple times. *Trace repeats* loop the entire trace. In addition, using the `mincore` system call, the microbenchmark can optionally report the status of the valid, accessed, and modified bits for each page within the region at the end of its execution.

To reduce the number of extraneous events that are captured by the performance counters, the microbenchmark stores intermediate counter values in an array and prints them all out after the trace has finished executing, rather than displaying them to the user incrementally.

Because the `INVLPG` instruction used to invalidate a specific page in the TLB is privileged, we implement a small kernel module that wraps it in a system call handler to facilitate its use from userspace when needed.

3.2 Observed Behavior

Ryzen MMUs will coalesce 8 (on Zen and Zen+) or 4 (on Zen 2 and Zen 3) L1 PTEs contained within the same cache line or half cache line into a single TLB entry if they map aligned, contiguous physical memory and their attributes are exactly identical. In other words, the PTEs are required to map virtually aligned and contiguous regions into similarly aligned and physically contiguous 4 KB pages. Moreover, this means that each PTE must have the same protection (e.g. read-only versus read-write), accessed bit, and modified bit.

The requirement that all of the accessed bits be identical implies that each constituent page must have been touched to create a medium-sized superpage mapping. This is because coalescing only occurs when the MMU is trying to resolve a TLB miss due to a load or store involving one of the pages, which sets the accessed bit on that page. Hence, the only way for all of the accessed bits to be the same is for them to all be set. (It bears mentioning that speculatively executed loads and stores can potentially set the accessed flag.)

If a mapping is writable, there are two cases in which a coalesced superpage can be created: all of the modified bits are set, or all of them are unset. If they are all set, a read-write coalesced entry will be created. If the bits are all not set (i.e. each page within the coalescing region was read from but not written to), a coalesced entry will be created that is logically read-only.

When a write occurs within this read-only entry, it will be destroyed after rechecking the page table to confirm that the region is still mapped as writable. The write causes the modified bits within the PTEs to no longer be identical, and a read-write coalesced entry will not be created until all of the constituent pages are written to. This behavior is logically necessary to ensure that conflicting 4 KB and coalesced entries cannot exist simultaneously in the TLB, and allows the hardware to maintain 4 KB granularity modified bit tracking for coalesced entries.

The TPS paper [23] argues that under a LRU replacement policy, there is no need to explicitly evict smaller page mappings when a superpage mapping is created in the TLB. This is because the smaller mappings will naturally be used less frequently than the larger ones, so they will age out of the TLB on their own. The behavior of AMD's PTE Coalescing feature appears to be consistent with this approach.

For example, if a coalesced mapping is explicitly destroyed by issuing an `INVLPG` instruction that targets one of its constituent pages, an access to a different page within the coalescing region may still hit in the TLB due to a lingering 4 KB entry. Also, when a logically read-only coalesced entry that was created out of clean, read-write 4 KB entries is invalidated due to a write to one of its pages, a 4 KB granularity TLB hit may be observed upon an access to one of the other pages since some of the clean precursor entries have not yet been evicted from the TLB. (If the access is a write, the MMU will recheck that the relevant PTE is still writable while walking the page table to set the modified bit, and a page fault will occur if it is no longer writable.)

Finally, although AMD has not to our knowledge documented the fact that Zen+ CPUs support coalesced ITLB entries, the performance counters observed in Section 5 provide strong evidence that older Ryzen CPUs indeed can use PTE Coalescing on code mappings.

3.3 Discussion

These observations make it clear that PTE Coalescing is completely transparent from an architectural standpoint. In other words, architecturally defined state, such as the PTE's accessed and dirty bits, is maintained as if PTE Coalescing did not exist. And, in fact, the June 2023 edition of "AMD64 Architecture Programmer's Manual Volume 2: System Programming", never mentions PTE Coalescing.

Nonetheless, an operating system could take actions informed by these observations that lead to additional PTE Coalescing. For example, consider code or read-only data that is demand paged from a file. In general, the operating system will load into physical memory the pages neighboring that on which the page fault occurred. Suppose that only 7 out of 8 loaded pages are accessed on a Zen+ processor. The operating system could choose to set the accessed bit on the one page that hasn't been accessed, enabling PTE Coalescing to occur.

4 METHODOLOGY

4.1 Experimental Setup

Our Zen+ system uses a Ryzen 2700X 8-core processor with 16 MB of last-level cache (LLC), and our Zen 3 system uses a Ryzen 5900X 12-core processor with 64 MB of LLC. Both systems have 32 GB of RAM. The implementation of both processors is evenly split between two core complexes (CCXs). The 4 cores within a 2700X CCX can only spill to the local half of the LLC [6], and likewise for the 6 cores within a 5900X CCX [3]. Both processors support two hyperthreads (SMT threads) per core, and all hyperthreads are enabled on both systems. We pin workloads to just one CCX, and often just one hyperthread to reduce the variance in our measurements. Section 4.2 describes how we run the workloads in more detail.

Table 1 shows the TLB organizations of the two processors. Both TLB structures support page mappings of different sizes within the same structure. While the L2 DTLB doesn't cache 1 GB mappings on either Zen+ or Zen 3, there is a 64-entry Page Directory Cache (PDC) on Zen 3 that can hold 1 GB mappings evicted from the L1 DTLB [3]. 1 GB instruction mappings are smashed into 2 MB mappings and cached in the L2 ITLB [3, 6] on both Zen+ and Zen 3. AMD has not to our knowledge publicly documented support of PTE Coalescing in the Zen+ ITLB; nonetheless, our data in Section 5 provide strong evidence that coalescing does occur. It is unclear whether the granularity in this case is in fact 32 KB.

L1 ITLB		
4KB, 32KB*, 2MB, 1GB	64 entries	fully associative
L2 ITLB		
4KB, 32KB*, 2MB	512 entries	8-way set associative
L1 DTLB		
4KB, 32KB, 2MB, 1GB	64 entries	fully associative
L2 DTLB		
4KB, 32KB, 2MB	1536 entries	12-way set associative

(a) Zen+ (2700X)

L1 ITLB		
4KB, 16KB, 2MB, 1GB	64 entries	fully associative
L2 ITLB		
4KB, 16KB, 2MB	512 entries	8-way set associative
L1 DTLB		
4KB, 16KB, 2MB, 1GB	64 entries	fully associative
L2 DTLB		
4KB, 16KB, 2MB	2048 entries	16-way set associative

(b) Zen3 (5900X)

Table 1: Zen+ and Zen 3 TLB structures.

For 2700X versus 5900X comparisons, we use FreeBSD 14.0-CURRENT (commit 9d843ba) as the operating system. FreeBSD provides compile-time options to turn its reservation-based physical memory allocator on or off. This allows us to study the impact of PTE Coalescing's granularity (16 KB vs. 32 KB) while holding the physical memory allocation policy constant under an OS that intentionally manages physical contiguity (Section 2.2). Both the kernel and userland are built as standard production releases with debugging options turned off. As explained in Section 2.3, a 4 KB `maxpagesize` could shift a binary's executable code segment out of phase. We therefore build with a `maxpagesize` of 2 MB on FreeBSD to enable PTE Coalescing on the main executable of applications.

For Linux versus FreeBSD comparisons using a binary compiled on Linux, we use a 5900X system. The Linux distribution we use is Ubuntu 22.04.2. We build the application binary using the default GCC compiler and GNU toolchain. We then copy the binary to a FreeBSD partition in the same system. The binary is then run as is using the Linux binary compatibility layer in FreeBSD [8]. This minimizes any potential differences between the Linux and FreeBSD binaries. With this setup we study the difference in effectiveness of an OS that intentionally creates physical contiguity and an OS that relies on coincidental contiguity.

We collect information from hardware performance-monitoring counters [2, 4] using the `perf` [11] utility on Linux and the `pmcstat` [13] utility on FreeBSD. We measure user and kernel space results separately; the numbers presented are for user space, because by default kernel code and data are already mapped with superpages to the fullest extent possible. We report the median of three runs for all workloads. Table 2 lists the `perf` events used in our tests. `pmcstat` uses the same event names as `perf`. L2 DTLB hits and misses are computed as sums of page-size-specific L2 DTLB hits and misses. L1 DTLB misses are computed as sum of L2 DTLB hits and misses, and the same procedure is followed for the L1 ITLB. When results are presented as "per thousand instructions" in Section 5, we divide the corresponding counter by retired instructions and then multiply by 1000.

Counter Description	Perf Events & Equations
Retired instructions	ex_ret_instr
Unhalted clock cycles	ls_not_halted_cyc
L2 DTLB 1GB hits	ls_l1_d_tlb_miss.tlb_reload_1g_l2_hit
L2 DTLB 2MB hits	ls_l1_d_tlb_miss.tlb_reload_2m_l2_hit
L2 DTLB coalesced hits	ls_l1_d_tlb_miss.tlb_reload_{coalesced_page,32k}_l2_hit
L2 DTLB 4KB hits	ls_l1_d_tlb_miss.tlb_reload_4k_l2_hit
L2 DTLB 1GB misses	ls_l1_d_tlb_miss.tlb_reload_1g_l2_miss
L2 DTLB 2MB misses	ls_l1_d_tlb_miss.tlb_reload_2m_l2_miss
L2 DTLB coalesced misses	ls_l1_d_tlb_miss.tlb_reload_{coalesced_page,32k}_l2_miss
L2 DTLB 4KB misses	ls_l1_d_tlb_miss.tlb_reload_4k_l2_miss
L2 ITLB hits	bp_l1_tlb_miss_l2_tlb_hit
L2 ITLB misses	bp_l1_tlb_miss_l2_tlb_miss
L1 ITLB misses	*L2 ITLB hits + L2 ITLB misses*
L2 DTLB hits	*SUM(L2 DTLB {1GB,2MB,coalesced,4KB} hits)*
L2 DTLB misses	*SUM(L2 DTLB {1GB,2MB,coalesced,4KB} misses)*
L1 DTLB misses	*L2 DTLB hits + L2 DTLB misses*

Table 2: Performance counters and their corresponding `perf`/`pmcstat` event names [10, 14].

4.2 Workloads

Our workloads are based on eight widely used applications with various code and data sizes and access patterns, covering a range of application types, including a compiler (Clang), an out-of-core PageRank implementation (GraphChi), a framework for linear classification training (BlockSVM), a multi-threaded data mining algorithm (Freqmine), a language runtime with just-in-time (JIT) compilation capabilities (the V8 Javascript runtime in Node.js), a web application framework (Node.js), a machine emulator performing JIT-compilation-like binary translation (QEMU), a physics simulation algorithm (XSBench), and a cache-aware optimization algorithm (Canneal).

The applications differ in whether they execute statically compiled code from files or JIT-compiled code in anonymous memory, and whether they are statically linked or dynamically linked. Some applications naturally run as multi-threaded workloads. Otherwise, we run a single instance of the application within a single thread.

In either case, we assign dedicated hyperthreads, and we pin the workloads to the assigned hyperthreads with the `cpuset` [12] utility on FreeBSD or the `taskset` [17] utility on Linux, and measure only the activity of those hyperthreads. Workloads are run one at a time, and the machine is rebooted in between workloads to ensure the same starting conditions for all. With the exception of the hyperthreads running the workload, the machine is otherwise idle.

	Main exec size (MB)	Main exec RX aligned	Data size (GB)	Static linking	JIT code in anon	# of threads
Clang-Linux	108.5	Y	<0.002	N	N	1
Clang-FreeBSD	109.3	Y	<0.002	N	N	1
GraphChi-Linux	0.3	N	1	N	N	4
GraphChi-FreeBSD	0.3	Y	1	N	N	4
BlockSVM-Linux	0.15	N	2.3	N	N	1
BlockSVM-FreeBSD	0.15	Y	2.3	N	N	1
Freqmine-Linux	0.07	N	1.4	N	N	4
Freqmine-FreeBSD	0.07	Y	1.4	N	N	4
Node.js-Linux	71.6	Y	0.05	Y	Y	1
Node.js-FreeBSD	41.6	Y	0.05	N	Y	1
QEMU-Linux	15.8	N	0.7	N	Y	1
QEMU-FreeBSD	15.5	Y	0.7	N	Y	1
XSBench-Linux	0.05	N	5.8	N	N	4
XSBench-FreeBSD	0.05	Y	5.8	N	N	4
Canneal-Linux	0.22	N	0.7	N	N	1
Canneal-FreeBSD	0.22	Y	0.7	N	N	1

Table 3: Workload characteristics: the main executable size in MB, whether the read-and-execute code (RX) segment of the main executable is properly aligned for PTE coalescing, the amount of heap or mmap'd data in GB, whether the main executable is statically linked, whether the workload generates JIT-compiled code, and the number of threads. "-Linux" stands for binaries built on the Linux system and running either natively under Linux or with the compatibility layer under FreeBSD. "-FreeBSD" stands for binaries built on the FreeBSD system and running natively under FreeBSD.

Table 3 lists the characteristics of each workload. Main executables and shared libraries are demand-paged from ordinary files. JIT-compiled code is generated at runtime and written into anonymous memory. The main executables range in size from 0.07 MB to as large as 109.3 MB. The heaps range in size from a few MB to as large as 2.3 GB. For FreeBSD native binaries, we build with a modified `maxpagesize` of 2 MB to ensure proper alignment for the RX segment of the main executable. On Linux, default linker scripts are used, and proper alignment of the RX segment does coincidentally occur in some cases.

Clang (SQLite): Clang is a C/C++ compiler with a built-in assembler based on the LLVM infrastructure. We run one instance of Clang 16.0.6 on one hyperthread, compiling the source code for version 3.42.0 of the SQLite database [16] 10 times back-to-back. The compilation options are "`-O2 -DSQLITE_ENABLE_FTS3 -DSQLITE_ENABLE_RTREE -DSQLITE_ENABLE_DBSTAT_VTAB -DSQLITE_ENABLE_RBU -DSQLITE_ENABLE_SESSION`".

GraphChi: GraphChi [27] uses out-of-core implementations for solving graph computations. We use GraphChi to compute 3 iterations of PageRank on the preprocessed Twitter-2010 dataset [25].

BlockSVM: BlockSVM [38] is a framework for linear classification training. We measure a model training run on the kdd2010-bridge dataset [24] with options "`-a cookie -m 1 -O 10000 -s 1 -v 5 -M 1`". It has a relatively large data footprint.

Freqmine: Freqmine is a PARSEC [19] benchmark that implements an array-based version of the FP-growth (Frequent Pattern-growth) method [22] for data mining. It has a relatively large data footprint.

Node.js: Node.js [9] is a JavaScript runtime built on the V8 JavaScript engine [18]. We run version 18.16.1 of Node.js, and use the React server-side rendering benchmark [1]. By default this benchmark tries to run for a fixed amount of time, varying the number of iterations. We instead fix the number of iterations to achieve roughly the same duration as the benchmark would by default. This ensures the same fixed amount of work across runs. We run one instance of Node.js on one hyperthread.

QEMU: QEMU is an emulator that supports running programs compiled for a different architecture [15]. We use QEMU version 8.0.3 with one hyperthread to run a full FreeBSD 14.0-CURRENT system image built for AArch64. Within the single-vCPU emulated system, we run the same workload as Clang (SQLite), using the bundled Clang compiler to compile SQLite for a fixed number of times. Since code targeted at AArch64 does not run natively on our x86-64 processor, QEMU performs dynamic binary translation, which generates a large amount of code that is stored in anonymous virtual memory. We refer to this as JIT-compiled code.

XSBench: XSBench [36] implements a key computational kernel of the Monte Carlo neutron transport algorithm. We ran version 20 under the OpenMP threading model with options "`-t 4 -m history -s large -l 34 -p 5000000 -G unionized`". This results in a relatively large data footprint, and 4 hyperthreads being fully utilized.

Canneal: Canneal is a PARSEC [19] benchmark that implements a cache-aware simulated annealing. We measure a single-threaded run on the *2500000.nets* input with 6000 temperature steps, 15000 swaps per temperature step, and a starting temperature of 2000.

5 RESULTS

Our goal is to evaluate the extent to which reservations allow PTE Coalescing and superpages to improve TLB performance for data and code. There are three central questions that we attempt to answer. First, what advantages for the effectiveness of PTE Coalescing do reservations have over simply relying on the buddy allocator to produce coincidental contiguity? Second, how successful is PTE Coalescing at increasing TLB coverage versus traditional 2 MB superpages? Third, what was the effect of AMD's decision to switch from 32 KB coalescing to 16 KB coalescing in their newer processors? We first consider the impact of reservations on the data TLB, and then examine the instruction TLB. After that, we discuss how these observations relate to performance in terms of unhalted clock cycles, and finally compare the newer 5900X CPU to the older 2700X.

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	3.781	3.064	3.964	3.121	0.792	2.628	1.916	1.453	0.913	3.009	2.173	1.811	1.249
L2 DTLB hits	3.439	2.807	3.593	2.908	0.757	2.476	1.836	1.401	0.880	2.686	1.950	1.631	1.113
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	0.136	0.000	0.000	0.268	0.000	0.000	0.110	0.196	0.000	0.000	0.124	0.249
L2 DTLB coalesced hits	0.516	0.429	0.078	2.312	0.180	0.000	1.365	0.920	0.424	0.000	1.142	0.807	0.377
L2 DTLB 4 KB hits	2.923	2.242	3.514	0.596	0.309	2.476	0.471	0.371	0.259	2.685	0.808	0.701	0.487
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	4.856	0.007	0.009	35.461	0.010	0.014	7.839	22.268	0.009	0.013	7.581	22.346
L2 DTLB coalesced hits (%)	15.013	15.270	2.179	79.484	23.757	0.003	74.352	65.685	48.256	0.008	58.551	49.473	33.897
L2 DTLB 4 KB hits (%)	84.987	79.874	97.815	20.507	40.781	99.987	25.634	26.476	29.476	99.983	41.436	42.947	43.757
L2 DTLB misses	0.342	0.257	0.371	0.212	0.035	0.151	0.080	0.052	0.033	0.324	0.223	0.180	0.135
L2 DTLB 1 GB misses	0.048	0.041	0.039	0.036	0.023	0.022	0.021	0.019	0.017	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.019	0.018	0.027	0.021	0.008	0.025	0.020	0.016	0.012	0.000	0.000	0.002	0.002
L2 DTLB coalesced misses	0.038	0.026	0.008	0.145	0.001	0.000	0.031	0.011	0.002	0.000	0.059	0.029	0.006
L2 DTLB 4 KB misses	0.237	0.172	0.298	0.010	0.004	0.104	0.008	0.006	0.003	0.324	0.164	0.149	0.127
L2 DTLB 1 GB misses (%)	14.067	15.914	10.511	16.917	66.143	14.638	25.905	35.610	50.182	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	5.448	6.885	7.150	10.110	21.561	16.806	25.489	31.432	36.563	0.040	0.053	1.152	1.473
L2 DTLB coalesced misses (%)	11.260	10.202	2.073	68.191	1.635	0.007	38.801	21.639	4.707	0.002	26.400	15.844	4.717
L2 DTLB 4 KB misses (%)	69.225	67.000	80.266	4.782	10.660	68.549	9.805	11.320	8.547	99.958	73.547	83.003	93.809
L1 ITLB misses	2.954	2.963	3.330	1.518	1.513	3.279	1.578	1.531	1.531	3.753	2.111	2.107	2.102
L2 ITLB hits	2.751	2.767	3.094	1.459	1.455	3.042	1.509	1.469	1.470	3.367	2.014	2.008	2.006
L2 ITLB misses	0.203	0.197	0.236	0.059	0.058	0.237	0.068	0.062	0.061	0.386	0.097	0.098	0.097
Unhalted clock cycles	0.779	0.777	0.781	0.767	0.755	1.019	1.004	1.000	0.998	1.383	1.355	1.349	1.342

Table 4: Clang (SQLite)

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	13.895	4.431	14.775	12.567	8.207	18.559	16.057	11.105	10.991	17.955	14.752	10.521	10.467
L2 DTLB hits	4.094	4.416	3.627	3.719	3.491	2.923	3.133	3.985	3.958	2.376	2.351	2.901	2.866
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	4.100	0.013	0.012	2.180	0.002	0.001	2.888	2.862	0.001	0.001	2.057	2.033
L2 DTLB coalesced hits	0.824	0.045	0.000	3.166	0.756	0.000	2.738	0.706	0.703	0.000	1.869	0.387	0.381
L2 DTLB 4 KB hits	3.270	0.271	3.614	0.541	0.555	2.922	0.393	0.391	0.393	2.375	0.481	0.457	0.452
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	92.846	0.347	0.327	62.445	0.060	0.047	72.469	72.311	0.052	0.039	70.904	70.927
L2 DTLB coalesced hits (%)	20.132	1.010	0.000	85.127	21.650	0.001	87.402	17.712	17.771	0.000	79.492	13.338	13.300
L2 DTLB 4 KB hits (%)	79.868	6.144	99.652	14.546	15.905	99.939	12.551	9.819	9.918	99.948	20.469	15.757	15.773
L2 DTLB misses	9.800	0.015	11.149	8.848	4.717	15.635	12.924	7.120	7.034	15.579	12.402	7.620	7.601
L2 DTLB 1 GB misses	0.006	0.003	0.005	0.005	0.005	0.001	0.001	0.001	0.001	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.008	0.007	0.006	0.006	0.060	0.001	0.001	0.857	0.828	0.002	0.002	1.606	1.602
L2 DTLB coalesced misses	0.541	0.000	0.000	6.879	2.745	0.000	10.267	3.637	3.607	0.000	9.076	2.742	2.732
L2 DTLB 4 KB misses	9.246	0.004	11.137	1.957	1.907	15.632	2.655	2.625	2.597	15.577	3.324	3.271	3.266
L2 DTLB 1 GB misses (%)	0.060	23.129	0.048	0.061	0.106	0.009	0.009	0.018	0.016	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.077	47.410	0.056	0.065	1.270	0.008	0.009	12.034	11.773	0.013	0.016	21.075	21.084
L2 DTLB coalesced misses (%)	5.525	0.596	0.002	77.750	58.191	0.002	79.438	51.077	51.289	0.000	73.180	35.993	35.947
L2 DTLB 4 KB misses (%)	94.338	28.865	99.894	22.125	40.433	99.982	20.543	36.870	36.922	99.987	26.803	42.932	42.969
L1 ITLB misses	0.029	0.000	0.028	0.029	0.028	0.288	0.300	0.305	0.313	0.000	0.000	0.001	0.001
L2 ITLB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 ITLB misses	0.029	0.000	0.028	0.028	0.028	0.288	0.300	0.305	0.313	0.000	0.000	0.000	0.001
Unhalted clock cycles	0.635	0.572	0.842	0.835	0.793	1.072	1.060	1.000	1.007	1.752	1.740	1.554	1.560

Table 5: GraphChi

The results of our evaluation can be found in Table 4 through Table 9. All numbers presented are per thousand retired instructions, with the exception of unhalted clock cycles, which are normalized against a baseline of a FreeBSD binary running with superpages enabled on a 5900X processor. The shaded sections break down L2 DTLB hits and misses by page size, first in absolute terms and then on a proportional basis. On Linux, "nothp" refers to a system with Transparent Huge Pages disabled, while "thp" refers to one with it enabled. On FreeBSD, "noresv" indicates that reservations have been disabled, "reserv" means that reservations are turned on, "super" data was collected with superpage promotion on, and "nomdv" means that `jemalloc` has been instructed not to make `madvise` system calls with the `MADV_FREE` flag. Only user space hits, misses, and cycles are reported.

`jemalloc` makes `MADV_FREE` calls to indicate to the OS that a particular region of memory no longer contains useful data and that it may be asynchronously reclaimed by the system and remapped on the next touch by the application. FreeBSD makes use of this information by clearing the accessed and modified flags within the region and prioritizing its constituent pages in the page reclamation process. This forces the virtual memory system to demote any 2 MB superpages that partially overlap with this region, potentially

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	11.321	6.357	11.335	10.928	3.389	6.821	6.257	3.493	3.491	6.801	6.088	3.411	3.400
L2 DTLB hits	3.311	3.859	3.285	5.376	3.293	1.512	2.267	3.312	3.307	1.322	2.043	2.948	2.938
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	1.866	0.000	0.000	2.458	0.000	0.000	2.401	2.402	0.000	0.000	2.215	2.210
L2 DTLB coalesced hits	0.585	0.097	0.001	4.870	0.241	0.000	2.240	0.892	0.883	0.000	1.920	0.620	0.620
L2 DTLB 4 KB hits	2.727	1.896	3.284	0.506	0.593	1.512	0.027	0.019	0.023	1.322	0.123	0.113	0.108
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	48.358	0.012	0.004	74.662	0.009	0.003	72.489	72.613	0.013	0.004	75.133	75.217
L2 DTLB coalesced hits (%)	17.658	2.509	0.037	90.590	7.320	0.000	98.802	26.929	26.697	0.000	93.989	21.020	21.106
L2 DTLB 4 KB hits (%)	82.342	49.134	99.951	9.406	18.018	99.991	1.195	0.582	0.690	99.987	6.006	3.847	3.676
L2 DTLB misses	8.010	2.498	8.049	5.552	0.097	5.309	3.991	0.181	0.184	5.479	4.045	0.463	0.462
L2 DTLB 1 GB misses	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.000	0.257	0.000	0.000	0.029	0.000	0.000	0.062	0.064	0.000	0.000	0.274	0.273
L2 DTLB coalesced misses	0.217	0.091	0.001	5.379	0.012	0.000	3.947	0.077	0.078	0.000	3.985	0.127	0.126
L2 DTLB 4 KB misses	7.792	2.150	8.048	0.173	0.056	5.309	0.043	0.042	0.042	5.479	0.060	0.062	0.062
L2 DTLB 1 GB misses (%)	0.001	0.000	0.001	0.001	0.041	0.000	0.000	0.002	0.002	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.001	10.273	0.003	0.004	29.975	0.003	0.004	34.326	34.647	0.005	0.005	59.176	59.188
L2 DTLB coalesced misses (%)	2.712	3.646	0.015	96.875	12.456	0.000	98.908	42.425	42.449	0.000	98.513	27.345	27.295
L2 DTLB 4 KB misses (%)	97.287	86.081	99.981	3.120	57.528	99.997	1.088	23.247	22.902	99.995	1.482	13.479	13.517
L1 ITLB misses	0.050	0.001	0.023	0.023	0.023	0.005	0.005	0.006	0.006	0.000	0.000	0.000	0.000
L2 ITLB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 ITLB misses	0.050	0.001	0.023	0.023	0.023	0.005	0.005	0.006	0.006	0.000	0.000	0.000	0.000
Unhalted clock cycles	1.281	1.162	1.263	1.254	1.137	1.113	1.106	1.000	1.000	1.374	1.385	1.136	1.137

Table 6: BlockSVM

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	0.077	0.016	0.189	0.085	0.026	0.297	0.091	0.034	0.034	0.439	0.091	0.043	0.042
L2 DTLB hits	0.070	0.015	0.171	0.079	0.024	0.276	0.084	0.030	0.030	0.404	0.081	0.036	0.035
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	0.001	0.000	0.000	0.002	0.000	0.000	0.001	0.001	0.000	0.000	0.002	0.002
L2 DTLB coalesced hits	0.005	0.002	0.002	0.060	0.011	0.000	0.062	0.015	0.015	0.000	0.048	0.009	0.009
L2 DTLB 4 KB hits	0.064	0.012	0.168	0.019	0.011	0.276	0.023	0.014	0.014	0.404	0.033	0.026	0.025
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	9.100	0.171	0.339	8.471	0.094	0.292	3.640	3.621	0.059	0.272	4.291	4.449
L2 DTLB coalesced hits (%)	7.443	13.790	1.304	75.827	45.077	0.004	73.016	49.493	49.175	0.001	58.801	24.809	25.495
L2 DTLB 4 KB hits (%)	92.557	77.110	98.525	23.834	46.453	99.902	26.692	46.867	47.204	99.940	40.926	70.900	70.056
L2 DTLB misses	0.007	0.001	0.018	0.006	0.002	0.021	0.007	0.004	0.004	0.035	0.010	0.007	0.007
L2 DTLB 1 GB misses	0.000	0.000	0.001	0.001	0.001	0.002	0.001	0.001	0.001	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.000	0.000	0.004	0.001	0.000	0.007	0.001	0.001	0.001	0.000	0.000	0.000	0.000
L2 DTLB coalesced misses	0.000	0.000	0.000	0.002	0.000	0.000	0.003	0.001	0.001	0.000	0.002	0.001	0.001
L2 DTLB 4 KB misses	0.007	0.000	0.014	0.002	0.001	0.012	0.002	0.002	0.002	0.035	0.008	0.006	0.006
L2 DTLB 1 GB misses (%)	3.939	16.216	3.178	9.976	21.191	8.198	17.145	27.136	25.982	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.962	6.724	20.574	21.811	20.445	33.834	18.450	12.819	13.173	0.108	0.282	1.171	1.374
L2 DTLB coalesced misses (%)	2.011	2.768	0.456	39.069	13.344	0.031	39.396	19.625	19.745	0.011	21.036	10.048	9.817
L2 DTLB 4 KB misses (%)	93.087	74.293	75.792	29.144	45.020	57.938	25.009	40.420	41.100	99.882	78.682	88.781	88.808
L1 ITLB misses	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 ITLB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 ITLB misses	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unhalted clock cycles	0.508	0.500	0.927	0.924	0.925	1.002	1.004	1.000	1.002	0.930	0.929	0.932	0.930

Table 7: Freqmine

reducing the use of superpages. (This issue is not relevant to our Linux experiments since the `malloc` included with glibc does not make use of `MADV_FREE`.)

5.1 Data TLB

In general, reservations have a strong impact on the performance of the data TLB. We first focus on applications running natively on a FreeBSD system with a 5900X processor, and then compare the behavior of identical binaries running both natively on Linux and under Linux emulation on FreeBSD.

5.1.1 Clang (SQLite). Enabling reservations reduces the number of L1 DTLB misses from 2.628 to 1.916. This closely tracks the decrease in the number of L2 hits (2.476 to 1.836). The breakdown of L2 hits by page size suggests that this is mostly explained by PTE Coalescing. Without reservations, virtually all L2 hits involve 4 KB pages; with reservations explicilty creating physical contiguity, nearly three quarters of them make use of 16 KB coalesced pages. L2 misses also decrease substantially, and exhibit a similar shift towards the use of coalesced pages.

Turning on superpages actually results in a smaller decrease in L1 DTLB misses than does enabling reservations. However, this is

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	7.295	7.087	7.781	5.629	5.550	8.039	6.753	6.746	6.794	5.960	4.705	4.694	4.747
L2 DTLB hits	7.216	7.009	7.694	5.553	5.474	7.951	6.672	6.665	6.714	5.860	4.612	4.603	4.654
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	0.006	0.000	0.000	0.048	0.000	0.000	0.002	0.002	0.000	0.000	0.007	0.007
L2 DTLB coalesced hits	0.364	0.440	0.048	1.338	1.266	0.000	1.130	1.192	1.167	0.006	0.548	0.539	0.542
L2 DTLB 4 KB hits	6.853	6.563	7.646	4.215	4.160	7.951	5.542	5.472	5.546	5.854	4.064	4.057	4.105
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	0.086	0.004	0.006	0.880	0.005	0.005	0.024	0.024	0.005	0.005	0.145	0.152
L2 DTLB coalesced hits (%)	5.042	6.281	0.620	24.087	23.126	0.001	16.928	17.882	17.377	0.106	11.875	11.716	11.645
L2 DTLB 4 KB hits (%)	94.958	93.634	99.375	75.907	75.994	99.995	83.066	82.094	82.599	99.890	88.119	88.139	88.203
L2 DTLB misses	0.078	0.078	0.087	0.076	0.076	0.088	0.081	0.081	0.079	0.100	0.093	0.091	0.093
L2 DTLB 1 GB misses	0.005	0.005	0.010	0.009	0.009	0.008	0.007	0.007	0.007	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.016	0.015	0.011	0.009	0.009	0.008	0.009	0.009	0.008	0.000	0.000	0.000	0.000
L2 DTLB coalesced misses	0.003	0.003	0.000	0.004	0.004	0.000	0.004	0.004	0.004	0.000	0.003	0.003	0.004
L2 DTLB 4 KB misses	0.053	0.054	0.066	0.055	0.055	0.072	0.061	0.060	0.060	0.100	0.089	0.087	0.090
L2 DTLB 1 GB misses (%)	7.000	6.920	11.517	11.347	11.213	9.000	8.881	9.070	8.961	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	20.884	19.779	12.678	11.196	11.389	9.043	11.005	10.709	9.872	0.020	0.029	0.064	0.064
L2 DTLB coalesced misses (%)	3.728	3.612	0.076	5.022	4.865	0.015	5.057	5.189	5.261	0.012	3.622	3.852	3.968
L2 DTLB 4 KB misses (%)	68.389	69.688	75.729	72.435	72.533	81.942	75.057	75.032	75.906	99.968	96.349	96.084	95.968
L1 ITLB misses	0.670	0.408	0.691	0.044	0.043	0.676	0.133	0.151	0.147	0.561	0.219	0.228	0.232
L2 ITLB hits	0.664	0.404	0.685	0.042	0.041	0.672	0.131	0.149	0.145	0.558	0.217	0.226	0.230
L2 ITLB misses	0.006	0.004	0.006	0.002	0.002	0.004	0.002	0.002	0.002	0.003	0.002	0.002	0.002
Unhalted clock cycles	0.720	0.720	1.088	1.087	1.083	1.000	0.999	1.000	1.001	1.364	1.357	1.357	1.358

Table 8: Node.js

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	3.654	0.836	3.900	2.080	0.800	3.853	2.073	0.831	0.767	4.511	2.197	1.199	1.106
L2 DTLB hits	3.373	0.819	3.589	1.904	0.773	3.562	1.913	0.806	0.742	3.938	1.789	1.008	0.966
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	0.475	0.000	0.000	0.406	0.000	0.000	0.392	0.380	0.000	0.000	0.485	0.493
L2 DTLB coalesced hits	0.235	0.037	0.031	1.529	0.217	0.012	1.543	0.267	0.218	0.001	1.157	0.221	0.197
L2 DTLB 4 KB hits	3.137	0.307	3.558	0.375	0.150	3.550	0.370	0.147	0.144	3.937	0.632	0.302	0.276
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	57.948	0.002	0.004	52.500	0.002	0.005	48.630	51.204	0.003	0.005	48.107	51.052
L2 DTLB coalesced hits (%)	6.973	4.559	0.867	80.294	28.124	0.342	80.673	33.102	29.391	0.019	64.658	21.954	20.405
L2 DTLB 4 KB hits (%)	93.027	37.493	99.131	19.701	19.376	99.656	19.321	18.268	19.406	99.978	35.336	29.938	28.543
L2 DTLB misses	0.281	0.017	0.311	0.176	0.027	0.291	0.160	0.025	0.025	0.573	0.407	0.191	0.140
L2 DTLB 1 GB misses	0.023	0.014	0.013	0.013	0.010	0.026	0.021	0.017	0.016	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.001	0.001	0.018	0.015	0.012	0.001	0.001	0.002	0.003	0.000	0.000	0.005	0.006
L2 DTLB coalesced misses	0.019	0.000	0.001	0.132	0.002	0.000	0.121	0.001	0.001	0.000	0.180	0.004	0.004
L2 DTLB 4 KB misses	0.237	0.001	0.279	0.016	0.003	0.264	0.017	0.004	0.005	0.573	0.227	0.182	0.130
L2 DTLB 1 GB misses (%)	8.353	86.306	4.116	7.103	36.503	8.893	13.346	69.315	66.175	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.438	8.682	5.701	8.726	45.610	0.362	0.623	9.899	10.243	0.022	0.031	2.699	4.082
L2 DTLB coalesced misses (%)	6.923	0.283	0.244	75.181	5.830	0.125	75.665	3.859	3.534	0.018	44.177	2.169	2.825
L2 DTLB 4 KB misses (%)	84.285	4.729	89.938	8.990	12.057	90.620	10.366	16.928	20.047	99.960	55.793	95.132	93.093
L1 ITLB misses	1.550	0.069	1.748	0.800	0.083	1.544	0.614	0.040	0.041	1.742	0.584	0.069	0.028
L2 ITLB hits	1.241	0.068	1.399	0.676	0.082	1.214	0.506	0.040	0.040	1.346	0.492	0.068	0.028
L2 ITLB misses	0.309	0.001	0.349	0.124	0.001	0.329	0.108	0.000	0.000	0.396	0.093	0.001	0.000
Unhalted clock cycles	1.020	0.976	1.022	0.996	0.967	1.064	1.028	1.000	1.003	1.367	1.344	1.304	1.292

Table 9: QEMU

mostly an artifact of the interaction between the FreeBSD `malloc` package's extensive use of `MADV_FREE` and Clang's memory allocation pattern; disabling `MADV_FREE` makes superpages much more effective. Under this configuration, L2 hits use 2 MB pages over 20% of the time, and use 16 KB pages almost half of the time. This establishes that although to some extent 2 MB superpages supplant coalesced superpages, the latter still play an important role in improving DTLB hit rates. Here, there are very few L2 DTLB misses, and they tend to make use of larger page sizes.

For Clang, Linux's buddy allocator is more effective at creating the physical contiguity needed to coalesce PTEs than FreeBSD without reservations. However, with reservations enabled, FreeBSD dominates Linux in this regard, producing 2.312 coalesced L2 hits to Linux's 0.516, and a similar pattern holds for L2 misses. This allows FreeBSD to take the lead in L1 misses as well. Finally, FreeBSD's superpage promotion system is much more effective at reducing L1 misses than Linux's THP.

5.1.2 *GraphChi.*

Turning reservations on has an even stronger impact on L1 DTLB misses for GraphChi, dropping them from 18.559 to 16.057, and the shift from 4 KB L2 hits to 16 KB L2 hits is more pronounced, with nearly 90% of L2 hits being coalesced

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	27.330	4.486	26.041	23.162	4.781	30.760	27.093	5.731	5.667	30.745	27.393	6.035	5.908
L2 DTLB hits	6.167	4.300	4.633	5.048	4.510	5.957	6.248	5.479	5.417	5.144	5.010	5.410	5.340
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	3.907	0.002	0.002	4.179	0.002	0.002	4.881	4.876	0.002	0.002	5.056	4.968
L2 DTLB coalesced hits	0.005	0.000	0.000	4.446	0.000	0.000	5.462	0.263	0.295	0.000	4.235	0.144	0.161
L2 DTLB 4 KB hits	6.161	0.393	4.632	0.601	0.331	5.955	0.783	0.334	0.246	5.141	0.773	0.210	0.211
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	90.871	0.035	0.030	92.666	0.030	0.028	89.100	90.015	0.043	0.037	93.456	93.031
L2 DTLB coalesced hits (%)	0.085	0.000	0.001	88.070	0.000	0.000	87.432	4.800	5.448	0.000	84.532	2.669	3.011
L2 DTLB 4 KB hits (%)	99.915	9.129	99.964	11.900	7.333	99.970	12.540	6.100	4.537	99.957	15.431	3.876	3.957
L2 DTLB misses	21.164	0.186	21.408	18.114	0.271	24.803	20.845	0.252	0.250	25.602	22.383	0.626	0.568
L2 DTLB 1 GB misses	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.000	0.129	0.001	0.001	0.204	0.001	0.001	0.221	0.222	0.001	0.002	0.513	0.470
L2 DTLB coalesced misses	0.002	0.000	0.014	17.884	0.001	0.000	20.786	0.024	0.022	0.000	22.364	0.106	0.090
L2 DTLB 4 KB misses	21.162	0.057	21.393	0.230	0.066	24.802	0.058	0.006	0.006	25.600	0.017	0.008	0.007
L2 DTLB 1 GB misses (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.000	69.146	0.005	0.005	75.348	0.004	0.005	87.844	88.693	0.006	0.007	81.914	82.814
L2 DTLB coalesced misses (%)	0.010	0.000	0.064	98.726	0.481	0.000	99.716	9.646	8.752	0.000	99.917	16.887	15.868
L2 DTLB 4 KB misses (%)	99.990	30.854	99.931	1.269	24.171	99.996	0.279	2.510	2.554	99.994	0.076	1.199	1.318
L1 ITLB misses	0.003	0.000	0.002	0.002	0.002	0.003	0.003	0.003	0.003	0.000	0.000	0.000	0.000
L2 ITLB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 ITLB misses	0.003	0.000	0.002	0.002	0.002	0.003	0.003	0.003	0.003	0.000	0.000	0.000	0.000
Unhalted clock cycles	0.843	0.759	1.191	1.185	1.071	1.124	1.116	1.000	1.001	1.336	1.367	1.145	1.146

Table 10: XSBench

Counter	Linux binary, 5900X					FreeBSD binary, 5900X				FreeBSD binary, 2700X			
	Linux		FreeBSD										
	nothp	thp	noresv	reserv	super	noresv	reserv	super	nomdv	noresv	reserv	super	nomdv
L1 DTLB misses	5.471	2.658	5.391	5.218	2.248	3.679	3.559	1.666	1.645	3.707	3.536	1.602	1.600
L2 DTLB hits	0.903	1.807	0.850	1.631	2.216	0.692	1.236	1.626	1.605	0.655	1.202	1.572	1.571
L2 DTLB 1 GB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits	0.000	1.451	0.000	0.000	2.008	0.000	0.000	1.285	1.280	0.000	0.000	1.294	1.294
L2 DTLB coalesced hits	0.123	0.000	0.000	1.505	0.002	0.000	1.087	0.215	0.217	0.000	0.974	0.178	0.180
L2 DTLB 4 KB hits	0.781	0.355	0.849	0.126	0.205	0.691	0.149	0.126	0.108	0.655	0.228	0.100	0.097
L2 DTLB 1 GB hits (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
L2 DTLB 2 MB hits (%)	0.000	80.330	0.031	0.018	90.635	0.026	0.017	79.036	79.758	0.026	0.011	82.338	82.361
L2 DTLB coalesced hits (%)	13.575	0.016	0.016	92.239	0.101	0.000	87.910	13.215	13.527	0.000	81.053	11.323	11.436
L2 DTLB 4 KB hits (%)	86.425	19.654	99.953	7.743	9.264	99.974	12.073	7.748	6.716	99.974	18.935	6.339	6.203
L2 DTLB misses	4.567	0.852	4.541	3.586	0.032	2.987	2.322	0.041	0.040	3.052	2.334	0.030	0.029
L2 DTLB 1 GB misses	0.047	0.019	0.008	0.002	0.004	0.031	0.030	0.029	0.028	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses	0.001	0.016	0.048	0.045	0.017	0.012	0.012	0.010	0.010	0.000	0.000	0.001	0.000
L2 DTLB coalesced misses	0.000	0.410	0.001	3.376	0.001	0.000	2.274	0.000	0.000	0.000	2.300	0.002	0.001
L2 DTLB 4 KB misses	4.519	0.406	4.484	0.164	0.011	2.944	0.007	0.001	0.002	3.052	0.034	0.028	0.027
L2 DTLB 1 GB misses (%)	1.027	2.260	0.186	0.042	12.813	1.030	1.294	70.826	70.609	0.000	0.000	0.000	0.000
L2 DTLB 2 MB misses (%)	0.033	1.878	1.048	1.262	52.151	0.407	0.513	24.563	24.479	0.013	0.019	2.558	1.622
L2 DTLB coalesced misses (%)	0.000	48.185	0.024	94.136	1.956	0.000	97.897	1.009	0.921	0.000	98.538	6.617	4.851
L2 DTLB 4 KB misses (%)	98.940	47.677	98.742	4.561	33.081	98.564	0.295	3.602	3.991	99.987	1.444	90.825	93.527
L1 ITLB misses	0.002	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.003	0.004	0.003	0.003
L2 ITLB hits	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.003	0.004	0.003	0.003
L2 ITLB misses	0.002	0.002	0.001	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Unhalted clock cycles	0.962	0.813	0.865	0.854	0.763	1.076	1.078	1.000	0.983	1.263	1.269	1.142	1.142

Table 11: Canneal

after reservations were enabled. Almost 80% of L2 misses can be resolved by creating coalesced entries that go on to create more L1 and L2 hits in the future.

Superpages result in significantly improved TLB performance for GraphChi, with L1 DTLB misses dropping down to 11.105. (`MADV_FREE` does not have a major effect here.) In this case, 72.469% of L2 hits are handled by 2 MB entries, while 17.712% use 16 KB entries. The majority of L2 misses result in the creation of coalesced entries in the TLB, again indicating that PTE Coalescing can still provide benefits when 2 MB superpages are also used.

Reservations again outperform Linux with THP turned off, primarily due to their ability to produce many more coalesced hits. With THP on, however, Linux achieves much better TLB performance than FreeBSD does, with 4.431 L1 misses versus 8.207 on FreeBSD with superpages. In this case, Linux produces a much greater percentage of L2 DTLB hits (92.846%) than FreeBSD (62.445%).

5.1.3 *BlockSVM.*

Enabling reservations without superpages results in a modest improvement in BlockSVM's DTLB performance. L1 misses fall from 6.821 to 6.257, and L2 misses fall from 5.309 to

3.991. In this case, the ability of coalesced entries to replace 4 KB entries is very apparent. The entries used by L2 hits and misses go from being almost entirely 4 KB in size to nearly all 16 KB pages.

Switching on superpages has a greater impact than just enabling reservations. L1 misses are reduced to 3.493, with `MADV_FREE` having almost no effect. L2 misses are almost eliminated entirely, indicating that superpages allow BlockSVM's working set to mostly be covered by the TLB. Around three quarters of L2 hits are handled by 2 MB entries, with the rest making use of 16 KB entries.

Generally, the story is similar when comparing Linux and FreeBSD. In terms of L1 misses, FreeBSD without reservations (11.335) performs slightly worse than Linux without THP (11.321) due to differences in the buddy allocator's ability to create coincidental contiguity, but FreeBSD with reservations on outperforms both (10.928). FreeBSD's superpage promotion is more effective at eliminating L1 misses than THP, but both have a significant impact. Again, FreeBSD is able to reduce L2 misses to a negligible amount.

5.1.4 Freqmine. Freqmine's working set is smaller than that of the other applications, and it hence incurrs very few L1 DTLB misses. Nevertheless, enabling reservations decreases the count by nearly two thirds, from 0.297 to 0.091. Almost all of these misses subsequently hit in the L2, with roughly three quarters of these hits resulting from coalesced entries.

The use of 2 MB superpages again reduces L1 misses by two thirds to 0.034. L2 hits are handled by both 4 KB and 16 KB entries at roughly the same rate, and the relative lack of 2 MB L2 hits indicates that the 2 MB entries needed to address the application's working set are contained almost entirely in the L1 TLB.

For the Linux binary, native Linux outperforms FreeBSD (with reservations on or off) in terms of L1 misses, but FreeBSD is nonetheless able to create more coalesced L2 hits than Linux. In addition, THP is slightly more successful at eliminating L1 misses than FreeBSD superpages. It bears repeating that the differences here are quite small since Freqmine makes very good use of the TLB.

5.1.5 Node.js. Reservations decrease the number of L1 DTLB misses from 8.039 to 6.753. Since our workload has a relatively small working set, almost all of these L1 misses end up hitting in the L2. FreeBSD does not achieve a particularly high proportion of coalesced L2 hits (16.928%) in this case, but PTE Coalescing still allows the L1 TLB to cover more of the address space and thus miss less often.

Turning on superpages does not result in a noticeable difference in DTLB performance for Node.js. This is likely because Node.js tends to manage anonymous memory in blocks that are smaller than 2 MB in size. FreeBSD still allocates reservations for these mappings since they are heap memory, allowing PTE Coalescing to occur even though superpage promotions do not. Although this has the potential to cause physical memory fragmentation, if the system comes under memory pressure these reservations can be easily broken to reclaim contiguity.

A similar pattern appears when comparing DTLB performance with the Linux binary. The value of FreeBSD's reservation system is clearly apparent; Linux's THP is unable to achieve any performance benefit due to Node.js's memory management behavior, but reservations are able to reduce L1 DTLB misses from 7.781 to 5.629 by creating the contiguity that is necessary for PTE Coalescing much more effectively than Linux can.

5.1.6 QEMU. QEMU's DTLB performance resembles that of Clang. Turning on reservations drops the number of L1 misses from 3.853 to 2.073, which mirrors the drop in L2 hits from 3.562 to 1.913. L2 hits shift from being nearly 100% 4 KB to being around 80% 16 KB.

2 MB superpages further reduce L1 misses to 0.831, and disabling `MADV_FREE` drops the count to 0.767. In either case, L2 misses are eliminated almost entirely. L2 hits are handled mostly by 2 MB entries, with coalesced and 4 KB entries taking care of the rest.

Compared to Linux without THP, FreeBSD reservations are again very effective at creating the contiguity needed for PTE Coalescing. Superpage promotion on FreeBSD outperforms THP by a small amount (0.800 versus 0.836).

5.1.7 XSBench. With reservations disabled, XSBench makes exceptionally poor use of the data TLB. Enabling them reduces the L1 DTLB miss rate from 30.760 to 27.093 and the L2 DTLB miss rate from 24.803 to 20.845, with the vast majority of L2 hits and misses making use of coalesced mappings. This indicates that the coalescing size is simply not large enough to allow either the L1 or L2 DTLB to cover the application's working set.

Activating 2 MB superpages has a massive impact on DTLB performance, with the L1 miss rate dropping all the way down to 5.731. Nearly 90% of L2 DTLB hits and misses use 2 MB mappings in this case. Because XSBench exhibits good superpage-granularity locality, the L1 and L2 DTLB are able to mostly cover its working set with superpages enabled.

FreeBSD reservations again enable more coalescing than Linux with THP disabled does, but with THP on, Linux's more aggressive superpage creation policy slightly outperforms FreeBSD's approach (4.486 L1 misses versus 4.781).

5.1.8 Canneal. Enabling reservations has a fairly small impact on Canneal's L1 DTLB performance, with the miss rate dropping from 3.679 to 3.559, but the L2 hit rate nearly doubles from 0.692 to 1.236.

Turning on 2 MB superpages cuts the L1 DTLB miss rate by over 50%, to 1.666 and allows the application's working set to fit within the L1 and L2 TLB, nearly eliminating L2 misses entirely. Almost 80% of L2 hits make use of 2 MB superpage mappings.

The Linux binary for Canneal performs the best on FreeBSD with superpages enabled, with a L1 DTLB miss rate of 2.248 compared to 2.658 on Linux with THP. Again, FreeBSD's reservation-based allocation is able to create many more coalesced superpages than Linux's buddy allocator.

5.2 Instruction TLB

PTE Coalescing has a noticeable impact on applications which use a large number of code pages. GraphChi, BlockSVM, Freqmine, XSBench, and Canneal do not, and hence are not addressed below. We do not explicitly summarize the performance of Linux's THP here, as THP does not by default affect code pages.

It is important to note that all of our applications on FreeBSD were compiled with `maxpagesize` equal to 2 MB. If this were not the case, the misalignment explained in Section 2.3 would prevent both PTE Coalescing and superpage promotion from functioning at all.

5.2.1 *Clang (SQLite).*
Enabling reservations reduces L1 ITLB misses by over 50%, from 3.279 to 1.578, while L2 ITLB misses are reduced nearly to zero. Interestingly, enabling superpages has little impact beyond what PTE Coalescing is already able to accomplish given the contiguity created by reservations.

5.2.2 *Node.js.*
Turning reservations on cuts L1 ITLB misses from 0.676 to 0.133, a reduction of over 80%. There are very few L2 misses in ether case, suggesting that the code pages used (those of Node.js itself as well as any code that is just-in-time compiled) are well-covered by the L1 ITLB. Superpages actually increase the L1 miss count slightly, though this is somewhat mitigated by disabling `MADV_FREE`.

5.2.3 *QEMU.*
L1 ITLB misses drop from 1.544 to 0.614 when FreeBSD's reservations are activated. In contrast to the prior two applications, 2 MB code superpages have a strong impact on ITLB performance for QEMU, reducing L1 misses to 0.040 and essentially eliminating L2 misses.

5.3 Overall Performance

We use unhalted userspace clock cycles to measure performance in a way that is independent of any time needed for I/O. Node.js is not mentioned below since PTE Coalescing and superpages have very little impact on its performance in terms of cycles.

Although FreeBSD's Linux emulation layer may impose a performance penalty, prior work [40] suggests this overhead is small enough that a fair comparison can still be made between the FreeBSD and Linux kernels.

It is important to note that on an operating system which employs reservation-based allocation (like FreeBSD), the performance benefits of PTE Coalescing come at *zero cost*; no kernel modifications are required. Reservation based allocation has been shown [40] to be a competitive approach to superpage creation with a reasonable overhead.

5.3.1 *Clang (SQLite).*
Enabling reservations reduces the unhalted cycle count by 1.5%, and turning on superpages with `MADV_FREE` increases this figure to 2.1%. FreeBSD with reservations enabled outperforms Linux with THP off by 1.5%, and FreeBSD with superpages takes 2.8% fewer cycles than Linux with THP on.

5.3.2 *GraphChi.*
Reservations reduce the cycles needed by 1.1%, and 2 MB superpages offer a reduction of 6.7%, with disabling `MADV_FREE` leading to a negligible decline in performance. Linux substantially outperforms FreeBSD's Linux emulation, with THP taking 27.9% fewer cycles than FreeBSD superpages.

5.3.3 *BlockSVM.*
Although reservations have very little impact on performance in terms of cycles (around 0.6%), superpages have a large impact, reducing the unhalted cycle count by 10.2%. For the Linux binary, FreeBSD with superpages offers the best performance overall.

5.3.4 *Freqmine.*
Performance for Freqmine does not vary significantly based on the selected reservation or superpage policy. However, Linux achieves a significant overall performance advantage over FreeBSD. We believe that this can be attributed to unusually poor instruction cache behavior under FreeBSD that is unrelated to TLB performance. For example, comparing Linux with THP off to FreeBSD's Linux emulation with reservations disabled, FreeBSD incurs nearly 4.6 times as many instruction cache stalls. The same relationship holds for the native FreeBSD binary as well. Because the Freqmine executable is small, FreeBSD does not back it with a reservation, and its code pages are hence not guaranteed to be physically contiguous, which may lead to poor cache utilization.

5.3.5 *QEMU.*
Reservations speed up QEMU by 3.4%, and superpages increase this boost to 6.0%, with `MADV_FREE` not having a significant effect. Under FreeBSD's Linux emulation, FreeBSD with reservations outperforms Linux with THP off by 2.4%, and FreeBSD with superpages outpaces Linux with THP by 0.9%.

5.3.6 *XSBench.*
The coalescing enabled by reservations speeds up by XSBench by a small amount, while superpages result in a 7.1% reduction in clock cycles. Linux achieves a notable performance advantage over FreeBSD, and just as in the case of Freqmine above, we believe that this is explained by poor instruction cache behavior. FreeBSD with reservations disabled encounters 1.5 times as many instruction cache stalls as does Linux with THP off.

5.3.7 *Canneal.*
The benchmark's performance is not significantly impacted by enabling reservations, with the slight increase in unhalted clock cycles from 1.076 to 1.078 being within the variance of the experiment. Superpages do have an impact, however, reducing the cycle count by 7.1%. Running the Linux binary on FreeBSD with superpages on is actually the most performant configuration we examined, outperforming Linux with THP on by 6.2% and the FreeBSD native binary with superpages by 23.7%.

5.4 Coalescing Region Size

The older Ryzen 2700X's Zen+ microarchitecture coalesces 8 PTEs together at a time to form 32 KB pages instead of the 16 KB pages created by the Ryzen 5900X.

For the data TLB, 32 KB coalescing generally results in a greater percentage decrease in L1 misses after reservations are enabled. For example, enabling reservations causes Node.js's L1 DTLB misses to drop by 21.1% (from 5.960 to 4.705) on the 2700X but only by 16.0% (from 8.039 to 6.753) on the 5900X.

This pattern reverses for the instruction TLB, at least for the large applications that cannot be covered entirely by the L1. Again using Node.js as an example, L1 misses decrease by 60.9% (from 0.561 to 0.219) on the 2700X and by 80.25% (from 0.676 to 0.133) on the 5900X.

A potential explanation for this phenomenon is that code pages in large applications may be sparsely accessed when certain code paths are left unused throughout a program's entire execution, while anonymous memory tends to all be touched at least once. In the former case, a smaller coalescing size makes it more likely that all of the constituent PTEs will be accessed, allowing them to be coalesced together; in the latter case, larger entries allow the TLB to cover a larger amount of memory.

It is also worth mentioning that, in contrast, 16 KB coalescing clearly allows for a greater proportion of L2 DTLB hits to be handled using coalesced entries than 32 KB coalescing. For instance, with reservations enabled, 74.352% of Clang's L2 hits make use of coalesced entries on the 5900X, versus only 58.551% on the 2700X.

5.5 Discussion

To summarize, on the data side, enabling reservations generally reduces the number of L1 TLB misses by creating more of the contiguity that is necessary for PTE Coalescing to occur. In tandem, the proportion of L2 TLB hits that use coalesced pages usually increases substantially. Turning on superpages has the potential to improve L1 TLB performance further, but the feature's effectiveness varies from application to application and in many cases the vast majority of the performance improvement that superpages bring can be achieved using only reservations and PTE Coalescing.

The Linux buddy allocator does a better job generating coincidental contiguity than does FreeBSD with reservations disabled, but turning on reservations allows FreeBSD to far surpass Linux in terms of contiguity production.

For instructions, when considering applications whose code cannot be covered by the L1 ITLB using 4 KB entries, a similar pattern as for data holds. With the exception of QEMU, PTE Coalescing and reservations are able to achieve most of the ITLB performance benefits of 2 MB superpages.

In general, the improvements in unhalted clock cycles offered by reservations and coalescing are modest, with superpages tending to make a larger difference.

Finally, comparing results from the 2700X and 5900X indicates that 16 KB coalescing results in the creation of more coalesced entries, but that on the data side larger 32 KB coalesced pages may do more to alleviate L1 TLB misses because they allow the TLB to cover more of an application's working set.

6 RELATED WORK

In contrast to FreeBSD's current reservation-based allocator, Navarro et al. demonstrated a *multi-level* reservation-based allocator that supported the automatic creation of both medium and large superpages (64 KB, 512 KB, and 4 MB) on a DEC Alpha processor [30]. However, most of the subsequent work on the operating system-side of transparent superpage support has focused on the huge page sizes that are supported by x86-64 processors, frequently seeking to address the issue of memory fragmentation [21, 26, 31, 32, 40]. Today, in the absence of operating system support for medium-sized superpages, some ARMv8-A-based systems (such as Mac computers with Apple silicon) are configured with a larger base page size of 16 or even 64 KB.

Other techniques for creating physical contiguity exist that do not rely on reservation-style allocation. One example is Translation Ranger [37], which introduces the concept of an "anchor point," or a paired virtual and physical page, around which it attempts to build contiguity. It achieves this by swapping interfering physical pages out of the way in order to minimize the number of physically contiguous regions that are needed to cover each virtual address range.

Another example of hardware support for explicit medium-sized superpage creation is the Svnapot extension to the RISC-V architecture [5]. In its currently standardized form, the extension allows for 64 KB superpages to be created analogously to ARM's Contiguous bit. However, the scheme is more general than ARM's Contiguous bit, allowing for different page sizes to be specified in the number of trailing zeros in each PTE's physical page frame number.

This encoding technique is also used by the Tailored Page Sizes (TPS) mechanism proposed by Guvenilir et al. [23]. TPS goes further than the RISC-V approach by mandating support for aligned superpages with arbitrary power-of-two sizes. It also introduces the concept of "alias" PTEs, which act as pointers to a "true" superpage PTE and do not need to contain all of the attributes of the superpage.

Similar to our investigation into the effects of PTE Coalescing on the instruction TLB, Zhou et al. [39] investigate the impact of automatic support for superpages on improving the performance of virtual-to-physical address translation when fetching instructions within large executables.

Most of the work on coalesced TLBs since Pham et al.'s original CoLT paper [35] has focused on relaxing the contiguity requirement. For example, Pham et al. extend CoLT to exploit so-called "clustered spatial locality" by mapping clusters of virtual pages to clusters of physical pages without a strict contiguity requirement [34]. Du et al. propose gap-tolerant sequential mapping, which allows superpages to be formed even in the presence of "retired" physical pages, i.e., physical pages that are no longer allocated because of a hardware failure [20]. Park et al. propose Hybrid TLB Coalescing, which can dynamically change sizes for translation coalescing in order to adapt to the currently available physical contiguity, and so the operating system does not need to provide any particular allocation size [33].

7 CONCLUSIONS

This paper has investigated the behavior of the PTE Coalescing feature of modern AMD x86-64 microarchitectures. PTE Coalescing allows MMUs to store four or eight (depending on the generation) page table entries which map aligned and contiguous physical memory with identical attributes using only a single TLB entry, essentially offering transparent medium-sized superpage support. Because official documentation about the feature from AMD is limited, we developed a custom trace-driven microbenchmark in order to develop a full specification of its behavior. We then examined the extent to which FreeBSD's reservation system can be used to explicitly create alignment and contiguity beyond what arises coincidentally due to the use of the buddy allocators, and compared the effects of reservations to multiple techniques for creating 2 MB superpages. This was accomplished by benchmarking a variety of applications using different TLB-related CPU performance counters.

At a high level, reservations and PTE Coalescing combine to form an effective technique for improving TLB performance through medium-sized superpage creation. Reservations do a much better job at creating physically contiguous memory than either the FreeBSD or Linux buddy allocators are able to do coincidentally. In some cases, when reservations are activated, PTE Coalescing is able to provide most of the benefits of 2 MB superpages for both the data and instruction TLBs at zero additional cost. Enabling reservations leads to modest but significant increases in overall application performance, and in some cases 2 MB superpages can further speed up applications. AMD's decision to reduce the coalescing size from

32 KB to 16 KB on Zen 2 and later processors led to an increase in the number of coalesced entries used by the TLB, at the cost of not allowing each coalesced entry to cover as much memory.

A number of opportunities for further research into reservations, PTE Coalescing, and related virtual memory techniques exist.

For example, the smallest reservation size that modern FreeBSD systems create is 2 MB. As a consequence of this, if a file, executable, or shared library is less than 2 MB in size, a reservation will not be created to back it. On systems that support 16 KB or 32 KB medium-sized superpages through PTE Coalescing, it may make sense to additionally support smaller reservation sizes in order to improve the feature's effectiveness. (This approach could also be used by operating systems which support explicit medium-sized superpage creation with the ARMv8-A architecture's Contiguous bit.) The reservation-based allocator proposed by Navarro et al. [30] implemented a multi-level reservation system that supported this capability, but modern FreeBSD ships with a simplified version which supports multiprocessor systems but only offers a single reservation size.

Also, this paper has determined that all of the accessed bits within a coalescing region must be set in order to create a coalesced TLB entry. This presents the possibility of presetting the accessed bit on the prefaulted mappings which are speculatively created when a cluster of pages is brought into memory following a disk I/O operation. Doing so would allow coalescing to occur on the first access to the relevant virtual memory region instead of deferring it until all of the relevant PTEs have been touched.

Finally, prior work like Quicksilver [40] has suggested modifying FreeBSD's superpage promotion policy to make it more aggressive by reducing the number of PTEs within a 2 MB region that must be valid in order to attempt a superpage promotion. Currently, FreeBSD requires that all 512 constituent entries must exist, while Quicksilver proposes cutting this threshold to 64. It is possible that this change may need to be reconsidered in light of the existence of PTE Coalescing, because the latter allows much of the TLB performance benefit of 2 MB pages to be achieved before a 2 MB region is actually promoted into a superpage.

REFERENCES

[1] 2016. React server-side rendering benchmark. https://www.npmjs.com/package/react-ssr-benchmarks.

[2] 2019. Processor Programming Reference (PPR) for AMD Family 17h Models 01h,08h, Revision B2 Processors.

[3] 2020. Software Optimization Guide for AMD Family 19h Processors, Revision 3.00. https://www.amd.com/en/support/tech-docs/56665-software-optimization-guide-for-amd-family-19h-processors-pub.

[4] 2021. Processor Programming Reference (PPR) for AMD Family 19h Model 21h, Revision B0 Processors.

[5] 2021. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture.

[6] 2021. Software Optimization Guide for AMD Family 17h Processors, Revision 3.01. https://www.amd.com/en/support/tech-docs/software-optimization-guide-for-amd-family-17h-processors.

[7] 2023. Arm Architecture Reference Manual for A-profile architecture.

[8] 2023. The FreeBSD Handbook: Linux Binary Compatibility. https://docs.freebsd.org/en/books/handbook/linuxemu/.

[9] 2023. Node.js: an open-source, cross-platform JavaScript runtime environment. https://nodejs.org.

[10] 2023. perf events. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/perf/pmu-events/arch/x86.

[11] 2023. perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page.

[12] 2023. pmcstat - configure processor sets. https://man.freebsd.org/cgi/man.cgi?query=cpuset&sektion=1.

[13] 2023. pmcstat - performance measurementwith performance monitoring hardware. https://man.freebsd.org/cgi/man.cgi?query=pmcstat&sektion=8.

[14] 2023. pmcstat events. https://cgit.freebsd.org/src/tree/lib/libpmc/pmu-events/arch/x86.

[15] 2023. QEMU. https://www.qemu.org/.

[16] 2023. The SQLite Amalgamation. https://www.sqlite.org/amalgamation.html.

[17] 2023. taskset - set or retrieve a process's CPU affinity. https://www.man7.org/linux/man-pages/man1/taskset.1.html.

[18] 2023. V8. https://v8.dev.

[19] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques* (Toronto, Ontario, Canada) *(PACT '08)*. Association for Computing Machinery, New York, NY, USA, 72–81. https://doi.org/10.1145/1454115.1454128

[20] Y. Du, M. Zhou, B. R. Childers, and D. Mossénd R. Melhem. 2015. Supporting superpages in non-contiguous physical memory. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 223–234. https://doi.org/10.1109/HPCA.2015.7056035

[21] Mel Gorman and Patrick Healy. 2008. Supporting Superpage Allocation Without Additional Hardware Support. In *Proceedings of the 7th International Symposium on Memory Management* (Tucson, AZ, USA) *(ISMM '08)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/1375634.1375641

[22] G. Grahne and J. Zhu. 2003. Efficiently Using Prefix-trees in Mining Frequent Itemsets. (2003).

[23] Faruk Guvenilir and Yale N. Patt. 2020. Tailored Page Sizes. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 900–912. https://doi.org/10.1109/ISCA45697.2020.00078

[24] S Ritter GJ Gordon J Stamper, A Niculescu-Mizil and KR Koedinger. 2010. Bridge to algebra 2008–2009 *(Challenge data set from KDD Cup)*.

[25] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 591–600. https://doi.org/10.1145/1772690.1772751

[26] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. 2016. Coordinated and Efficient Huge Page Management with Ingens. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 705–721.

[27] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 31–46.

[28] Linux hugepage text 2015. hugepage_text.cc. https://chromium.googlesource.com/experimental/chromium/src/+/refs/wip/bajones/webvr_1/chromeos/hugepage_text/hugepage_text.cc.

[29] H.J. Lu, Kshitij Doshi, Rohit Seth, and Jantz Tran. 2006. Using Hugetlbfs for Mapping Application Text Regions. In *Linux Symposium*.

[30] Juan Navarro, Sitararn Iyer, Peter Druschel, and Alan Cox. 2003. Practical, Transparent Operating System Support for Superpages. *SIGOPS Oper. Syst. Rev.* 36, SI (dec 2003), 89–104. https://doi.org/10.1145/844128.844138

[31] Ashish Panwar, Naman Patel, and K. Gopinath. 2016. A Case for Protecting Huge Pages from the Kernel. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems* (Hong Kong, Hong Kong) *(APSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 15, 8 pages. https://doi.org/10.1145/2967360.2967371

[32] Ashish Panwar, Aravinda Prasad, and K. Gopinath. 2018. Making Huge Pages Actually Useful. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. ACM, New York, NY, USA, 679–692. https://doi.org/10.1145/3173162.3173203

[33] C. H. Park, T. Heo, J. Jeong, and J. Huh. 2017. Hybrid TLB coalescing: Improving TLB translation coverage under diverse fragmented memory allocations. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 444–456. https://doi.org/10.1145/3079856.3080217

[34] Binh Pham, Abhishek Bhattacharjee, Yasuko Eckert, and Gabriel H. Loh. 2014. Increasing TLB reach by exploiting clustering in page translations. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 558–567. https://doi.org/10.1109/HPCA.2014.6835964

[35] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. 2012. CoLT: Coalesced Large-Reach TLBs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 258–269. https://doi.org/10.1109/MICRO.2012.32

[36] John Tramm, Andrew Siegel, Tanzima Islam, and Martin Schulz. 2014. XSBench - The development and verification of a performance abstraction for Monte Carlo reactor analysis.

[37] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Translation Ranger: Operating System Support for Contiguity-Aware TLBs. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 698–710. https://doi.org/10.1145/3307650.3322223

[38] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. 2012. Large Linear Classification When Data Cannot Fit in Memory. *ACM Trans. Knowl. Discov. Data* 5, 4, Article 23 (feb 2012), 23 pages. https://doi.org/10.1145/2086737.2086743

[39] Yufeng Zhou, Alan L. Cox, Sandhya Dwarkadas, and Xiaowan Dong. 2023. The Impact of Page Size and Microarchitecture on Instruction Address Translation Overhead. *ACM Trans. Archit. Code Optim.* 20, 3, Article 38 (jul 2023), 25 pages. https://doi.org/10.1145/3600089

[40] Weixi Zhu, Alan L. Cox, and Scott Rixner. 2020. A Comprehensive Analysis of Superpage Management Mechanisms and Policies. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 829–842. https://www.usenix.org/conference/atc20/presentation/zhu-weixi