Energy-Efficient Address Translation

Vasileios Karakostas [1,2] Jayneel Gandhi [5] Adrián Cristal [1,2,6] Mark D. Hill [5]
Kathryn S. McKinley [3] Mario Nemirovsky [4] Michael M. Swift [5] Osman S. Unsal [1]

[1]Barcelona Supercomputing Center [2]Universitat Politecnica de Catalunya
[3]Microsoft Research [4]ICREA Senior Research Professor at Barcelona Supercomputing Center
[5]University of Wisconsin - Madison [6]Spanish National Research Council (IIIA-CSIC)

{vasilis.karakostas, adrian.cristal, mario.nemirovsky, osman.unsal}@bsc.es
{jayneel, markhill, swift}@cs.wisc.edu, mckinley@microsoft.com

ABSTRACT

Address translation is fundamental to processor performance. Prior work focused on reducing Translation Lookaside Buffer (TLB) misses to improve performance and energy, whereas we show that even TLB hits consume a significant amount of dynamic energy.

To reduce the energy cost of address translation, we first propose *Lite*, a mechanism that monitors the performance and utility of L1 TLBs, and adaptively changes their sizes with way-disabling. The resulting TLB_{Lite} organization opportunistically reduces the dynamic energy spent in address translation by 23% on average with minimal impact on TLB miss cycles. To further reduce the energy and performance overheads of L1 TLBs, we also propose RMM_{Lite} that targets the recently proposed Redundant Memory Mappings (RMM) address-translation mechanism. RMM maps most of a process's address space with arbitrarily large ranges of contiguous pages in both virtual and physical address space using a modest number of entries in a range TLB. RMM_{Lite} adds to RMM an L1-range TLB and the Lite mechanism. The high hit ratio of the L1-range TLB allows Lite to downsize the L1-page TLBs more aggressively. RMM_{Lite} reduces the dynamic energy spent in address translation by 71% on average. Above the near-zero L2 TLB misses from RMM, RMM_{Lite} further reduces the overhead from L1 TLB misses by 99%.

These proposed designs target current and future energy-efficient memory system design to meet the ever increasing memory demands of applications.

1. INTRODUCTION

Processors employ Translation Lookaside Buffers (TLBs) to perform quick address translation on every memory operation. The TLB holds mappings from the virtual to the physical address space. Since their invention in the 1960s [19], TLBs have been a small monolithic structure and were able to deliver high performance. Commercial processors, however, keep on devoting more resources to memory and address translation to meet the ever increasing memory demands of memory intensive workloads. The common TLB organization found today includes multi-level TLBs with support for huge pages [6, 23, 48].

978-1-4673-9211-2/16/$31.00

Figure 1: A common per-core two-level TLB organization that supports multiple page sizes (4 KB, 2 MB, and 1 GB) through separate L1 TLBs. All L1 TLBs are accessed on every memory operation, increasing the dynamic energy spent in address translation.

TLBs have been reported to consume a significant fraction of processor energy [2, 4, 21, 31, 32, 33]. The recent growth in the complexity of TLBs has further increased their energy consumption—a recent industrial report suggests that TLBs consume 3-13% of a processor's power [49].

The energy overheads associated with the TLBs come from two sources: (i) the *static energy of the chip* due to TLB misses that lead to longer execution times [22, 35], and (ii) the *dynamic energy* of TLB resources that are accessed to lookup the address translation on every memory operation. However, reducing the energy of address translation is not straightforward. When the static energy of the chip decreases due to fewer TLB misses, the dynamic energy of the TLBs increases due to the augmented complexity that ensures the low TLB miss ratio.

Prior research focused on reducing the dynamic energy of TLBs through various techniques, such as optimizing TLB circuits [31], partitioning TLBs [10, 17, 18, 37], filtering accesses to TLBs [11, 17, 21], dynamically resizing monolithic TLBs [9], virtual caches to access TLBs on L1 cache misses [14, 29, 52], and selectively avoiding TLB accesses [32, 33, 34]. However, these energy optimization techniques do not take into account hardware support for increasing the TLB reach (e.g., huge pages), that primarily targets improving performance and reducing static energy overheads due to TLB misses. Only the recent work on TLB_{Pred} [41] considers huge pages for improving the dynamic energy efficiency in TLBs. The performance of TLB_{Pred} depends on huge pages successfully reducing misses, but prior work shows that huge pages can still incur high performance overheads due to TLB misses [13, 15, 36]. In response, researchers proposed techniques that further increase the TLB reach [13, 22, 35, 42, 43, 50] to overcome the limitations of huge pages.

The goal of this work is to improve the energy efficiency of address translation in the presence of mechanisms that increase TLB reach.

Towards that goal, we perform energy characterization of the address translation path. We use a common TLB organization, found in Intel x86-64 processors as our baseline, that includes a per-core two-level TLB hierarchy, with a separate set associative L1 TLB for each supported page size, e.g., for 4 KB, 2 MB, and 1 GB pages, as shown in Figure 1. Our analysis shows that the L1 TLBs are the primary source of dynamic energy spent in address translation. We also find that page walks consume significant amount of energy with 4 KB pages. While huge pages and other techniques that increase TLB reach [13, 22, 35, 42, 43, 50] reduce the energy due to page walks, we observe that the "innocent" L1 TLB remains the dominant source of dynamic address translation energy, because separate L1 TLBs are accessed on every memory operation.

Our approach for providing energy-efficient address translation is driven by the following key observation: simply accessing all L1 TLB resources might not improve performance, because not all L1 TLBs contribute the same to hits, especially when techniques that increase the TLB reach are employed.

We propose *Lite*, an opportunistic mechanism that targets commodity processors with TLB support for huge pages. Lite monitors the utility of ways in the L1 TLBs for each page size based on the distance of TLB hits from the least-recently-used (LRU) position in an interval fashion, similar to the accounting cache [20] and utility-based cache partitioning [46]. At the end of each interval, Lite evaluates the utility of L1 TLBs. In case the utility of some active ways is insignificant, Lite downsizes each L1 TLB individually by disabling ways [8]. Lite thus accesses fewer ways in the L1 TLBs, saving energy at the cost of introducing a few additional misses. The resulting TLB_{Lite} organization requires minimal modifications and opportunistically reduces L1 TLB energy with negligible impact on performance.

We additionally propose RMM_{Lite} to reduce the energy further with Redundant Memory Mappings (RMM), a recent proposal for reducing page walks [35]. RMM provides support for arbitrarily large range translations, i.e., range of pages that are contiguous in both virtual and physical address space with same page protections. Prior work considered only L2-range TLBs. We introduce to RMM an L1-range TLB and add the Lite resizing mechanism to the L1-page TLBs. The L1-range TLB is accessed in parallel with the L1-page TLBs and is small (e.g., 4 entries) in order to meet the tight timing requirements of L1 TLBs, yet the L1-range TLB is powerful. Each range TLB entry can hold a mapping of unlimited size, that enables the L1-range TLB to enjoy a high hit ratio. Therefore, Lite downsizes L1-page TLBs more aggressively without affecting performance. Overall, RMM_{Lite} improves both energy efficiency and performance of address translation.

To evaluate the proposed TLB_{Lite} and RMM_{Lite} designs, we developed a TLB simulator based on Pin [39], `pagemap` [3], and Cacti [38] and we ran various TLB intensive workloads from Spec2006 [26], BioBench [7], and Parsec [16]. Our findings show that TLB_{Lite} reduces the dynamic energy spent in address translation by 23% while slightly increasing the cycles spent in TLB misses (from 16.6% to 17.2%) compared to huge pages [5]. RMM_{Lite} reduces the dynamic energy spent in address translation by 71% on average compared to huge pages. Above the near-zero L2 TLB misses from RMM, RMM_{Lite} further reduces the overhead from L1 TLB misses by 99%.

In summary, the main contributions of this paper are:

- We characterize the dynamic energy spent in address translation, and identify the L1 TLBs and page walks as the most significant sources.
- We show that simply accessing all L1 TLBs might not improve performance in the presence of huge pages.
- We propose Lite to reduce the energy spent in L1 TLBs by opportunistically disabling resources with low impact on performance, and apply it to a standard TLB hierarchy with support for huge pages (TLB_{Lite}).
- We propose RMM_{Lite}, that adds to RMM an L1-range TLB and Lite, to reduce further the energy and performance overheads spent in L1 TLBs leveraging the efficient representation of range translations.

2. BACKGROUND

This section provides background on address translation, highlights some trends for improving TLB performance, and then outlines some characteristics that are found in commodity processors. Note that although we focus on the x86-64 architecture, the proposed solutions apply to other architectures that include TLB support for huge pages.

2.1 Address Translation Hardware Support

With virtual memory each process sees a vast amount of memory. The operating system divides the physical memory into pages and allocates them to different processes [25].

Page Table. The page table implements the abstraction of virtual memory, storing in memory all the translations from virtual to physical address space for each process as a four-level hierarchical radix tree in the x86-64 architecture [30].

Translation Lookaside Buffer (TLB). To accelerate the translation of virtual to physical addresses, processors employ a Translation Lookaside Buffer (TLB). The TLB caches recently used page table entries. The TLB is consulted on every memory operation, and is on the critical path of accessing the memory hierarchy. In case of a TLB miss, a hardware state machine walks the page table, a process named *page walk*, and fetches the corresponding page table entry from memory. Thus, the TLB is the most crucial component for accelerating virtual memory, and its miss ratio significantly affects the performance of the processor [13, 15, 30, 36].

MMU cache. Due to the performance impact of page walks, the TLB is backed by a memory management unit (MMU) cache [12, 15, 27]. The MMU cache reduces the latency of page walks by caching intermediate levels of the page table. A hit in the MMU cache eliminates one or more levels of a page walk. Thus, a page walk requires between one and four memory operations.

We use the term "TLB" to refer either to the general structure of the TLB or to the TLB hierarchy as a whole depending on the context, whereas we use L1 or L2 TLBs to refer explicitly to that (L1 or L2) level's TLBs.

L1 DTLBs			L2 DTLBs								
Sandy Bridge / Haswell / Broadwell			Sandy Bridge			Haswell			Broadwell		
Page-size	Entries	Assoc.	Page-size	Entries	Assoc.	Page-size	Entries	Assoc.	Page-size	Entries	Assoc.
4 KB	64	4-way	4 KB	512	4-way	4 KB/2 MB	1024	8-way	4 KB/2 MB	1536	n/a
2 MB	32	4-way	2 MB	—							
1 GB	4	fully	1 GB	—		1 GB	—		1 GB	16	n/a

Table 1: Details of the private, per-core, data TLB hierarchy for the three latest Intel processor architectures.

2.2 Trends in TLBs

Two-level TLBs form a common organization for address translation in today's processors [23, 48]. The TLB organization is per-core. The L1 TLB is usually small (e.g., 64 entries) and features a very fast search operation (1-2 cycles), while the L2 TLB is usually larger (e.g., 512 entries) and holds more translations at the cost of increased access latency ($\sim$7 cycles [28]). To boost the performance further, processors provide separate TLBs for data and instructions.

Huge Pages [1, 5] increase the TLB reach and reduce the performance overhead of page walks [13, 15, 36, 41] by mapping a large fixed size region of memory with a single TLB entry [27, 40, 45, 48]. For example, the x86-64 architecture allows a process to use 4 KB pages with 2 MB pages and 1 GB pages at the same time. The hardware support for huge pages usually includes either a separate set associative L1 TLB for each page size, as in Intel processors [23], or a single fully associative L1 TLB that supports both 4 KB and huge pages, as in SPARC and AMD processors [6, 48]. These two approaches dominate because supporting all page sizes in a single set associative TLB is not straight-forward: the page size defines the index bits to access the TLB, but the page size is unknown during the TLB lookup time [41, 51]. Separate set associative TLBs are generally more energy-efficient as compared to fully associative TLB. As we focus on energy, our baseline in this work assumes the more efficient separate set associative L1 TLBs.

Redundant Memory Mappings (RMM) [35] is a recent proposal that increases the TLB reach and reduces the number of page walks transparently to applications. RMM has some similarity to other approaches that pack multiple page table entries into a single TLB entry, such as sub-blocked TLBs [50], CoLT [43], Clustered TLBs [42], and Direct Segments [13, 22]. What distinguishes RMM from those schemes is the concept of *range translation*: an arbitrarily large range of pages that are contiguously allocated in both virtual and physical address space. The range translations provide an efficient alternative representation—in addition to paging—that allow RMM to map most of the process's address space with a modest number of entries. In this way, RMM reduces robustly most page walks, compared to other schemes [35]. RMM introduces a software managed range table to store per-process range translations and an L2-range TLB that is accessed in parallel with the L2-page TLB. To increase the likelihood and contiguity of range translations, RMM introduces eager paging in the operating system that allocates memory in ranges at request time.

2.3 Summary

Table 1 overviews the details of the per-core TLB hierarchy for the recent Sandy Bridge and Haswell, and the forthcoming Broadwell x86-64 processors. We observe that all these processors have a two-level TLB organization, with support for various pages sizes by separate individual L1 TLBs. This data suggests their recipe for improving TLB performance consists of having separate L1 TLBs for various page sizes, and increasing the size of the L2 TLB which is off the critical path.

To summarize, the TLB resources become larger and more complex to meet the increasing TLB demands of memory intensive workloads. However the performance and static energy improvements come at the cost of accessing multiple structures and increasing dynamic energy. Our approach reduces the dynamic energy spent in address translation by leveraging mechanisms that increase TLB reach, such as huge pages and range translations, making the case for energy-efficient address translation.

3. ENERGY CHARACTERIZATION

In this section we analyze the sources of dynamic energy spent in address translation. We first provide an overview of our methodology, and then we analyze where the dynamic energy is spent with 4 KB pages, huge pages, and RMM.

3.1 Methodology Overview

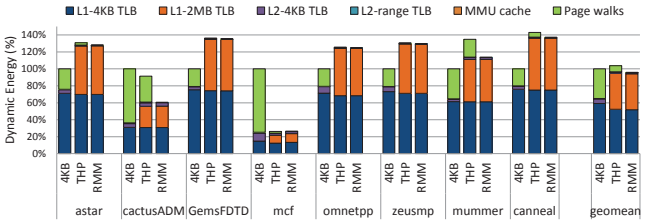
For the purposes of this study, we developed a detailed TLB simulator based on Pin [39], `pagemap` [3], and Cacti [38]. We model a private, per-core, two-level TLB organization backed by an MMU cache. The configuration and the parameters are based on those of an x86-64 Intel Sandy Bridge processor and summarized in Table 1.

We assume the existence of a mechanism that statically disables accesses to TLB resources that are not used. For instance, the L1-2MB TLB and L1-1GB TLB could be disabled for a running process that uses only 4 KB pages and no 2 MB or 1 GB pages. Such a mechanism could be easily implemented in hardware; a mask would enable lookups in the L1-2MB TLB only after a 2 MB page table entry has been fetched by a page walk. In this study we assume the existence of such mechanism, and thus unused TLB structures do not account for the dynamic energy overhead.

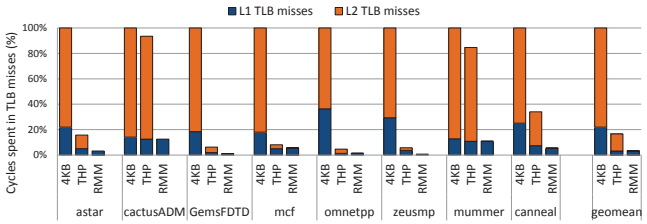
We fast-forward execution for 50 billion instructions and then simulate for the next 50 billion instructions. More details about our methodology can be found in Section 5.

3.2 Where is the energy spent?

Figures 2a and 2b break down the dynamic energy spent in address translation and the cycles spent in L1 and L2 TLB misses for various workloads with the following three configurations: (i) *4KB* supports 4 KB pages, (ii) *THP* supports both 4 KB and 2 MB pages with transparent huge pages [5], and (iii) *RMM* supports 4 KB, 2 MB pages, and range translations with a 32-entry fully associative L2-range TLB [35]. We assume optimistically that all page walk references hit



(a) Dynamic energy spent in address translation (%)



(b) Cycles spent in L1 and L2 TLB misses (%)

Figure 2: Dynamic energy spent in address translation (a) and cycles spent in TLB misses (b) with three configurations: (i) *4KB* supports only 4 KB pages, (ii) *THP* supports both 4 KB and 2 MB pages with transparent huge pages [5], and (iii) *RMM* supports 4 KB, 2 MB pages, and range translations with an L2-range TLB [35]. The results are normalized to those with 4 KB pages per workload. The two major sources of dynamic energy overhead with 4 KB pages are the L1 TLBs and the page walks. THP and RMM reduce the energy and cycles spent in page walks, but increase the total dynamic energy spent in address translation because multiple L1 TLBs are accessed on every memory operation.

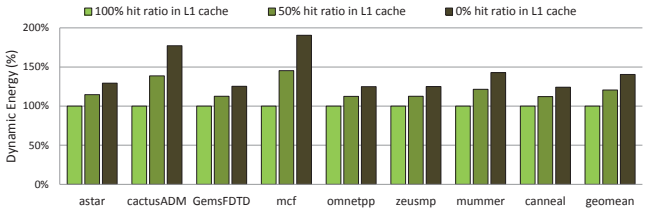


Figure 3: Sensitivity analysis of the dynamic energy spent in address translation, ranging the L1 cache hit ratio from 100% (all accesses hit in L1 cache) to 0% (all accesses miss in L1 cache but hit in L2 cache) for the page walk references with 4 KB pages. The locality of page walks significantly affects the dynamic energy.

always in the L1 cache of the memory hierarchy with respect to the dynamic energy. The results are normalized to the dynamic energy spent with 4KB pages per workload.

We identify two major sources of dynamic energy overhead with 4KB and THP configurations:

1. **L1 TLBs energy consumption**. To make address translation as fast as possible, the processor accesses *all L1 TLB structures*, i.e., the L1-4KB TLB, the L1-2MB TLB, and the L1-1GB TLB, in parallel on *every memory operation*. Consequently, the L1 TLBs consume 60% and 91% of dynamic energy with 4KB and THP. We further identify the L1-4KB TLB as the most dominant source of dynamic energy (50% of dynamic energy with THP) due to its larger size compared to the other L1-page TLBs.

2. **Page walk energy consumption**. On a TLB miss at every TLB level, the page table hardware walks the page table, which requires multiple memory accesses (e.g., 4, 3, and 2 memory accesses for 4 KB, 2 MB, and 1 GB pages) that incur performance and energy penalties. This source of energy overhead becomes more prevalent (i) for applications that suffer frequently from page walks, such as cactusADM and mcf, and (ii) as the page walk references hit less in the L1 cache. Figure 3 quantifies the impact of page walk locality in the dynamic energy as the L1 cache hit ratio for the page walk references reduces from 100% (all references hit in L1 cache) to 0% (all references miss in L1 cache, but hit in L2 cache). The dynamic energy may increase by up to 91% for mcf, due to poor page walk locality in the cache hierarchy.

3.3 Do huge pages help?

We observe that THP reduces the cycles spent in TLB misses by 83% on average compared to 4KB. However, THP affects the dynamic energy of address translation in a less straightforward way compared to performance. With THP, the dynamic energy of address translation decreases only for cactusADM and mcf, and increases for all other workloads. This happens because THP reduces the number of page walks and their portion in dynamic energy along with static energy by completing the workload faster, as explained next in Section 3.5. However, this saving occurs at the cost of accessing one extra L1 TLB for 2MB pages on every memory operation, which in turn increases the dynamic energy spent for address translation in the L1-page TLBs. Overall, THP increases the dynamic energy spent in address translation by up to 43% for canneal and by 4% on average, compared to 4 KB pages.

3.4 Does RMM help?

The RMM configuration has the same TLB organization as THP, including a 32-entry L2-range TLB. In addition, the RMM configuration assumes perfect eager paging, i.e., the operating system perfectly allocates all contiguous pages of virtual address space to contiguous physical pages. We observe that RMM eliminates almost completely the page walks, and reduces by 96% the cycles spent in TLB misses compared to 4KB. However, RMM incurs high dynamic energy overhead (only 4% less on average compared to 4KB), as the access pattern to the L1 TLBs is similar to THP.

3.5 Discussion

The total energy consumption is the sum of static and dynamic energy. Since huge pages and range translations (and other techniques that increase TLB reach [13, 22, 42, 43, 50]) enable most applications to execute faster, they also decrease the static energy of the system. However, optimizing for energy efficiency requires addressing both dynamic and static sources of energy. Thus, in addition to reducing the execution cycles and the static energy, in this paper we focus on reducing the dynamic energy spent in address translation.

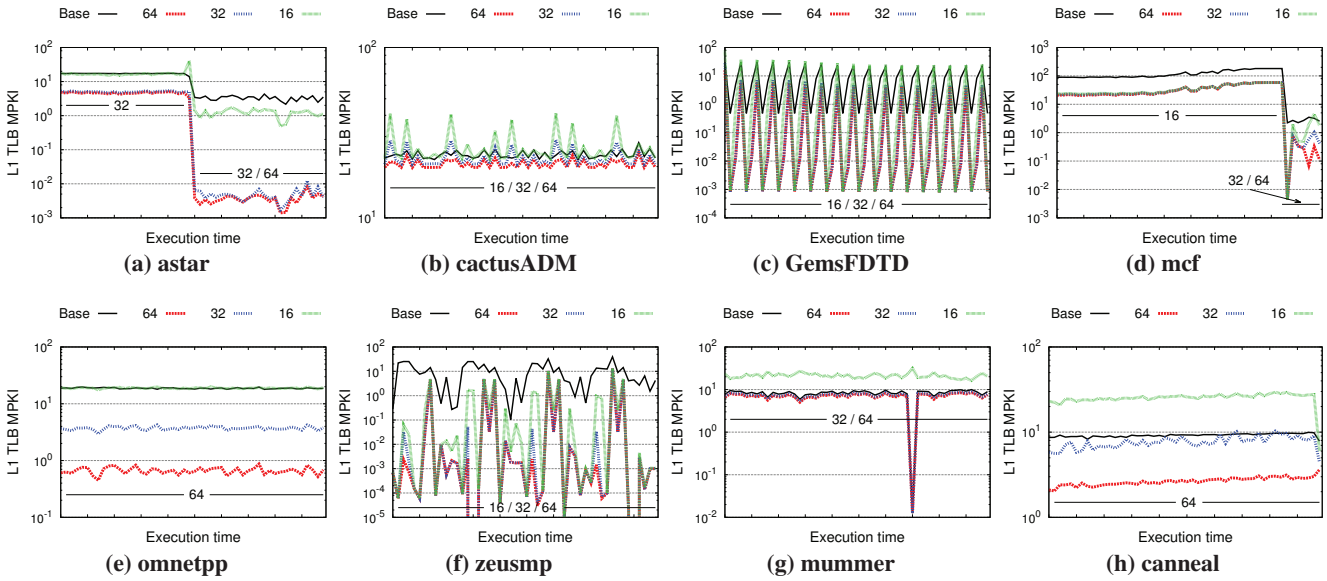


Figure 4: L1 TLB misses per thousand instructions (MPKI) (aggregated for all L1 TLBs) during the execution of 50 billion instructions with the following four configurations: (i) *Base* supports only 4 KB pages (same as *4KB* in Section 3), (ii) *64* supports both 4KB and 2 MB pages (same as *THP* in Section 3), (iii) *32* has the same configuration as *64* but with 32-entry 2-way L1-4KB TLB, and (iv) *16* has the same configuration as *64* but with 16-entry direct-mapped L1-4KB TLB. We observe that most workloads exhibit similar performance even with smaller L1-4KB TLBs in the presence of huge pages, but there is no single TLB configuration that is optimal for all workloads and during all execution time.

4. EFFICIENT ADDRESS TRANSLATION

An ideal solution for energy-efficient address translation would reduce the energy spent in L1 TLB accesses and page walks with negligible impact on performance. To provide energy-efficient address translation, we propose:

- **Lite**, a mechanism that monitors the utility of ways in all L1-page TLBs and adaptively changes their size with way-disabling [8]. The resulting **TLB**$_{Lite}$ organization opportunistically reduces L1 TLB energy with negligible impact on performance, and requires minimal modifications to commodity processors.

- **RMM**$_{Lite}$, a novel TLB organization that leverages the powerful representation of range translations in RMM [35]. RMM$_{Lite}$ adds an L1-range TLB and the Lite mechanism to RMM. The high hit ratio in the L1-range TLB allows Lite to further reduce the energy spent in L1-page TLBs and reduce significantly the total energy and performance overheads of L1 TLB misses.

4.1 Opportunity

Our approach is based on the question: *"Do we need to access all L1 TLB resources on every memory operation?"* For example, if the hits in L1 TLBs are dominated by those entries for 2 MB pages, then the L1-4KB TLB could be dynamically downsized to reduce the dynamic energy spent in L1 TLBs without affecting performance, and vice versa.

To quantify our hypothesis, we profile the performance of L1 TLBs with transparent huge pages enabled [5], when a smaller L1-4KB TLB with fixed size is employed during the execution. We assume that the L1-4KB TLB becomes smaller by reducing ways in powers-of-two while the number of sets remains constant. Figure 4 shows the misses in the L1 TLBs per thousand instructions (MPKI) during the execution of 50 billion instructions. Configurations *64*, *32*, and *16* employ a 64-entry 4-way, a 32-entry 2-way, and a 16-entry direct-mapped L1-4KB TLB, respectively. The L1-2MB TLB is 32-entry 4-way for all configurations.

We find that most workloads exhibit similar performance even with smaller L1-4KB TLBs in the presence of huge pages. However, there is no single TLB configuration that is optimal for all workloads. For example, astar and mcf require configuration *16*, while cactusADM, GemsFDTD, and mummer require configuration *32*, to provide similar performance as with configuration *64* that runs with all L1 TLB resources enabled. In addition, a single TLB configuration is often not the optimal during the workload's total execution due to phased behavior. For example, astar, GemsFDTD, and mcf require different configurations to preserve similar performance. Thus, a mechanism that dynamically resizes the L1 TLBs is required to adapt to the workload.

4.2 The *Lite* Mechanism

Lite dynamically adapts the size of L1 TLBs to reduce their dynamic energy. Lite consists of three components: (i) the *monitoring mechanism* that tracks the actual performance of L1 TLBs and estimates the utility of all L1 TLBs, (ii) the *decision algorithm* that decides whether to change the size of the L1 TLBs, and (iii) the *reconfiguration mechanism* that configures the size of L1 TLBs.

4.2.1 Monitoring TLBs

Lite tracks the performance of the L1 TLBs in the *actual-*

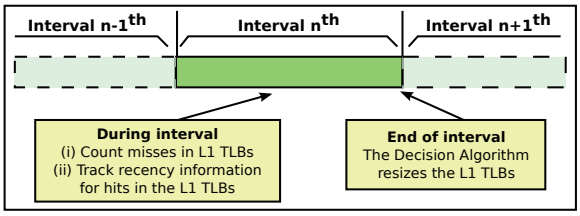


Figure 5: Lite divides the execution time of an application into *intervals*. During each interval, Lite tracks the performance of L1 TLBs. At the end of each interval, Lite decides whether to resize the L1 TLBs.

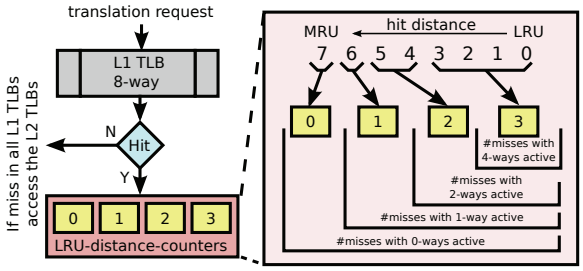


Figure 6: Lite introduces *lru-distance-counters* per L1 TLB to track the utility of ways [20, 46]. The corresponding counter is increased on each L1 TLB hit depending on the distance from the LRU position. At the interval end, each counter holds the number of L1 TLB misses that would have occurred, if those ways were disabled.

misses-counter for an interval. The counter is increased on every translation lookup that misses in L1 TLBs of that core and that triggers an access to the L2 TLB.

Lite estimates the cost of way-disabling by tracking the utility of all active ways for each L1 TLB in powers-of-two. Lite leverages the LRU replacement policy and relies on the distance of TLB hits from the LRU position in each set to estimate the utility of ways, similar to the accounting cache [20] and utility-based cache partitioning [46]. Lite introduces *lru-distance-counters* per L1 TLB. Since Lite disables ways in powers-of-two, we only need $[log_2(n)+1]$ lru-distance-counters for each n-way set-associative L1 TLB. Figure 6 shows Lite for an 8-way L1 TLB. The corresponding lru-distance-counter is increased on every L1 TLB hit: a hit with distance 7, 6, 4-5, or 0-3 from the LRU position increases the lru-distance-counters [0], [1], [2], or [3], correspondingly. In this way, each lru-distance-counter holds the number of TLB misses that would have occurred, if those ways were disabled. Note that when less ways are active, the corresponding lru-distance-counters are not used, because there are no tags to keep track of activity.

Finally, Lite keeps the actual number of L1 TLB misses of the previous interval in the *previous-misses-counter* to respond to TLB performance degradation, as explained next.

4.2.2 The Decision algorithm

Figure 7 shows the simplified pseudocode for the decision algorithm of Lite. Lite resizes all L1-page TLBs (4KB, 2MB, and 1GB) of each core's TLB organization separately. The algorithm uses the number of L1 TLB misses per thousand instructions (MPKI) to estimate the performance of the L1 TLBs and the utility of the active ways in each L1 TLB.

Disabling ways. At the end of each interval, Lite estimates for each L1-page TLB (4KB, 2MB, and 1GB) the potential MPKI if way disabling had been applied to the currently active ways. To achieve this, Lite uses the actual-misses-counter and the lru-distance-counters. In case the potential MPKI for fewer ways does not significantly increase compared to the actual MPKI, based on a threshold ε, then Lite disables those ways for that L1 TLB. During next interval, the resized L1 TLBs will save dynamic energy on every memory access.

Activating ways. Lite profiles only the active ways and therefore can reason only for them. However, Lite is unaware whether more than the active ways would be really useful. For example, consider an 8-way L1 TLB that currently runs with 4 active ways; Lite sees that the potential MPKI does not significantly change whether using 4 or 2 active ways, and decides to use 2 ways. However, if all 8 ways were active, the potential MPKI could be significantly lower and Lite would have not decided to apply way-disabling. The problem becomes even more prevalent when the 1-way configuration is used, as no alternatives are evaluated. To respond in such cases, Lite randomly activates all the ways in all L1 TLBs based on a probability. The randomly introduced variability allows also Lite to avoid pathological cases in which the decisions synchronize with non-representative phases of the application, that may lead to poor decisions.

Finally, Lite activates all ways in the L1 TLBs when their performance degrades, e.g., when the application experiences phased TLB behavior, or the operating system breaks huge pages to 4 KB pages to respond to memory pressure. Lite records the actual MPKI of the previous interval, and compares it to the actual MPKI of the current interval. In case the MPKI surpasses the defined threshold ε, Lite activates all ways in the L1 TLBs.

Threshold. The threshold ε can either be a relative percentage increase or an absolute value increase of MPKI with respect to the reference value, i.e., the MPKI with all ways activated in L1 TLBs. The choice depends on the reference value itself. A relative percentage is preferable for high reference values (e.g., more than 1 MPKI) to control Lite's impact. An absolute value is preferable for lower reference values, because even though the MPKI increases with respect to the reference value, the MPKI remains still negligible and, thus, Lite correctly decides to disable ways.

4.2.3 Reconfiguring TLBs

Lite reconfigures the L1 TLBs through way-disabling [8]. Way-disabling requires that the memory structure be partitioned into subarrays. We assume that such partitions are either already present in L1-TLBs for both timing and energy reasons, or can be easily implemented with minor circuit modifications. With way-disabling, only the active ways are searched in each TLB lookup, and thus the dynamic energy spent in address translation reduces.

Consistency. TLBs are read-only structures and do not hold dirty data. Thus, when Lite deactivates ways in a TLB, no write-back operations are necessary. Lite only invalidates translations in the disabled ways, so that when these ways are re-activated, they will not hold any stale translations.

```
// at the end of each interval;
compute the actual_mpki based on actual_misses_counter;
if (random probability is triggered) then
    activate all ways in L1 TLBs;
else if (actual_mpki - previous_mpki ≥ ε) then
    activate all ways in L1 TLBs;
else
    potential_misses = actual_misses;
    for each L1 TLB of that core do
        for (i = log2(active_ways); i ≥ 1; i–) do
            potential_misses += lru_distance_counter[i];
            compute the potential_mpki based on
            potential_misses;
            if (potential_mpki - actual_mpki < ε) then
                disable half of the active ways;
            else
                potential_misses -= lru_distance_counter[i];
                break;
            end
        end
    end
end
if (previous_mpki > actual_mpki) then
    previous_mpki = actual_mpki;
end
```

Figure 7: Pseudocode of Lite's decision algorithm.

4.3 *RMM$_{Lite}$* for Energy-Efficient TLBs

We also propose to add Lite and an L1-range TLB to Redundant Memory Mappings (RMM) [35] to further reduce the energy and performance overheads of L1 TLBs. As described in Section 2.2, RMM uses *range translations*, an efficient, alternative representation of arbitrarily large ranges of pages that are contiguously allocated in both virtual and physical address space. RMM targets reducing the number of page walks and employs an L2-range TLB that is accessed in parallel with the L2-page TLB.

RMM$_{Lite}$ augments the RMM with a small *L1-range TLB* and adds the Lite mechanism to the L1-page TLBs. The L1-range TLB is accessed on every memory operation in parallel with the L1-page TLBs. The L1-range TLB is fully associative and very small, e.g., 4 entries like the small L1-1GB TLB, so that it meets the tight timing requirements of L1 TLBs. The functionality and organization of L1-range TLB is the same as the originally proposed L2-range TLB, i.e., it caches a small number of range translations. Figure 8 shows the proposed TLB hierarchy for energy-efficient address translation. On an L1-range TLB hit, the address translation is obtained fast. On an L2-range TLB hit (after a miss in L1 TLBs), the hit range translation entry is copied to the L1-range TLB, in addition to copying the corresponding page table entry in the L1-page TLBs as in RMM. Note that the L1 TLBs for huge pages could either be simply disabled by the naive mechanism that was discussed in Section 3.1, or completely replaced by the (possibly larger) L1-range TLB.

The L1-range TLB itself increases the dynamic energy spent in L1 TLB accesses, because one more TLB is accessed on every memory operation. In addition, the L1-range TLB performs range checks instead of equality checks as in translation for fixed size memory regions. Thus, an L1-range TLB access costs more than an L1-page TLB access in terms of energy. However, each L1-range TLB entry maps an arbitrarily large range of contiguously allocated pages. The L1-range TLB can achieve higher hit ratio compared to page TLBs. This increased hit ratio in the L1-range TLB reduces further the utility of the L1-page TLBs. In response, Lite disables ways more aggressively in the L1-page TLBs compared to when only huge pages are supported, and reduces the total dynamic energy due to L1 TLB accesses.

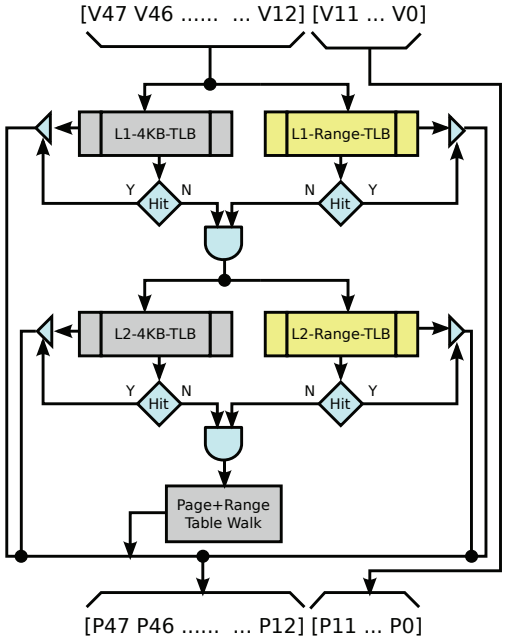


Figure 8: RMM$_{Lite}$ introduces an L1-range TLB and Lite (not shown) to the L1-page TLBs, in addition to the architectural support of RMM.

Thus, RMM$_{Lite}$ makes the case for energy-efficient address translation, reducing further the energy overheads at all levels while improving also performance.

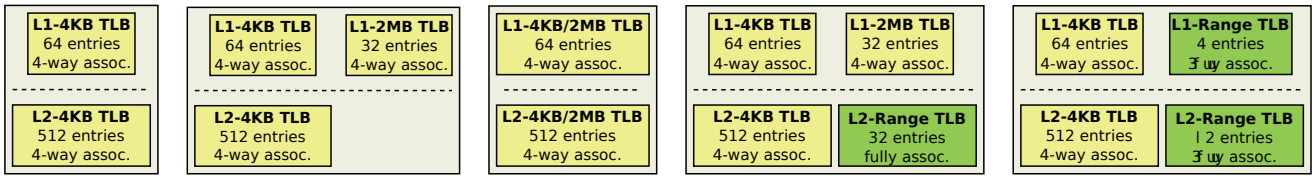
4.4 Discussion

Fully associative TLBs. We described Lite in the context of TLB organizations that support huge pages with separate set associative L1 TLBs [23], where each L1 TLB holds mappings of a single size. A different approach for huge page support is having a single fully associative L1 TLB that holds mappings of all page sizes [6, 48]. The same Lite mechanism for reducing dynamic energy applies to such TLB organizations. Although there is no notion of ways in a fully associative TLB, Lite clusters the distance of TLB hits from the LRU position as if there were ways, and reduces the TLB size in powers-of-two.

Lite's Cost. We did not analyze additional circuitry overheads for Lite, because the cost of computing the LRU distance and incrementing the corresponding counter (on a TLB hit) should be much lower than looking up the address (independently of a hit or miss) [20]. In addition, when the TLB operates with the minimum configuration (e.g., with only 1-way active), the additional circuits of Lite are not used and do not affect dynamic energy; this case is responsible for 63.7% of L1 TLB lookups with RMM$_{Lite}$ (Table 5).

5. METHODOLOGY

This section describes our simulation infrastructure and benchmarks.



(a) 4KB (b) THP & TLB_{Lite} (c) TLB_{PP} (d) RMM (e) RMM_{Lite}

Figure 9: TLB configurations.

Component	Size (entr.)	Assoc.	Read (pJ)	Write (pJ)	Leakage (mW)
L1-4KB TLB	64	4-way	5.865	6.858	0.3632
	32	2-way	1.881	2.377	0.1491
	16	1-way	0.697	0.945	0.0636
L1-2MB TLB	32	4-way	4.801	5.562	0.1715
	16	2-way	1.536	1.924	0.0703
	8	1-way	0.568	0.764	0.0295
L1-range TLB	4	fully	1.806	1.172	0.1395
L2-4KB TLB	512	4-way	8.078	12.379	1.6663
L2-range TLB	32	fully	3.306	1.568	0.2401
MMU-cache$_{PDE}$	32	2-way	1.824	2.281	0.1402
MMU-cache$_{PDPTE}$	4	fully	0.766	0.279	0.0500
MMU-cache$_{PML4}$	2	fully	0.473	0.158	0.0296
L1-Cache	32KB	8-way	174.171	186.723	13.3364

Table 2: Dynamic energy per read operation and write operation, and leakage power with 32 nm process technology for the memory structures that participate in address translation, based on Cacti [38].

Energy Model	
TLBs / MMUcache	$E_{TLB/MMUcache} = A * E_{read} + M * E_{write}$
Page walks	$E_{page\_walks} = Mem * Eread_{L1\_cache}$
Total energy	$E_{total} = \sum_{i=1}^{n} (E_{TLBi/MMUcachei}) + E_{page\_walks}$
Performance Model	
L1 TLB hits	$Cycles_{L1TLBmisses} = 0$ (all L1 TLBs are accessed in parallel with L1 dcache)
L1 TLB misses	$Cycles_{L2TLBmisses} = M_{L1TLBs} * 7$ (all L2 TLBs are accessed in parallel)
L2 TLB misses	$Cycles_{L2TLBmisses} = M_{L2TLBs} * 50$
Total cycles	$Cycles_{TLBmisses} = Cycles_{L1TLBmisses} + Cycles_{L1TLBmisses}$

A: Accesses M: Misses
Mem: Memory references to fetch PTEs (up to 4)

Table 3: Dynamic energy and performance models.

Simulation infrastructure. We developed a Memory Management Unit (MMU) simulator based on Pin [39], instrument all memory operations, and simulate various TLB configurations. Because TLB studies require longer instruction counts than other processor components for applications to realistically stress the TLBs, slow cycle-accurate simulators make for infeasibly long simulation times. Thus, we developed our own simulation infrastructure that focuses on the address translation path based on Pin, `pagemap`, and Cacti.

For a simulated L2 TLB miss, we access the real page table of the running process through `pagemap` [3] to determine whether it is a 4 KB page, a 2 MB page, or a range translation entry and its boundaries. To deduce the number of required memory references per page walk, we simulate a per-core MMU cache based on Intel's Paging Structure Caches [27]. The MMU cache consists of three individual structures, each of which holds different levels of the page table (PDE, PDPTE, and PML4 levels). These structures are all accessed in parallel after an L2 TLB miss. The configuration details of the MMU cache is based on [15] and summarized in Table 2.

We use Cacti [38] with 32 nm process technology to estimate the dynamic energy of the memory structures that participate in address translation. To estimate the dynamic energy of an N-entry range TLB, we use Cacti with the configuration of a regular fully associative page TLB, but with 2× more tag bits in order to account for the effect of the double comparison that takes place in the range TLB. To estimate the dynamic energy of a page TLB with some ways disabled (e.g., 64-entry 4-way, with 2 ways disabled), we use the dynamic energy results from Cacti for the resulting smaller structure (e.g., 32-entry 2-way TLB). Table 2 summarizes the results from Cacti for all simulated structures.

We couple the results from our MMU simulator with those from Cacti. This simulation infrastructure computes misses and hits per memory structure, and estimates the dynamic energy and the cycles spent in L1 and L2 TLB misses.

Configurations. We simulate the following configurations: (i) *4KB* supports only 4 KB pages. (ii) *THP* supports 4 KB and 2 MB pages through transparent huge pages [5], and is the state of the practice for reducing the performance overhead of L1 and L2 TLB misses. (iii) TLB_{Lite} includes the Lite mechanism on top of THP. (iv) *RMM* supports 4 KB, 2 MB pages, and an L2-range TLB. (v) TLB_{PP} is a perfect implementation of TLB_{Pred} [41]. TLB_{Pred} is a state of the art scheme that seeks to improve the energy efficiency of TLBs by supporting different page sizes in a single set associative TLB. TLB_{Pred} uses prediction to decide whether a reference goes to a huge page or not, in order to choose the appropriate TLB index bits and access the TLB. Our perfect implementation of TLB_{Pred}, named TLB_{PP}, assumes a perfect predictor with no energy overhead that always chooses the correct page size per lookup operation. In addition, TLB_{PP} mixes 4 KB and 2 MB pages in both L1 and L2 TLBs. (vi) RMM_{Lite} supports 4 KB pages and range translations in both L1 and L2 TLBs, and includes the Lite mechanism.

We set the threshold ε of Lite to 12.5% ($1/8_{th}$) relative increase in MPKI for TLB_{Lite} and as a 0.1 absolute increase for RMM_{Lite}, i.e., Lite disables ways if the predicted MPKI remains less than 12.5% or by 0.1 compared to the MPKI with fully enabled resources for TLB_{Lite} and RMM_{Lite}. Lite reduces the L1 TLBs down to 1-way active but never turns off

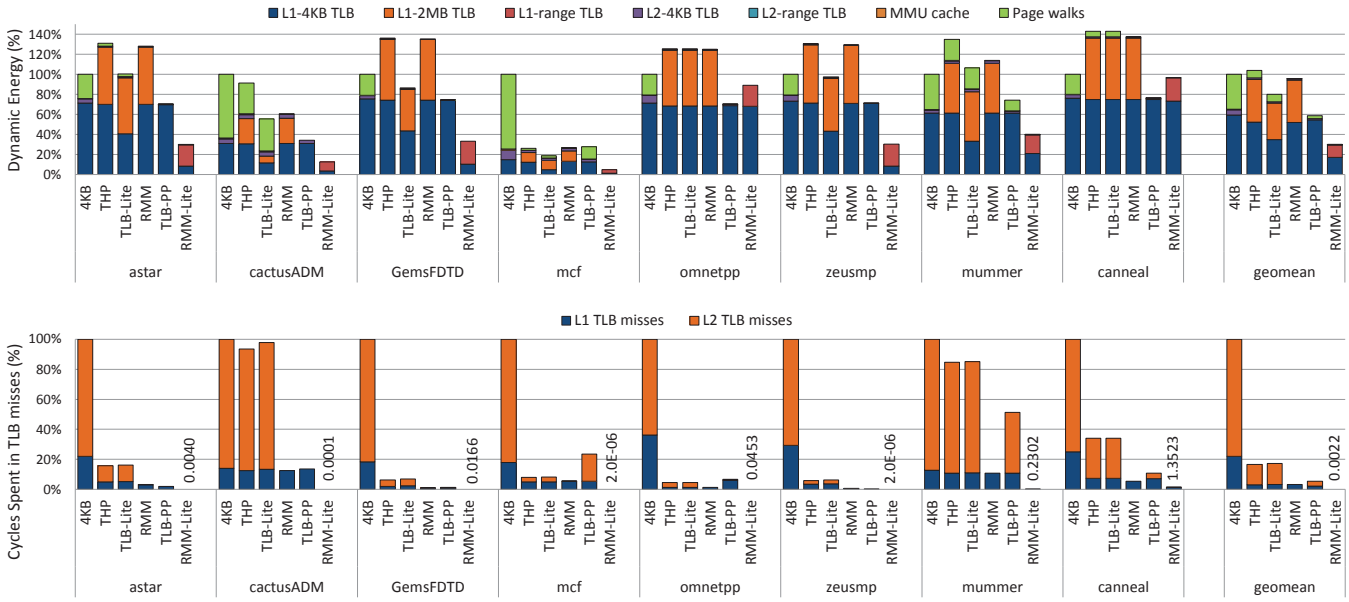


Figure 10: Dynamic energy spent in address translation and cycles spent in TLB misses for TLB intensive workloads.

Suite	Description	Application	Memory
SPEC 2006	compute & memory intensive single-threaded workloads	astar	350 MB
		cactusADM	690 MB
		GemsFDTD	860 MB
		mcf	1.7 GB
		omnetpp	165 MB
		zeusmp	530 MB
PARSEC	RMS multi-threaded workloads	canneal	780 MB
BioBench	Bioinformatics single-threaded workloads	mummer	470 MB

Table 4: Workload description and memory footprint.

completely an L1 TLB in our experiments. Finally, *RMM* and *RMM$_{Lite}$* use perfect eager paging, i.e., the operating system perfectly allocates all contiguous pages of virtual address space to contiguous physical pages. Figure 9 summarizes the simulated configurations and the corresponding parameters.

Dynamic energy. We report the amount of dynamic energy spent in the address translation path. Table 3 summarizes the equations of our energy model. The dynamic energy per translation structure is the sum of the dynamic energy spent due to lookup operations and the dynamic energy spent due to write operations after misses. Our model for the dynamic energy spent in page walks optimistically assumes that all page walk references always hit in the L1 cache of the memory hierarchy.

Performance. We report misses per thousand instructions in the L1 and L2 TLBs, and cycles spent in L1 and L2 TLB misses. Our estimations are based on the following assumptions: (i) L1 TLBs are accessed in parallel with the data cache, so L1 TLB hits add no cycles, (ii) L1 TLB misses trigger lookup accesses in L2 TLBs that take 7 cycles [28], and (iii) L2 TLB misses trigger page walks that take 50 cycles [36] for all applications. Thus, the cycles spent in TLB misses are the sum of the cycles spent in L1 TLB misses and in L2 TLB misses. Table 3 summarizes the equations of our performance model. Note that short L1 TLB misses, like those that hit in the L2 TLB, can be overlapped with execution in some cases, and may not decrease performance by that much. For RMM and RMM$_{Lite}$, the range table walks occur in the background and do not add to the execution time, but they incur dynamic energy overhead.

Benchmarks. We focus on various workloads that exhibit poor TLB performance from Spec2006 [26], BioBench [7], and Parsec [16], summarized in Table 4. We define as TLB intensive workloads those that experience more than 5 L1 TLB misses per thousand instructions with 4 KB pages. We also report results for all remaining Spec2006 and Parsec workloads in the sensitivity analysis subsection. We fast-forward the execution for 50 billion instructions, and then simulate for the next 50 billion instructions.

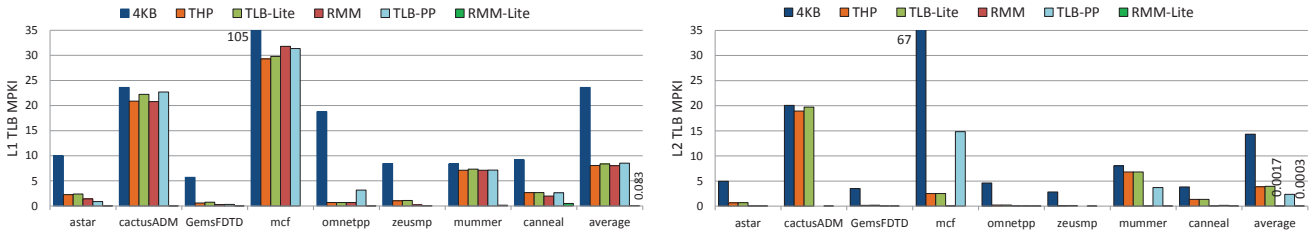
6. RESULTS

This section evaluates the two proposed TLB organizations: TLB$_{Lite}$ that adds the Lite mechanism on top of TLB support for huge pages, and RMM$_{Lite}$ that adds the Lite mechanism and the 4-entry L1-range TLB on top of RMM. We first evaluate how these TLB organizations reduce the dynamic energy in address translation and the cycles spent in L1 and L2 TLB misses for a set of TLB intensive workloads. Then we present results for more workloads, and finally we perform a sensitivity analysis based on the interval size and the random probability of Lite.

6.1 Dynamic Energy & Performance

Figure 10 shows the reduction of the dynamic energy in address translation and the cycles spent in L1 and L2 TLB misses for all the simulated configurations explained in Section 5. The results are normalized to the 4KB configuration.

Overview. The results show that (i) TLB$_{Lite}$ reduces the dynamic energy with respect to THP (Figure 10 top) without significantly affecting the performance (Figure 10 bottom), and (ii) RMM$_{Lite}$ further reduces the dynamic energy



(a) L1 TLB MPKI **(b) L2 TLB MPKI**

Figure 11: L1 and L2 TLB misses per thousand instructions.

	L1 TLB Lookups (%)									L1 TLB Hits (%)			
	TLB_{Lite}						RMM_{Lite}			TLB_{Lite}		RMM_{Lite}	
	4KB			2MB			4KB						
	4-ways	2-ways	1-way	4-ways	2-ways	1-way	4-ways	2-ways	1-way	4KB	2MB	4KB	Range
astar	39.6	57.2	3.2	96.7	3.3	0.0	0.0	0.1	99.9	75.7	24.3	32.4	67.6
cactusADM	22.8	24.0	53.2	14.6	11.9	73.5	0.1	0.1	99.9	90.8	9.2	0.0	100.0
GemsFDTD	42.9	44.9	12.2	54.4	41.7	4.0	2.3	0.4	97.4	30.1	69.9	0.1	99.9
mcf	25.8	26.7	47.5	97.8	1.7	0.5	0.0	0.0	100.0	38.9	61.1	12.0	88.0
omnetpp	100.0	0.0	0.0	100.0	0.0	0.0	99.3	0.7	0.0	55.2	44.8	51.0	49.0
zeusmp	45.5	43.5	11.1	86.6	13.3	0.1	0.0	0.0	100.0	37.6	62.4	0.0	100.0
mummer	32.8	67.2	0.0	98.4	0.5	1.0	7.8	79.4	12.9	95.7	4.3	5.8	94.2
canneal	100.0	0.0	0.0	100.0	0.0	0.0	97.5	2.5	0.0	91.0	9.0	25.9	74.1
average (%)	**51.2**	**32.9**	**15.9**	**81.1**	**9.0**	**9.9**	**25.9**	**10.4**	**63.7**	**64.4**	**35.6**	**15.9**	**84.1**

Table 5: (i) Percentage of lookups with 4, 2 and 1 active ways in the L1-page TLBs (left), and (ii) percentage of hits in the L1 TLBs (right), for TLB_{Lite} and RMM_{Lite}. RMM_{Lite} disables more ways than TLB_{Lite} thanks to the L1-range TLB.

of TLB lookups and eliminates almost completely the performance and the associated static energy overheads of L1 TLB misses.

4KB exhibits two sources of dynamic energy overhead: the L1 TLB lookups and the page walks. Depending on the workload's locality in the TLB hierarchy, one of the two sources dominates. Figure 11 shows the MPKI for the L1 and L2 TLBs. The L1 TLB lookups are responsible for the majority of overhead in these workloads, except for cactusADM and mcf that suffer more frequently from page walks. In addition, previous studies have shown that 4 KB pages lead to significant performance overhead [13, 15, 36] that increases in turn the total static energy.

THP reduces significantly the portion of dynamic energy and the performance overhead of page walks, due to fewer L1 and L2 TLB misses. However, THP increases the amount of dynamic energy spent in the L1 TLBs because the L1-2MB TLB is accessed on every memory operation, in addition to the L1-4KB TLB. These accesses increase the total dynamic energy consumption compared to 4KB for most workloads—up to 43% for canneal. On average, THP increases the dynamic energy by 4%, while reducing the cycles spent in TLB misses by 83%, compared to 4KB pages.

TLB_{Lite} reduces the dynamic energy by 23% on average and by 40% and 37% for cactusADM and GemsFDTD, compared to THP. TLB_{Lite} opportunistically reduces the dynamic energy spent in address translation when the utility of having all ways active becomes low. Table 5 shows the percentage of active ways during the execution time. On average, all 4-ways are active for 51% and 81% of the time in the L1-4KB TLB and L1-2MB TLB. In addition, TLB_{Lite} barely affects performance for most workloads except for canneal. Compared to the THP configuration, TLB_{Lite} increases the L1 and L2 TLB misses by 4% and 3% on average, and the cycles spent in TLB misses from 16.6% to 17.2%. Note that cycles spent in short TLB misses may be overlapped with execution; thus the impact on total execution time will be lower.

RMM eliminates the dynamic energy and performance overheads of page walks due to the L2-range TLB. However, the dynamic energy spent in the L1 TLBs remains high, similar to that with THP, because of accessing both L1 TLBs for 4KB and 2MB pages. On average, RMM reduces the dynamic energy by only 8% and the cycles spent in TLB misses by 80%, compared to THP.

TLB_{PP} is a perfect implementation of TLB_{Pred} [41], as explained in Section 5. We observe that TLB_{PP} reduces the dynamic energy and performance overheads of page walks because it enjoys larger reach compared to THP. In addition, the TLB_{PP} reduces the dynamic energy in L1 TLBs since only a single structure for both 4 KB and 2 MB pages is accessed on every memory operation, but these results under report its true costs. On average, TLB_{PP} would reduce the dynamic energy by 43% and the cycles spent in TLB misses by 67% compared to THP, but is unrealizable in practice.

RMM_{Lite} reduces the dynamic energy in address translation the most compared to the other approaches. RMM_{Lite} reduces dynamic energy by more than 80% for mcf and cactusADM, and by 71% on average while eliminating more than 99% of cycles spent in TLB misses compared to THP. This occurs because the high hit ratio of the L1-range TLB allows Lite to disable ways more aggressively in the L1-4KB TLB. Table 5 also shows the percentage of L1 TLB hits that come from the L1-4KB TLB and the L1-range TLB. The L1-range TLB contributes by 84.1% to the L1 TLB hits, and thus, RMM_{Lite} depends less on the performance of the L1-4KB TLB and runs 63.7% of time with only 1-way active in the L1-4KB TLB (Table 5).

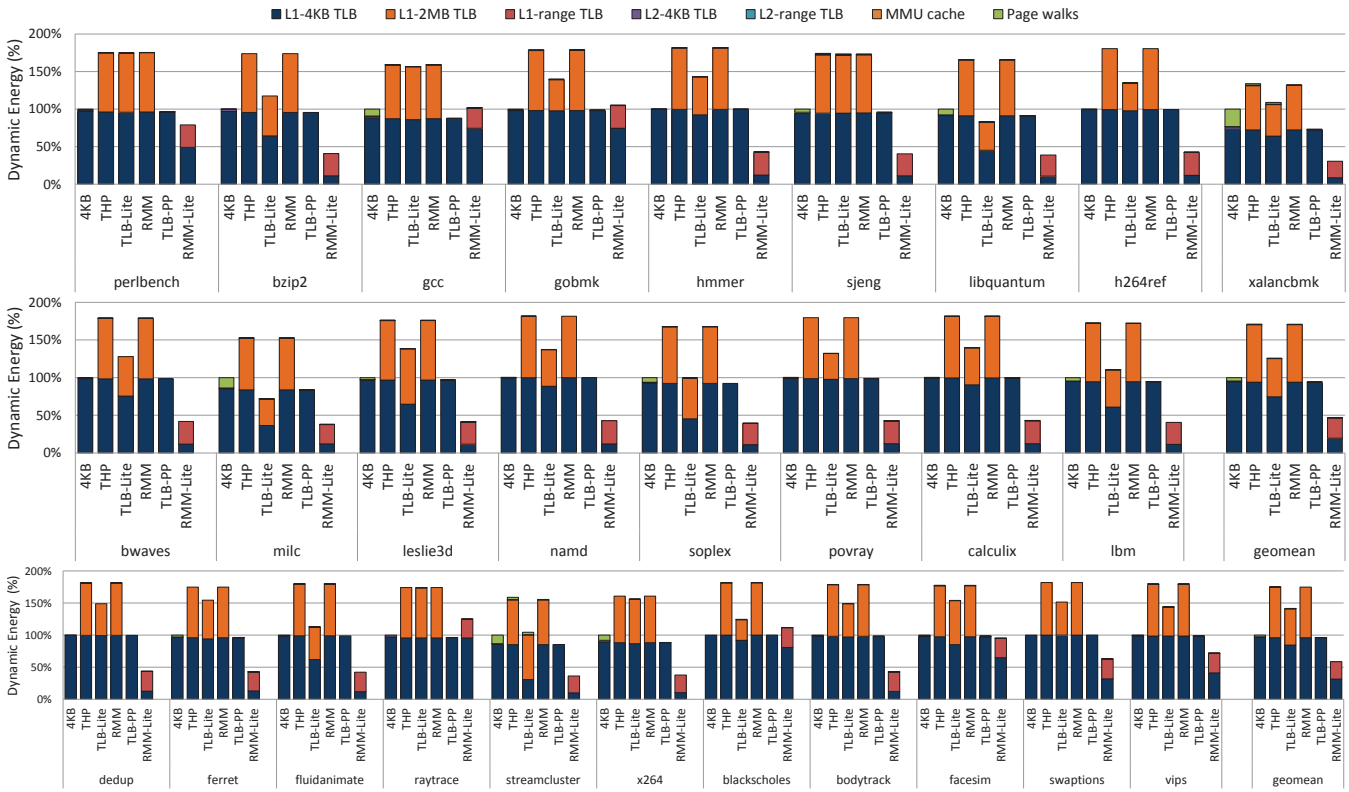


Figure 12: Dynamic energy reduction for the rest of Spec2006 (top and middle) and Parsec (bottom) workloads.

Compared to TLB_{PP}, RMM_{Lite} brings less dynamic energy improvements only for omnetpp and canneal because the L1-4KB TLB has high utilization for those workloads (Table 5). Still, RMM_{Lite} reduces the dynamic energy overhead by 49% on average, compared to TLB_{PP}. Note that RMM_{Lite} and TLB_{PP} are orthogonal; a combined approach could use the L1-range TLB for range translations, the TLB_{PP} for pages, and the Lite mechanism to disable ways opportunistically, as with regular page TLBs.

In addition to the dynamic energy savings, RMM_{Lite} significantly reduces L1 and L2 TLB misses, further improving the performance and reducing static energy overheads. Compared to RMM, RMM_{Lite} improves performance more because it eliminates most L1 TLB misses, in addition to eliminating most L2 TLB misses as RMM does. Overall RMM_{Lite} makes a good case for energy-efficient address translation.

6.2 Sensitivity Analysis

Other workloads. Our evaluation in the previous section focused on a set of TLB intensive workloads. For completeness, we ran experiments with other workloads that stress the TLB hierarchy less and observed similar results. Figure 12 shows the reduction in dynamic energy for the rest of Spec2006 (top and middle) and Parsec (bottom) workloads. On average, TLB_{Lite} reduces the dynamic energy spent in address translation by 26% and 20% for those Spec2006 and Parsec workloads, while RMM_{Lite} reduces the dynamic energy by 72% and 66%. Regarding performance, the results are similar to those for the TLB intensive workloads.

Interval size and random probability. Lite depends on the size of the interval and the random probability for activating all ways in the L1 TLBs. To quantify the impact of these parameters, we performed a sensitivity analysis varying the interval size from 1 million to 10 million instructions and the random probability from 1/8 to 1/128. We find that Lite performs slightly better in terms of both performance and dynamic energy, with shorter interval and with lower probability. The short interval allows Lite to respond faster to performance changes, while the low probability avoids frequently enabling all ways to check the potential for performance improvement.

Reducing static energy. Although we focused on reducing the dynamic energy of address translation, the proposed techniques can also reduce the static (leakage) energy of TLBs when combined with schemes that power-gate the disabled ways [24, 44].

Threshold. The benefits of Lite depend also on the threshold ε for increased MPKI due to way-disabling. The threshold choice introduces a trade-off between dynamic and static energy. Studying the impact of different thresholds on total energy and performance could be the subject of future work.

7. RELATED WORK

This section reviews the related work (except for TLB_{Pred} [41] discussed in Section 6), categorized into techniques that optimize TLBs for energy efficiency, dynamically resizing TLBs, selective TLB lookups, and virtual caches.

Optimizing TLBs for energy efficiency. Several techniques have been proposed to improve the energy efficiency of TLBs. Juan et al. [31] proposed circuit optimizations that reduce the lookup energy in TLBs. Banked TLBs [17, 18,

37] and TLB filtering [11, 17, 21] can also help in reducing dynamic energy by accessing only one bank or just a filter on each memory operation. Similarly, Lee et al. [37] proposed a partitioned L1 TLB, with each part serving translations for a semantic region (stack, heap, global data). That TLB organization was further improved leveraging the low entropy of information in the stack and global data memory addresses [10]. To reduce the TLB energy for multi-issue superscalar processors, Ballapuram et al. [11] proposed a compaction mechanism for issuing only a single TLB lookup, when multiple memory references access the same page at the same cycle. Xue et al. [53] proposed to speculatively perform address translation, based on the base-displacement address, by accessing a small L0 TLB early in the pipeline, so that the translation latency is not increased. Finally, Seyedi et al. [47] proposed combining nano electro mechanical switches with CMOS technology for fully associative L1 TLBs.

While these techniques reduce the dynamic energy spent in TLBs, they do not consider mechanisms that increase TLB reach [13, 22, 35, 42, 43, 50] to improve energy efficiency. Our proposed designs leverage the benefits of such mechanisms to reduce the total energy spent in address translations. Thus, TLB_{Lite} and RMM_{Lite} are orthogonal to those approaches, and could further improve their benefits.

Dynamically resizing TLBs. Balasubramonian et al. [9] proposed an interval-based scheme to dynamically resize the TLB, trading off dynamic energy for performance. The objective of that approach is similar to Lite. However, their design and algorithm targets a monolithic, fully associative TLB and tracks only whether a TLB entry was referenced or not to decide for resizing. Thus, the energy savings opportunity becomes lower in case that the TLB entries are referenced only few times but not heavily utilized. In contrast, Lite tracks the utility of TLB entries in the miss ratio, considers the presence of separate L1 TLBs, and provides better opportunity for resizing TLB resources.

Selective lookups in TLBs. Kadayif et al. [32] combined hardware and compiler techniques to avoid lookups in instruction TLBs. A register holds the most recently used iTLB entry, and the compiler generates instructions that access only a register instead of the iTLB. That approach was extended later for the data TLB [33, 34]. However, the TLBs are still used in such system. Thus, TLB_{Lite} and RMM_{Lite} are again orthogonal and can further reduce the total energy cost of address translation in these systems.

Virtual caches. Prior work proposed virtual caches [14, 29, 52] to reduce the energy and performance overheads of address translation. With virtual caches, the cache hierarchy is accessed without TLB lookups, unless a cache miss occurs. While saving almost all TLB energy, they introduce many more changes to the architecture and require additional support to handle synonyms and enforce protection.

8. SUMMARY

The goal of this paper is to improve the energy efficiency in address translation. We proposed *Lite*, a mechanism that monitors the performance and utility of L1 TLBs and adaptively changes their sizes with way-disabling, and applied Lite to a standard TLB hierarchy with support for huge pages, named TLB_{Lite}. In addition, we proposed RMM_{Lite}, based on Redundant Memory Mappings (RMM). RMM_{Lite} augments RMM with an L1-range TLB and the Lite mechanism. The high hit ratio of the L1-range TLB allows Lite to disable ways in L1-page TLBs more aggressively. Our results show that TLB_{Lite} reduces the dynamic energy spent in address translation by 23% with minimal impact on TLB miss cycles. RMM_{Lite} further reduces the energy spent in address translation by 71% and the overhead from L1 TLB misses by 99%, on top of the near-zero L2 TLB misses of RMM.

ACKNOWLEDGEMENTS

We thank our anonymous reviewers, Oscar Palomar and Adria Armejach for their insightful comments and feedback on the paper. This work is supported in part by the European Union (FEDER funds) under contract TIN2012-34557, the European Union's Seventh Framework Programme (FP7/2007-2013) under the ParaDIME project (GA no.318693), the National Science Foundation (CCF-1218323, CNS-1302260, CCF-1438992, and CCF-1533885), Google, and the University of Wisconsin (Kellett award and Named professorship to Hill). Karakostas is also supported by an FPU research grant from the Spanish MEC. Hill has a significant financial interest in AMD.

9. REFERENCES

[1] "Huge Pages Part 1 (Introduction)," http://lwn.net/Articles/374424/.

[2] "Intel Strongarm Processor," www.intel.com/design/pca/ applicationsprocessors/1110_brf.htm.

[3] "Pagemap, from the userspace perspective," https://www.kernel.org/doc/Documentation/vm/pagemap.txt.

[4] "Sh-3 RISC Processor family," http://www.hitachi-eu.com/hel/ecg/products/micro/32bit/sh_3.html.

[5] "Transparent Huge Pages in 2.6.38," http://lwn.net/Articles/423584/.

[6] Advance Micro Devices, *Software Optimization Guide for AMD Family 15h Processors*, 2014, no. 47414.

[7] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung, "BioBench: A Benchmark Suite of Bioinformatics Applications," in *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 2–9, 2005.

[8] D. H. Albonesi, "Selective Cache Ways: On-demand Cache Resource Allocation," in *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 248–259, 1999.

[9] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-purpose Processor Architectures," in *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 245–257, 2000.

[10] C. Ballapuram, K. Puttaswamy, G. H. Loh, and H.-H. S. Lee, "Entropy-based Low Power Data TLB Design," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 304–311, 2006.

[11] C. S. Ballapuram, H.-H. S. Lee, and M. Prvulovic, "Synonymous Address Compaction for Energy Reduction in Data TLB," in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pp. 357–362, 2005.

[12] T. W. Barr, A. L. Cox, and S. Rixner, "Translation Caching: Skip, Don'T Walk (the Page Table)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pp. 48–59, 2010.

[13] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient Virtual Memory for Big Memory Servers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pp. 237–248, 2013.

[14] A. Basu, M. D. Hill, and M. M. Swift, "Reducing Memory Reference Energy with Opportunistic Virtual Caching," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, pp. 297–308, 2012.

[15] A. Bhattacharjee, "Large-reach Memory Management Unit Caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 383–394, 2013.

[16] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 72–81, 2008.

[17] Y. Chang and M. Lan, "Two New Techniques Integrated for Energy-Efficient TLB Design," *IEEE Trans. VLSI Syst.*, vol. 15, no. 1, pp. 13–23, 2007.

[18] J.-H. Choi, J.-H. Lee, S.-W. Jeong, S.-D. Kim, and C. C. Weems, "A Low Power TLB Structure for Embedded Systems," *Computer Architecture Letters*, vol. 1, 2002.

[19] J. F. Couleur and E. L. Glaser, "Shared-access data processing system," Nov. 19 1968, US Patent 3,412,382.

[20] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott, "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power," in *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pp. 141–152, 2002.

[21] D. Fan, Z. Tang, H. Huang, and G. R. Gao, "An Energy Efficient TLB Design Methodology," in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pp. 351–356, 2005.

[22] J. Gandhi, A. Basu, M. D. Hill, and M. M. Swift, "Efficient Memory Virtualization: Reducing Dimensionality of Nested Page Walks," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 178–189, 2014.

[23] P. Hammarlund, "4th Generation Intel Core processor, codenamed Haswell," in *Proceedings of Hot Chips Symposium*, 2013.

[24] H. Hanson, M. S. Hrishikesh, V. Agarwal, S. W. Keckler, and D. Burger, "Static energy reduction techniques for microprocessor caches." *IEEE Trans. VLSI Syst.*, vol. 11, no. 3, pp. 303–313, 2003.

[25] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2003.

[26] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[27] Intel Corporation, "TLBs, Paging-Structure Caches and their Invalidation," 2008, no. 317080-003.

[28] Intel Corporation, "Intel® 64 and IA-32 Architectures Optimization Reference Manual," April 2012, no. 248966-026.

[29] B. Jacob and T. Mudge, "Virtual Memory in Contemporary Microprocessors," *IEEE Micro*, vol. 18, no. 4, pp. 60–75, Jul. 1998.

[30] B. L. Jacob and T. N. Mudge, "A Look at Several Memory Management Units, TLB-refill Mechanisms, and Page Table Organizations," in *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 295–306, 1998.

[31] T. Juan, T. Lang, and J. J. Navarro, "Reducing TLB Power Requirements," in *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, pp. 196–201, 1997.

[32] I. Kadayif, A. Sivasubramaniam, M. Kandemir, G. Kandiraju, and G. Chen, "Generating Physical Addresses Directly for Saving Instruction TLB Energy," in *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 185–196, 2002.

[33] I. Kadayif, P. Nath, M. T. Kandemir, and A. Sivasubramaniam, "Compiler-directed physical address generation for reducing dTLB power," in *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 161–168, 2004.

[34] M. Kandemir, I. Kadayif, and G. Chen, "Compiler-directed Code Restructuring for Reducing Data TLB Energy," in *Proceedings of the 2Nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 98–103, 2004.

[35] V. Karakostas, J. Gandhi, F. Ayar, A. Cristal, M. D. Hill, K. S. McKinley, M. Nemirovsky, M. M. Swift, and O. Unsal, "Redundant Memory Mappings for Fast Access to Large Memories," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 66–78, 2015.

[36] V. Karakostas, O. S. Unsal, M. Nemirovsky, A. Cristal, and M. Swift, "Performance Analysis of the Memory Management Unit under Scale-out Workloads," in *Proceedings of the 2014 IEEE International Symposium on Workload Characterization*, pp. 1–12, 2014.

[37] H.-H. S. Lee and C. S. Ballapuram, "Energy Efficient D-TLB and Data Cache Using Semantic-aware Multilateral Partitioning," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pp. 306–311, 2003.

[38] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 694–701, 2011.

[39] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 190–200, 2005.

[40] MIPS Technologies, Incorporated, "MIPS32 Architecture for Programmers Volume iii: The MIPS Privileged Resource Architecture," 2001, no. MD00090, Revision 0.95.

[41] M.-M. Papadopoulou, X. Tong, A. Seznec, and A. Moshovos, "Prediction-based superpage-friendly TLB designs," in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture*, pp. 210–222, 2015.

[42] B. Pham, A. Bhattacharjee, Y. Eckert, and G. H. Loh, "Increasing TLB reach by exploiting clustering in page translations," in *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture*, pp. 558–567, 2014.

[43] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced Large-Reach TLBs," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 258–269, 2012.

[44] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 90–95, 2000.

[45] D. Quintero, S. Chabrolles, C. H. Chen, M. Dhandapani, T. Holloway, C. Jadhav, S. K. Kim, S. Kurian, B. Raj, R. Resende, B. Roden, N. Srinivasan, R. Wale, W. Zanatta, and Z. Zhang, "IBM Power Systems Performance Guide Implementing and Optimizing," 2013.

[46] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 423–432, 2006.

[47] A. Seyedi, V. Karakostas, S. Cosemans, A. Cristal, M. Nemirovsky, and O. S. Unsal, "NEMsCAM: A novel CAM cell based on nano-electro-mechanical switch and CMOS for energy efficient TLBs," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 51–56, 2015.

[48] M. Shah, R. Golla, G. Grohoski, P. Jordan, J. Barreh, J. Brooks, M. Greenberg, G. Levinsky, M. Luttrell, C. Olson, Z. Samoail, M. Smittle, and T. Ziaja, "Sparc T4: A Dynamically Threaded Server-on-a-Chip," *IEEE Micro*, vol. 32, no. 2, pp. 8–19, Mar. 2012.

[49] A. Sodani, "Race to Exascale: Opportunities and Challenges," in *MICRO Keynote*, 2011.

[50] M. Talluri and M. D. Hill, "Surpassing the TLB Performance of Superpages with Less Operating System Support," in *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 171–182, 1994.

[51] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson, "Tradeoffs in Supporting Two Page Sizes," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 415–424, 1992.

[52] D. A. Wood, S. J. Eggers, G. Gibson, M. D. Hill, and J. M. Pendleton, "An In-cache Address Translation Mechanism," in *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pp. 358–365, 1986.

[53] J. Xue and M. Thottethodi, "PreTrans: Reducing TLB CAM-search via Page Number Prediction and Speculative Pre-translation," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, pp. 341–346, 2013.