TLC: A Tag-Less Cache for Reducing Dynamic First Level Cache Energy

Andreas Sembrant, Erik Hagersten, and David Black-Shaffer
Uppsala University, Department of Information Technology
P.O. Box 337, SE-751 05 Uppsala, Sweden
{andreas.sembrant, erik.hagersten, david.black-schaffer}@it.uu.se

ABSTRACT

First level caches are performance-critical and are therefore optimized for speed. To do so, modern processors reduce the miss ratio by using set-associative caches and optimize latency by reading all ways in parallel with the TLB and tag lookup. However, this wastes energy since only data from one way is actually used.

To reduce energy, phased-caches and way-prediction techniques have been proposed wherein only data of the matching/predicted way is read. These optimizations increase latency and complexity, making them less attractive for first level caches.

Instead of adding new functionality on top of a traditional cache, we propose a new cache design that adds way index information to the TLB. This allow us to: 1) eliminate extra data array reads (by reading the right way directly), 2) avoid tag comparisons (by eliminating the tag array), 3) filter out misses (by checking the TLB), and 4) amortize the TLB lookup energy (by integrating it with the way information). In addition, the new cache can directly replace existing caches without any modification to the processor core or software.

This new Tag-Less Cache (TLC) reduces the dynamic energy for a 32 kB, 8-way cache by 78 % compared to a VIPT cache without affecting performance.

Categories and Subject Descriptors

B.3.2 [**MEMORY STRUCTURES**]: Cache Memories

1. INTRODUCTION

Modern processors optimize L1 caches by trading energy for performance. As a result, a significant part of the processor's core power is spent in the local cache hierarchy. Intel reports that 3% - 13% and 12% - 45% of the core power comes from TLBs and caches, respectively [20] (Figure 1a). Of this, 80% of the TLB and cache energy is spent in the data-array (Figure 1b).

MICRO-46 December 07 - 11, 2013, Davis, CA, USA

ACM 978-1-4503-2638-4/13/12 ...$15.00.

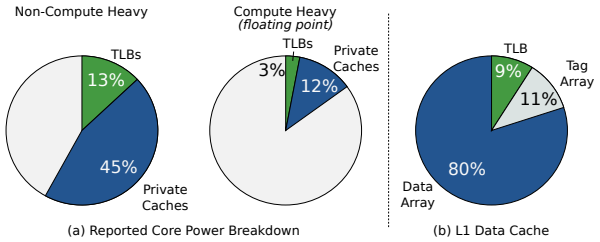


Figure 1: Reported Core Power Breakdown (a) for a non-compute and a compute heavy application [20], and (b) modeled L1 data cache power breakdown.

Most L1 access energy is wasted reading meta-data (e.g., tags) or data that will not be used. To optimize for latency, the TLB, the tag-array and the data-array are first read in parallel, and then afterwards the correct way is selected by comparing the tags. In an 8-way set-associative cache, only one of the eight ways is actually used in the end, meaning that up to 87.5% of the energy is wasted.

However, the L1 cache is performance-critical and this energy does improve latency. In this paper, we propose a new Tag-less Cache (TLC) design that removes most of the wasted lookup energy, without sacrificing performance. More specifically, the new cache design is: 1) fast (supports high clock frequencies, and low latencies) and 2) effective (has low miss ratio). To save energy, it also: 3) minimizes the number of bits that are read and 4) allows engineers to choose a good SRAM aspect ratio (i.e., power efficient shape).

Several techniques have been proposed to reduce data-array lookup energy. However, they come at a cost in increased latency and complexity. For example, *phased caches* perform the lookup in two phases. The tags are first read and compared, then only the matching way is read. This saves most of the data-array lookup energy in associative caches, but increases latency. To both save energy and minimize the effect on latency, *way-prediction* has been proposed [5, 22, 13, 9]. A small prediction table is typically placed in front of the cache and is used to predict the way. On a correct prediction, energy is saved and the latency remains the same, but a miss incurs extra costs in both energy and latency.

These techniques generally add extra hardware or functionality on top of an traditional cache. Instead, we propose a new cache design. Our design is based on the observation that the TLB is already consulted on every memory access to translate virtual addresses into physical addresses and to

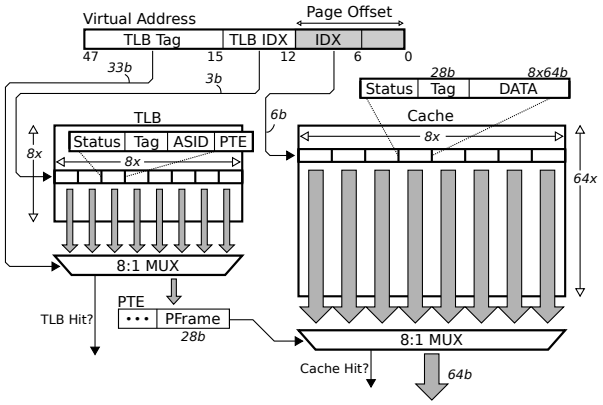


Figure 2: Schematic overview of a 32 kB, 8-way Virtually Indexed Physically Tagged (VIPT) cache, with a 64-entry, 8-way TLB.

provide page protection status, etc. We therefore asked the question: Can we augment the TLB in such a way that we can remove the power inefficiency of a traditional cache?

Yes, we can save most of the energy by creating an extended TLB (eTLB) that stores extra way index information. This information consists of the way location and presence for each of the page's cache lines. A lookup in the eTLB will therefore provide information about where the cache line is located in the data-array. This enable us to: 1) eliminate extra data array reads (by determining the correct way to read from the TLB), 2) avoid tag comparisons (by eliminating the tag array), 3) filter out cache misses (by checking a presence bit in the eTLB), and 4) amortize the TLB lookup energy (by integrating it with the way information). This results in fewer reads and better SRAM aspect ratios. Moreover, the final design is fast enough to support contemporary clock frequencies and it has the same performance (miss ratio and CPI) as a standard VIPT cache.

Our TLC design reduces dynamic first level cache energy by 78% compared to a standard VIPT cache, while delivering similar performance, and handling synonyms and coherency. In this work, we focus on the L1 data cache, but the TLC design should be equally applicable to instruction caches as well.

2. BACKGROUND

To put our Tag-Less Cache (TLC) design in context, we compare it with standard cache organizations[1] in terms of energy and performance. We use a 32kB VIPT L1 cache (*see Figure 2*) as our baseline, and we evaluate the relative energy, performance and SRAM shape. Table 1 highlights the differences between the cache organizations (PIPT, VIVT, VIPT and TLC), where VIPT has been divided into standard parallel lookup (STD), two-phase lookup (2P), and way-prediction (WP).

Energy Efficiency. A cache's energy efficiency typically depends on how many *bits are read from the data-array* and how much *meta-data is read from the TLB and the tag-array.* Phased, way-prediction and TLC reduce the dynamic read energy by only accessing one way from the data-array. In addition, TLC also reduces the amount of meta-data reads by eliminating the tag-array. VIVT on the other hand reduces TLB reads by only performing the address translation on an L1 cache miss.

[1]PIPT (physically indexed/physically tagged), VIVT (virtually indexed/virtually tagged), VIPT (virtually indexed/physically tagged), and optimizations including parallel lookup and phased lookup.

		PIPT	VIVT	VIPT STD	VIPT 2P	VIPT WP	TLC
Perf.	Latency	−−		1	−−	−	−
	Frequency			1	+	+	+
SRAM Shape				1	+	+	+
Energy Efficiency	Data			1	+	+	+
	Meta		+	1			+

Table 1: Performance and energy-efficiency trade-offs. VIPT is divided into the standard parallel lookup (STD), 2-phase lookup (2P), and way-prediction (WP).

Performance. There are two main components to cache performance: latency (i.e., number of cycles), and clock frequency (i.e., cycle time). The main component to the cycle time is typically the data-array, since it is much larger than the TLB and the tag-array. Phased, way-prediction, and TLC, read the right way directly (i.e., faster cycle time [5]) at a cost in longer latency. Phased caches serialize tag and data lookups, and way-predicting caches incur extra latency at mispredictions. The TLC first reads the way information from the eTLB, but, this latency can be reduced by integrating the eTLB more closely with the data-array (Section 4.2). PIPT, on the other hand, always pays extra latency costs for serializing TLB and L1 lookups, without receiving any benefit.

SRAM Shape. The SRAM's aspect ratio (number of wordlines to bitlines) affects both performance and energy. For example, it takes 17% more energy and 26% longer to read a word from an 4:1 (wide) ratio SRAM than a 1:1 (square) SRAM. Standard set-associative caches with parallel lookups can put severe constraints on how the SRAM is shaped since all the ways are read in parallel (i.e., making it wider). For example, to support 256-bits AVX [8] aligned reads in a single access, 2048 bits must be read in an 8-way cache (i.e., 16:1 minimum aspect ratio for a 32kB cache). Phased, way-prediction, and TLC, only have to read one way from the data-array (256 bits) and can therefore support a more efficient 1:1 aspect ratio. Furthermore, caches are typically also sub-banked for more parallelism, which exacerbates this problem further by enforcing wider aspect ratios.

In short, the new TLC design pays a small penalty in latency by requiring way information from the eTLB in exchange for much better energy efficiency and SRAM flexibility.

3. SAVING ENERGY WITH TLC

The goal of this work is to reduce dynamic L1 cache energy. To do so, we minimize meta-data reads by removing tags and we save data-array energy by reading the correct way directly. A traditional VIPT cache does two tag

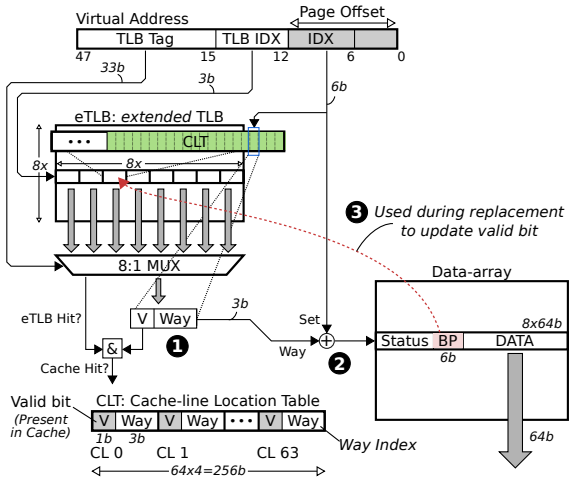


Figure 3: Tag-Less Cache (TLC) design overview for a 32 kB, 8-way cache, with a 64-entry, 8-way eTLB.

lookups, one in the TLB and one in the cache. However, the second lookup is redundant. By augmenting the TLB with extra cache line location information (see Figure 3), we can remove the cache tag lookup, and we can directly index into the data-array with the help of the extra location information from the extended TLB. In this section, we describe how the TLC works with the help of two examples, a cache hit and a cache miss.

A cache tag typically answers two questions: Is my data in the cache, and, if so, where is my data located? To avoid a cache tag lookup, the extended TLB (eTLB) must be able to answer these two questions on its own. In addition, the TLB generally works at page granularity (4 kB), but, the eTLB must be able to answer where cache lines (64 B) are located.

To do so, each eTLB entry has a field containing cache line location information for all cache lines belonging to that page, called a Cache-line Location Table (CLT). Each entry in the CLT (i.e., cache line) contains a valid bit for whether the cache line is present in the cache and way index information indicating in which way the cache line is located in (1 valid and 3 way-index bits for an 8-way cache).

The TLC design requires that all cache lines must always have an valid entry in the eTLB to ensure that they can be found by the eTLB.

3.1 Example 1: Cache Hit

A cache hit requires two steps: 1) lookup the cache line valid bit and the *way index* in the eTLB, and 2) read the data from the data-array.

Step 1. The eTLB is indexed and accessed like a regular TLB, with the exception that it also provides the cache line location information from the CLT. The CLT contains information for all the cache lines within a page. However, we only need the addressed cache line's way index. To access only that part of the CLT, we use the cache index bits (IDX) from the virtual address to select the correct cache line information from the CLT. This is possible since a page's cache lines are linearly mapped to different sets.

Step 2. The second step is to generate the address in the data array by combining the way index from the CLT with the set index (IDX) from the virtual address to directly index into the data-array (see point 1 and 2 in Figure 3).

The second step does not depend on the TLB to output the page frame number since there are no tag comparisons. We only need the page frame when we write back data to next level caches. This means that we can reduce the number bits read from the TLB on cache hits by skipping the page frame (28 bits). As a result we end up reading 24 fewer bits from the eTLB compared to a standard VIPT TLB.

		VIPT		TLC	
		Storage	Read	Storage	Read
TLB	Tag	0.33 kB	42 B	0.33 kB	42 B
	Data	0.28 kB	36 B	2.33 kB	12 B
Cache	Tag	1.9 kB	29 b	–	–
	Data	32 kB	64 B	32.5 kB	8 B
	Total	34.5 kB	162 B	35.2 kB	62 B
				+2 %	-62 %

Table 2: Storage requirements and number of bytes read from SRAM cells during a cache read hit for a 32 kB (8-way, 64 B blocks, 64-bit words) data cache with a 64-entry (8-way, 4 kB page) TLB.

Table 2 shows a bit-count analysis comparing a standard VIPT cache and our TLC design. By extending the TLB to an eTLB, the TLC reads fewer bits from the eTLB, removes the tag reads, and only reads one way from the data-array. This results in a 62% decrease in number of bits read[2].

Note that we do add more data to the eTLB, so the total cache size increases. However, since the new eTLB data is approximately the same size as the tag-array (which we removed), the total storage requirements increases only by 2%.

3.2 Example 2: Cache Miss

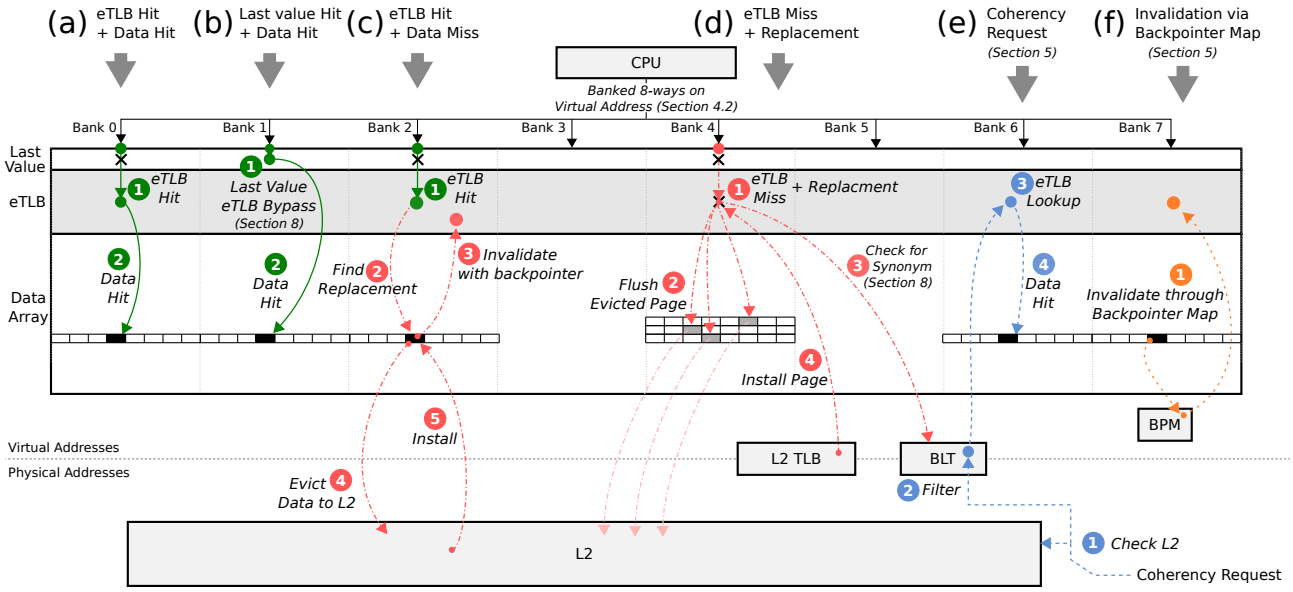
Since we have no tags, there are two ways in which we can determine that the data is not in the cache: 1) the valid bit for the cache line in the eTLB is not set, or 2) the page is not in the eTLB (i.e., TLB miss). A side effect of this is that the TLC cache implements a miss-filter for free. We can therefore conserve energy on cache misses by not reading the data-array at all. A VIPT cache will read all ways regardless whether there is a miss or not.

A cache miss is typically handled in two steps: First, we evict conflicting cache lines, and then we insert the new cache line. Both of these steps need to update the eTLB since it keeps track of what data is in the cache and where it is located. A cache miss is handled differently depending on whether: 1) the cache line is missing, but the page exist in the eTLB, or 2) both the cache line and the page are missing.

eTLB Hit. If there is a eTLB hit, but the cache line is not in the CLT, then we have a cache miss and the new line must be inserted. Before that, the cache line selected for replacement[3] must be evicted by invalidating its CLT entry in the eTLB and by writing back any dirty data to the next

[2]Note that this is a conservative analysis due to a very narrow cache bus width (64-bits). The difference would be far greater with a wider bus width.

[3]The design is not limited to any particular cache replacement policy. For our experiments, we use standard LRU cache replacement.



(a) eTLB hit and Data hit: The CPU sends a request to the cache. The request hits in the eTLB (1) and the CLT tells us that the data is in the cache and the specific way. The data is then read and sent to the CPU from the cache by combining the set index from the virtual address with the way information from the eTLB (2).

(b) eTLB Optimization using last value prediction: The request matches the last-used eTLB entry and therefore hits in the last-value eTLB register (1). This allows us to bypass the eTLB to save energy. The data is then read and sent to the CPU from the cache by combining the set index from the virtual address with the way information from the eTLB (2).

(c) eTLB hit but Data miss: The CPU request hits in the eTLB (1), but the CLT shows that the data is not in the cache. A victim cache line is then found using LRU replacement (2). The victim cache line's eTLB entry is invalidated by following the backpointer (3). The victim cache line is then evicted to L2 (4). The missing cache line is then fetched from L2 and installed into L1 (5). Finally, the eTLB is updated to point to the inserted cache line's way.

(d) eTLB miss and replacement: The CPU request misses in the eTLB (1). The replacement eTLB page is evicted by flushing its cache lines to L2 (2). The reverse TLB (BLT) is consulted to detect synonyms (3). The missing page is then installed by reading the address translation from the L2 TLB.

(e) External coherence request: The L2 cache is checked as normal (1). The external request is a physical address, but the TLC cache is virtually indexed. A physical to virtual translation is done in the BLT, which also filters the accesses. (If the coherency request misses in the BLT then we do not need to check the TLC since every eTLB entry must also have an BLT entry.) The virtual address is then used to read the way information in the eTLB (3), and the cache line data is accessed (4).

(f) Synonym optimization: An indirection is used to optimize synonym handling by having a backpointer to the BPM, which in turn points to the eTLB. This added level of indirection means that we only need to update the BPM when a synonym moves in the eTLB.

Figure 4: Tag-Less Cache (TLC) design overview.

level cache. To invalidate the cache line in the eTLB, its valid bit in its page's CLT is set to false.

However, since we have no tags in the cache, we can no longer identify which page (eTLB entry) a cache line belongs to by just reading the data-array. Instead, we need to find the eTLB entry to invalidate the evicted cache line. To do so, we add a backpointer (BP) to each cache line that points to the cache line's eTLB entry. The cache line is then invalidated by following the backpointer to the right eTLB entry (see point 3 in Figure 3). To invalidate the cache line in the CLT, we use the set index from the virtual address to index into the CLT as we did when reading (Note that the bit calculation in Table 2 include the BP field).

Once we have invalidated the replaced cache line's entry in its page's CLT, the new cache line is inserted, and the new cache line's CLT entry is updated with the way index it was inserted into. Finally, the backpointer is updated to point to the new cache line's eTLB entry.

The backpointer does add some extra space. However, it is much smaller than a tag (6 vs. 28 bits). Furthermore, the backpointer is only used during replacement, while tags are read on every cache access.

eTLB Miss. If there is a eTLB miss (hence also a cache miss), then we have to both insert the new cache line into the cache and we have to insert the new page in the eTLB. To insert the new page, we do a L2 TLB lookup or a page table walk. We let the L2 TLB handle variable sized superpages ($> 4\,\text{kB}$), and transprently split them into eTLB-size pages at load (see Section 4.1).

However, we must also flush all the replaced page's cache lines on eTLB replacement, since all cache lines must have an valid entry in the eTLB. To do so, we walk through the

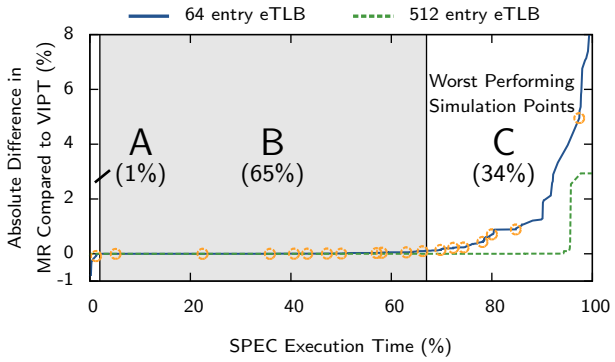


Figure 5: Absolute difference in miss ratio for unoptimized TLC compared to a VIPT cache for SPEC2006's execution, as a percentage of execution time across all SPEC benchmarks (gcc's simulation points are highlighted).

replaced page's CLT in order to locate and evict all the page's cache lines before inserting the new eTLB entry and cache line (Section 4 investigates how to minimize unnecessary cache evictions due to eTLB replacement).

3.3 Summary

The new Tag-less Cache design reduces energy by reading fewer bits from the TLB, eliminating cache tag reads, and by reading only one way from the data-array. This decreases the total number of bits that are read by 62% (Table 2) compared to a VIPT cache. To accomplish this we need to ensure that each cache line always has an valid entry in the eTLB, which requires a backpointer and page flushing on page replacement.

The design presented here is a fully functional virtual cache. To be a drop-in replacement for a VIPT cache it must also handle synonyms and coherency which we discuss in Section 5. We refer to this design (including the extra functionality in Section 5) as the *Basic TLC* in the rest of this paper.

4. MAKING THE TLC FASTER

The previous section outlined the Basic TLC design and illustrated how it can be used to reduce energy consumption by eliminating tags and only reading the correct way from the data-array. Still, there are many features that can be leveraged to improve performance and to save more energy. In this section, we look at a variety of different techniques to improve 1) the effectiveness (by reducing the miss ratio through better use of the eTLB), and 2) the efficiency (by improving the communication between the eTLB and the data-array).

4.1 Cache Effectiveness (Miss Ratio)

The cache effectiveness is how likely it is that an access hits in the cache (i.e., its miss ratio). This is typically affected by capacity (cache size), conflicts (associativity), etc. In TLC, a cache line can also be evicted due to eTLB replacement. Hence, the miss ratio is also affected by the eTLB size and its replacement policy as well. The miss ratio affects performance (since a high miss ratio increases access latency) and energy (since data must be fetched from

	Abs. MR. Diff vs. VIPT TLB		Abs. MR. Diff vs. VIPT Cache		Cache-line Evictions per eTLB Replacement		
	Avg	C	Avg	C	Avg	C	Max
LAD	0.67	1.02	1.07	2.58	0.74	1.11	2.14
LAD+LRU	**0.00**	**0.01**	**0.45**	**1.50**	**1.07**	**1.75**	**4.85**
LRU	0	0	0.55	1.74	1.88	2.58	5.72

Table 3: Average absolute difference in miss ratio compared to a VIPT cache and number of cache line evictions per eTLB replacement for different eTLB replacement policies.

the next level caches on a miss).

To study the TLC design, we use the SPEC2006 benchmarks with reference inputs, and we simulate 32 kB caches with 8-ways associativity using a Pin-based [4] cache simulator. For more details, see Section 6.1.

Figure 5 shows the absolute difference in miss ratio between Basic TLC and a VIPT cache for the 307 most prominent simulation points of all 29 SPEC2006 applications. These simulation points cover 80% of the full reference execution and are sorted by miss ratio. Note that this graph shows the full dynamic range of SPEC, and does not average together behavior within applications. This impact can be seen by examining the circled simulation points for gcc in Figure 5. As can be seen, they span the full dynamic range and would not be accurately represented by a single average.

Figure 5 shows that the miss ratio is approximately the same for 65% of SPEC's execution (Region B). In 1% (Region A), the miss ratio is actually lower for TLC compared to VIPT. Here, most of the cache lines that are evicted due to eTLB replacements are dead and will not be used again. This frees up space for new data, and we see a reduction in miss ratio. However, 34% of SPEC's execution (Region C) exhibit more cache misses from premature cache evictions due to eTLB evictions. That is, cache lines are evicted before they can be used again because their pages were evicted from the eTLB.

One way to reduce the number of cache line evictions due to eTLB replacement is to use a larger eTLB. For example, the green line (512 entry) in Figure 5 shows the absolute difference in miss ratio when using an 8× larger eTLB. With this many more entries, the miss ratio is approximately the same for 95% of SPEC's execution. However, this increases both the size and energy requirements. To improve the miss ratio without increasing energy and size, we first propose a new TLB replacement policy and then investigate different page granularities to minimize number of cache evictions.

LAD: Least Allocated Data Replacement

The goal of the new eTLB replacement policy is to minimize number of cache line evictions that are caused by eTLB replacements. To do so, we investigate a policy where the page with the Least Allocated Data (LAD) is replaced (i.e., the page that has the fewest cache lines in the cache). The amount of data in the cache for each eTLB entry can be calculated by summing the valid bits in the CLT or with a counter that is incremented when a new cache line is inserted.

Table 3 shows the difference in miss ratio for the eTLB and the cache compared to VIPT, and the number of cache lines that are evicted per eTLB replacement. The Avg col-

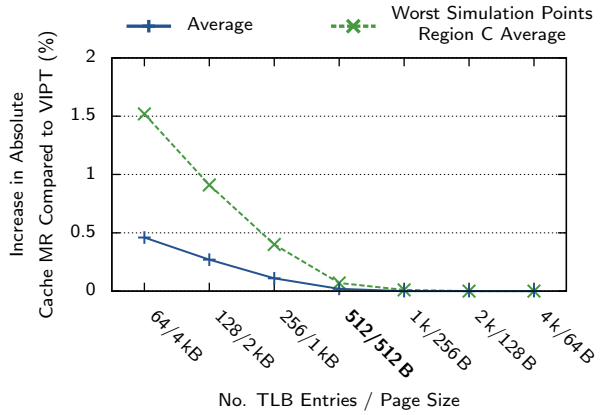


Figure 6: The average increase in cache miss ratio compared to a VIPT cache as a function of page size and number of eTLB entries.

umn shows the average across all SPEC, and the C column shows the average for the worst performing simulation points (Region C in Figure 5).

LRU TLB replacement evicts 1.88 cache lines per eTLB replacement on average (2.58 for Region C). LAD reduces number of cache line evictions to 0.74 cache lines per eTLB replacement (1.11 for Region C). However, both the eTLB and the cache miss ratio increase. The eTLB miss ratio increases because LRU handles temporal locality much better than LAD. The cache miss ratio on the other hand increases because LAD evicts pages that are about to be filled (i.e., the page is empty the first time it is used).

The LAD policy is therefore too aggressive on its own. Instead, we combine the basic LAD policy with LRU. To do so, LAD is only applied to the n least recently used pages. That is, the page with least allocated data (LAD) of the n least recently used (LRU) pages is selected for replacement. We varied n in an 8-way eTLB and found that selecting a page from the 3 least recently used pages with LAD produced the best results. This makes sure that newly loaded pages that are filling up with data stay in the cache (LRU), and that we minimize cache line evictions by evicting pages with the least data (LAD).

Table 3 shows that LAD+LRU's TLB miss ratio is essentially the same as for LRU replacement. However, the cache miss ratio decreases to only 0.45% above that of VIPT (1.5% for Region C). This is a modest decrease in miss ratio, but results in 18% reduction (32% for Region C) in evicted cache lines due to eTLB replacement.

Micro-Pages (Sparse Data)

Even with LAD+LRU, the miss ratio for TLC is still 0.45 percentage points higher on average and 1.5 percentage points higher for the worst simulation points. One of the reason for this is due to the discrepancy in number of entries in the eTLB and the cache (8× more cache entries than eTLB entries). Since every cache line must have an entry in the eTLB, the eTLB can become a bottleneck for applications with large data sets and sparse access patterns. For example, if only one cache line per page is used, the cache will only hold 64 cache lines out of 512 possible cache lines. The eTLB therefore needs to have more entries so that applications with sparse access patterns can utilize the whole cache.

Abs. MR. Diff vs. VIPT

		TLB		Cache		Storage	
		Avg	C	Avg	C	TLB	Total
4 kB	64 entry	0.00	0.01	0.46	1.50	2.66 kB	35.2 kB
	512 entry	-0.44	-1.19	0.12	0.38	21.2 kB	54.6 kB
512 B	512 entry	-0.07	-1.15	0.02	0.07	7.62 kB	40.9 kB
	Preloading	**-0.41**	**-1.28**	**0.04**	**0.08**	**7.62 kB**	**40.9 kB**
	15-ways	-0.41	-1.28	0.08	0.06	7.62 kB	38.4 kB
	VIPT					0.61 kB	34.5 kB

Table 4: Absolute difference in miss ratios compared to a VIPT cache, number of cache line evictions per eTLB replacement, and storage requirements for different page sizes and number of eTLB entries.

However, increasing the number of eTLB entries increases the size of the eTLB. For example, a 512-entry eTLB with 4 kB pages requires 8× more storage (21.2 kB vs. 2.66 kB) than a 64-entry eTLB. Most of this is due to the extra cache line location information in the eTLB. Instead of naively adding more entries to the eTLB, we use micro-pages (i.e., pages smaller than 4 kB) to minimize the eTLB size when we add more entries. A micro page contains fewer cache lines and therefore needs less cache line location information. For example, a 512-entry eTLB with 512 B pages needs 1/3 as many bits as a 512-entry eTLB with 4 kB pages (see Table 4).

The blue line (marked Average) in Figure 6 shows the average increase in cache miss ratio compared to VIPT for different number of eTLB entries and pages sizes. As expected, the miss ratio decreases with more TLB entries and smaller pages. The miss ratio starts to level off at 512 entries. At that point, the eTLB has the same number of entries as the cache. Meaning, there is no additional benefit of using more eTLB entries for optimizing sparse access patterns.

There are two additional benefits from micro-pages. First, micro-pages have a more space-efficient page utilization if only parts of the macro-page are used (i.e., they waste less space in the eTLB). Second, micro-pages provide better data to the eTLB replacement policy since the access pattern can be different in different parts of the page (i.e., replace the right micro-page instead of the full page).

Table 4 compares miss ratios and storage requirements. On average, a 512-entry eTLB with 512 B pages evicts 88% fewer cache lines due to eTLB replacement than a 64-entry eTLB with 4 kB pages. This decreases the cache miss ratio from 0.46% to only 0.02% more than VIPT (1.50% to 0.07% for Region C), which is essentially the same as VIPT.

The results from Figure 6 and Table 4 show that by using micro-pages, a TLC can achieve essentially the same cache miss ratio as a VIPT cache, while requiring only 36% as much storage as naively increasing the eTLB size. The total storage requirement increase by 16% from 35.2 kB to 40.9 kB, which will have a proportional effect on static power. To reduce this overhead, one could choose a design point with fewer eTLB entries (e.g., 256) in Figure 6.

Micro-Pages and Super-Pages

Note that micro-pages do not require additional operating system support. They can be transparently implemented in hardware by using regular or super-pages in the L2 TLB.

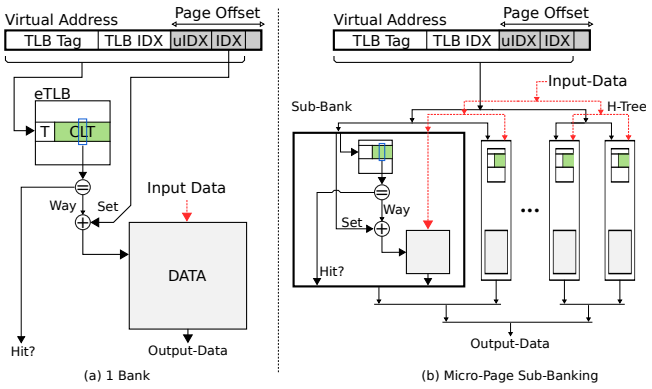


Figure 7: Micro-page sub-banking. The output from the eTLB goes directly to the data cache bank thereby eliminating the need for an output H-Tree.

On an eTLB miss, the micro-page is loaded by reading the corresponding page from the L2 TLB, and generating the appropriate micro-page. This is straight forward, since the micro-page is a trivial sub-set of the page found in the L2 TLB.

Macro-page Preloading

Table 4 shows that micro-pages also reduce the eTLB miss ratio. On average, the miss ratio is 0.07 percentage points lower than VIPT (1.15 for Region C) due to more eTLB entries. While this improves the miss ratio for most application, streaming data access patterns suffer. This happens because micro-pages need several eTLB misses to load a whole macro-page. For example, we need 8 eTLB misses to load a whole 4 kB page with 512 B micro-pages. To address this, we use a simple macro-page preloader. That is, on a eTLB miss, we load all micro-pages belonging to the same macro-page: For 512 B micro pages, we load all 8 micro-pages on a eTLB miss. To avoid evicting important data when preloading, the preloaded micro-page is installed into the LRU position (LAD position). It will then be evicted directly if it is not used since it has no allocated data.

The row marked Preloading in Table 4 shows that macro-page preloading reduces the eTLB miss ratio. This decreases the eTLB miss ratio further by 0.34 percentage points on average (0.13 for Region C). In total, the eTLB miss ratio is on average 0.41 percentage points lower than VIPT (1.28 for Region C). However, the cache miss ratio increases slightly as a result of macro-page preloading. This design point tradeoff needs to be considered when choosing the final design configuration.

Higher Associativity

Cache associativity is essentially free in the TLC design, since the correct way is directly found by reading the cache line location information in the eTLB. The associativity can therefore be easily increased by adding more bits to the cache line location information (i.e., $\log_2(\text{associativity})$). However, a better encoding can be used to increase the cache associativity without increasing the TLB size by combining the valid bit with the way index (see CLT in Figure 3). For example, by using 0000 to indicate that the cache line is not in the cache, and 0001 to mean that the cache line is in the first way. Thus, we can represent valid and 15-ways with only 4 bits. However, this decreases the cache size from 32 kB to 30 kB, since we loose one way.

In other words, we can nearly double the associativity (from n to $2n-1$) at the cost of loosing one way of capacity. The row marked 15-ways in Table 4 shows the difference in miss ratio and storage requirements. The miss ratio for the 30 kB, 15-way cache is essentially the same as a 32 kB, 8-way cache. However, the data-array is smaller. We can therefore use a smaller data-array without affecting the miss ratio by increasing the associativity. The smaller data-array can improve the access time by 6% and the read energy by 4%. The impact of this design point is presented in Figure 12 as "TLC-15W".

	Access Time (ns)			Energy (nJ)		
	per Bank	2-HT	1-HT	per Bank	2-HT	1-HT
1 Bank	0.2459			0.0082		
8 Banks	0.1377	0.1991	0.1844	0.0020	0.0062	0.0059
			-7%			-5%

Table 5: Access time and read energy for 1 and 8 banks, where the eTLB and the data-array are banked separately (2-HT), and together (1-HT) for a 32 kB, 8-way, 512 B micro-page TLC.

Summary

Cache lines can be evicted in the new TLC design due to eTLB replacements. To minimize number of cache line evictions, we use a smarter replacement policy to evict the page with least allocated data (LAD+LRU), and we use micro-pages to reduce cache line evictions caused by sparse data access patterns. Micro-pages however require more storage, but it can be alleviated by using a smaller cache but with higher associativity (Table 4), or by choosing a different micro-page configuration (Figure 6). These optimizations enable the TLC to achieve the same miss ratio as a traditional VIPT cache.

While there are clearly many different design points (e.g., micro-page size, number of eTLB entries, associativity, etc), we choose a 512-entry eTLB with 512 B micro-pages and macro-page preloading, as our *Optimized TLC* design.

4.2 Cache Efficiency (eTLB and Cache Communication)

The TLC design requires a way index lookup in the eTLB before the data can be accessed. This increases the latency compared to a VIPT cache that can access the cache in parallel with the TLB. To reduce this added latency we investigate how micro-page sub-banking can be used to reduce communication costs between the eTLB and the cache.

Caches are typically partitioned into smaller sub-banks in order to reduce wordline and bitline delays, and to improve bandwidth if each bank can be addressed separately. The smaller the sub-bank, the shorter the wordline and the bitline delays are. However, more sub-banks increase communication costs to and from each bank due to larger H-Trees.

If we bank the eTLB and the data-array the same logical way we can access them serially. That is, the output of each eTLB bank will go directly and exclusively to the associated data bank, thereby eliminating the eTLB output H-Tree. We use this property to reduce communication costs between the eTLB and the data-array by banking both the eTLB and

the data-array at micro-page granularity. This enables us to tightly integrate the eTLB with the data-array by combining the eTLB and the data-array sub-banks together. Each sub-bank is then addressed with the micro-page index bits from the virtual address using a single H-Tree.

Figure 7 shows an overview of the sub-banking design. With micro-page sub-banking, the cache line location information is sent directly from the eTLB to the data-array without exiting the sub-bank. This lowers the latency and saves energy compared to a design where the eTLB and the data-array are banked separately, since that would require the cache line location information to be transmitted across banks.

Table 5 shows the access time and read energy with 1 and 8 banks for a 32 kB, 8-way, 512 B micro-page, TLC cache using the CACTI [12] cache simulator. The Bank column shows the access time and energy per bank, and 2-HT and 1-HT columns show the total cache access time and read energy for an separately banked cache (2-HT), and an integrated banked cache (1-HT).

The results show that the latency and the dynamic energy consumption decreases with sub-banking. The access time and energy is further reduced by integrating the eTLB and the cache together due to faster communication between the eTLB and the cache. By sending the cache line location information directly from the eTLB to the cache we reduce the latency by 7% and the read energy by 5% .

5. MAKING THE TLC COMPATIBLE

The Basic TLC and the Optimized TLC described so far are fully functional virtual caches. However, the goal of the TLC design is for it to be a drop-in replacement to a VIPT cache. That means, it should be possible to use the TLC design without requiring modifications to the processor core or software.

Many techniques have been proposed to deal with synonyms, homonyms and coherency in virtual caches [11, 14], and can be applied to the TLC design. In this section, we discuss how we deal with these issues, and we propose a simple method tailored to the TLC design to handle synonyms.

Synonyms

Synonyms occur when several virtual addresses are mapped to the same physical address. This is not a problem for a VIPT cache because of the way it handles cache line replacements. In a VIPT cache, all synonyms access the same set since the cache is indexed with bits from the page offset, which are the same for virtual and physical addresses. The physical tags therefore solves the synonym problem since the replacement is handled within the set.

The TLC design on the other hand stores cache line location information in the eTLB, which needs to be updated on replacements. Since the eTLB is virtually indexed, several synonyms pointing to the same data in the cache could exist at the same time, and they could be located in different sets and ways. However, all synonyms must be updated on a cache line replacement for the cache to remain consistent. This is both energy inefficient and time consuming. Fortunately, synonyms are rarely used in practice and most cache accesses are done through a single virtual address [1]. We therefore do not allow multiple synonyms in the eTLB at the same time, which means that we only have to update one eTLB entry on a cache line replacement.

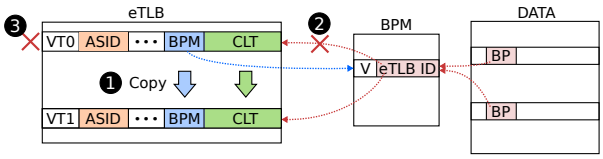


Figure 8: Synonym management.

To quickly find synonyms, we consult a reverse TLB (BLT)[4]. that translates physical address to virtual address. The BLT contains information for all micro-pages in the eTLB. A hit in the BLT therefore implies that a synonym exists.

To guarantee that there are no synonyms in the eTLB, we check for synonyms in the BLT during eTLB replacement and we evict the old synonym before inserting the new page into the eTLB and the BLT. The old synonym's cache line location information is simply copied to the new eTLB entry, as both pages map the same physical data (since they are synonyms).

In addition to copying the cache line location information from the old synonym to the new synonym, the cache lines' backpointers (used during replacement) must also be updated with the new eTLB location. However, that requires several cache accesses since a page can have several cache lines in the cache. To reduce number of cache access, we add a level of indirection (a backpointer map, or BPM) which allows us to update all the page's cache lines at once without accessing the cache.

Figure 8 shows how we handle synonym replacements. A synonym is replaced by: 1) copying the BPM pointer and the cache line location information to the new eTLB entry for the synonym, 2) updating the BPM (indexed with BPM in the eTLB) with the new eTLB location, and 3) removing the old synonym. Note that the data is not moved, we only update the eTLB information.

To replace a cache line, its cache line location information in the eTLB is updated by first looking up the right eTLB entry in the BPM, before accessing the eTLB. This adds extra latency, but is off the critical path since its only done during a cache miss.

In summary, removing synonyms in the eTLB minimizes number of eTLB accesses during cache line replacements, and using a single level of indirection eliminates cache accesses during synonym replacements.

Homonyms

Homonyms occur when the same virtual address are mapped to different physical addresses (e.g., two applications accessing different locations in memory but through the same virtual addresses). In order to not flush the eTLB (hence the cache), on context switches, we use an Address Space ID (ASID), to separate different memory domains. This could potentially increase the storage requirements, but, most processors already implement ASIDs for hardware virtualization support, so this is unlikely to add more space in practice.

Cache Coherence

Communication with the next level caches and coherency request are typically done via physical addresses. Since we have no cache tags to search the L1 cache, all cache accesses must go through the eTLB. However, the eTLB is virtually

[4]We model the BLT as a fully associative CAM.

	Little	Big
Frequency	2GHz	4GHz
Width / ROB / LSQ / Registers	2 / 32 / 16 / 64	8 / 192 / 64 / 256
L1 Instruction Cache	32kB, 8-way, LRU, 4c	
L1 Data Cache	32kB, 8-way, LRU, 4c	
L2 Unified Cache	512kB, 8-way, LRU, 12c	256kB, 8-way, LRU, 7c
L3 Cache	-	8MB, 16-way, LRU, 20c
Memory	2GB DDR3 11-11-11	
L1 TLB / eTLB	64 entries, 4kB Pages, 8-way, LRU	
L2 TLB	512 entries, LRU, 7c	

Table 6: Baseline processor configurations

indexed while the incoming requests are based on physical addresses. To handle this, we use the reverse TLB (BLT) to translate the request's physical address to the current synonym's virtual address in the eTLB (see Figure 4e). The coherency request is then resolved reading the way index from the eTLB. Note that extra pressure will be put on the eTLB ports, however, it is no worse than for the baseline tag-array.

The BLT increases the latency for coherency requests that hit in the L1 cache. However, the reverse TLB and the L1 cache access only have to be as fast as the L2 cache lookup. Furthermore, all pages must have an entry in the BLT, and all cache lines must have an entry in the eTLB. This means that the BLT is effectively also a snoop filter. Requests can therefore be dropped if they miss in the BLT since that guarantees that the cache does not contain the data.

Summary

To make the Basic TLC and the Optimized TLC design drop-in replacements for a standard VIPT cache, they must handle synonyms, homonyms, and coherency requests. To do so efficiently, we add two new structures (a BPM and a BLT). Both structures are off the critical path, since they are only accessed during misses, and will therefore not affect the performance.

6. RESULTS

The previous section explored different design decisions for minimizing cache line evictions due to eTLB replacements, compatibility with VIPT caches and better communication between the eTLB and the data-array. In this section, we finalize the TLC design and evaluate performance and energy.

6.1 Methodology

We used the Gem5 x86 simulator [3] to evaluate the performance. Table 6 shows the baseline configurations. To better understand the performance impact, we evaluate two different configurations. A small, simple, out-of-order core (little) and an aggressive out-of-order core (big) with a deeper cache hierarchy. The little core runs at 2 GHz, has a 2 level cache hierarchy, and targets mobile devices. The big core runs at 4 GHz, has a 3 level cache hierarchy, and is representative of high-end server processors.

To evaluate the TLC, we used the SPEC2006 [7] benchmarks with reference input. We used a phase detection tool [17] to find the largest phases [19] in each benchmark. Each phase was then simulated for 100 M instructions, and the results were scaled with the length of each phase[5].

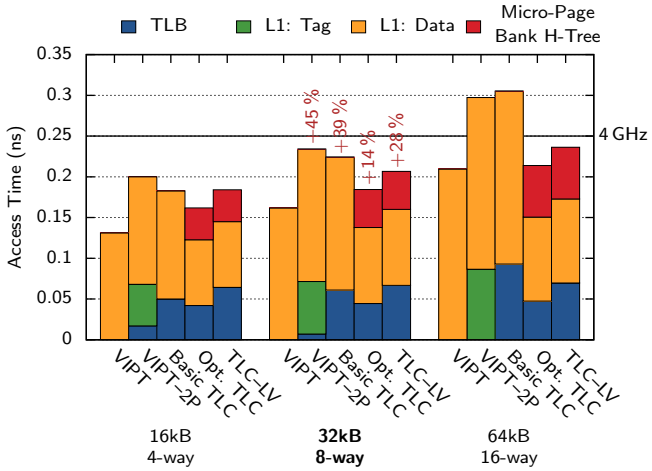


Figure 9: Latency breakdown.

We used the CACTI 6.5 [12, 10] cache simulator with a 22 nm process to compute access times and energy. We used one read-write port, high-performance transistors for first level caches and low static power transistors for second level caches, and we optimized for access times. For the CACTI studies, we also show results for 4-way, 16 kB with 32-entry TLBs and 16-way, 64 kB with 128-entry TLBs caches. The associativity is different since the cache size is a function of associativity in VIPT caches (i.e., cache size = associativity × page size). However, we focus the evaluation on the 32 kB caches we have studied throughout the paper.

We compare the TLC design with a standard VIPT cache with parallel lookup (VIPT) and a phased VIPT cache with 2-phase lookup (VIPT-2P). The TLC design has three versions, Basic TLC, Optimized TLC, and TLC-LV. The Basic TLC has none of the optimizations from the previous sections. The Optimized TLC uses LAD+LRU replacement, 512 B micro-pages, macro-page preloading, and micro-page sub-banking. Note that we only use micro-page sub-banking for better communication between the eTLB and data-array. Both TLC and VIPT support the same number of concurrent accesses. TLC-LV is the same as Optimized TLC but includes the last-value page-prediction discussed later in Section 8 that bypasses the eTLB for reduced energy on a last-value hit.

6.2 Access Time

Figure 9 shows access time breakdowns for the different caches. Overall, VIPT has the lowest latency since the access time is only due to the data-array. VIPT-2P's TLB component shrinks with larger caches. At 16 kB and 32 kB, the tag-array has to wait for the TLB translation before it can proceed with the tag comparison. However, at 64 kB, the TLB is fast enough to send the address translation to the tag-array before the tags arrive to the tag-comparator. The access times is therefore bound only by the tag-array

[5]Due to Gem5 limitations we could not simulate all phases. The results section therefore uses a subset (202 / 307) of the phases that were used in the Pin-based studies.

and the data-array at 64 kB.

The Basic TLC is faster than VIPT-2P at 16 kB and 32 kB caches, but slightly slower at 64 kB. The eTLB grows faster in size with more entries than VIPT-2P's tag-array. This eventually results in a longer access time at 64 kB, since the number of TLB entries are also doubled when the cache size is increased.

The access time is lower for the Optimized TLC than the Basic TLC. This is due to micro-page sub-banking, which reduces bitline and wordline delays, and communication delays between the eTLB and the data-array. However, a large fraction of the access time is now spent in the micro-page sub-banking H-Tree (Note that the internal H-Tree for CACTI's sub-banking is included in the Data latency).

6.3 Performance

Figure 9 shows that all the caches are fast enough to support a one cycle access latency for a 4 GHz processor with 32 kB cache sizes. We therefore use a 4 cycle latency (including address calculations, etc.) for all caches in the performance simulations. The performance results therefore depend only on the cache miss ratio, since the hit access time remains the same. Note that VIPT-2P is assumed to have the same miss ratio as VIPT, and TLC-LV the same miss ratio as Optimized TLC. We therefore only show results for VIPT, Basic TLC and Optimized TLC.

Figure 10 shows the relative CPI difference in percent compared to a VIPT cache for SPEC's execution as a percentage of execution time across all SPEC benchmarks. The Basic TLC has a higher cache miss ratio since it evicts more cache lines due to eTLB replacements. The little core has a smaller reorder buffer and can not hide the misses as well as the big core, resulting in a average slowdown of 0.75% on the little core but a speedup of 0.14% on the big core.

The leftmost part of the big core's CPI curve shows a noticeable speedup (9.6%). This is probably due to evicting dead cache lines early, and having more out-of-order execution and memory level parallelism. This highlights the importance of evaluating different core types (big vs. little).

The Optimized TLC reduces the number of cache misses by using a smarter eTLB replacement policy that evicts less data, and by using micro-pages with more eTLB entries for handling sparse access patterns. This removes nearly all performance differences, and we see a flat line across whole SPEC. In addition, is also has a lower eTLB miss ratio and we see a slight speedup by 0.12% (little) and 0.11% (big) on average. This shows that by applying these techniques, we can achieve the same or better performance than a VIPT cache.

6.4 Read Energy

Figure 11 shows the read hit energy. VIPT consumes most energy since it reads all ways. VIPT-2P reduces the energy by 37% by only reading one way from the data-array. However, the TLB and tag-array energy remains the same. The Basic TLC stores the cache line location information in the eTLB and can therefore eliminate the tag-array. However, the TLB energy increases slightly since the eTLB store more information, but the total energy is still lower than the phased VIPT-2P and 42% lower than parallel VIPT.

The Optimized TLC reduces the read energy further down to 52% of VIPT by decreasing bitline and wordline energy, and by reducing communication costs between the eTLB

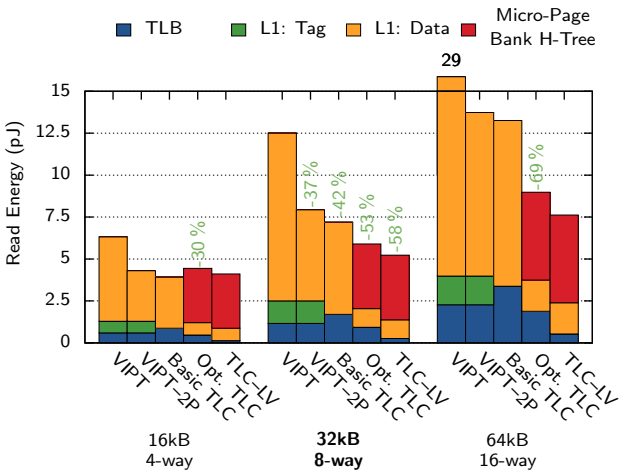


Figure 11: Dynamic energy breakdown.

and the data-array with micro-page sub-banking. However, a large part of the energy is now due to the H-Tree (CACTI's internal sub-bank H-Tree for the data arrays is included in the Data part).

For the other two cache sizes, the energy is reduced by 30% for the 16 kB cache, since it is smaller and has less assosiativity, and by 69% for the 64 kB cache, due to larger size and higher associativity.

Note that this analysis is conservative. We only read a word (64 bits) from the cache. Many processors support load operations (e.g., SSE [8]) that load several words from a cache line. Such a design would increase the energy difference since the TLC design would read even fewer bits compared to a parallel VIPT cache.

6.5 Runtime Energy

Figure 12 shows the aggregate dynamic runtime energy (i.e., number of reads and writes multiplied by the read and write energy for each cache structure) across SPEC. The energy is normalized to a parallel VIPT cache. The figure does not include L3 energy since it is similar for both VIPT and TLC. TLC's energy has been broken down further into sub-components. The L2 cache contributes more to the little core's energy since it is larger than the big core's L2 cache.

Most energy is spent in the micro-page sub-bank H-Tree (43% little and 45% big), followed by the data-array (22% little and 23% big) and the eTLB (17% little and big). Some applications have noticeable higher BLT energy, due to more eTLB misses and macro-page preloading (since the BLT needs to be access 8 times on every eTLB miss to detect synonyms).

The rightmost applications (at 98-100%, e.g., mcf) have very high TLB miss ratios and still evict many L1 cache lines due to eTLB replacements. This in turn results in more L2 accesses, and we see an increase in both BLT and L2 energy.

The Optimized TLC reduces the dynamic runtime energy more than phased VIPT-2P on average. In total, the energy is reduced by 78.1% down to 21.9% of a VIPT cache on both the little and big cores. This is significantly more than the reduction in read hit energy in Figure 11, and is largely due to not reading the cache on misses.

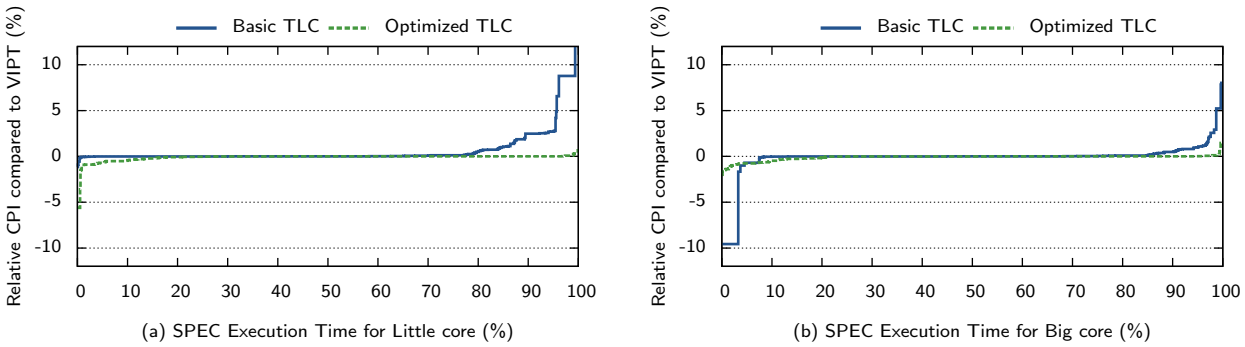


Figure 10: Relative CPI compared to VIPT.

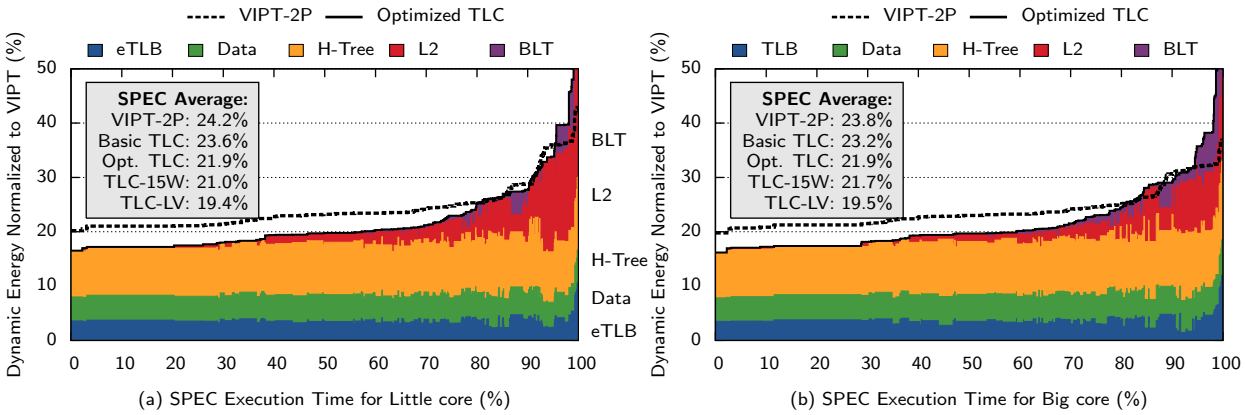


Figure 12: Dynamic runtime energy normalized to parallel VIPT.

6.6 Summary

The results show that the TLC design is: 1) fast enough to support clock frequencies beyond 4 GHz, 2) can achieve the same performance (miss ratio and CPI) as a traditional VIPT cache, and 3) uses 78% less dynamic runtime energy than a VIPT cache.

7. RELATED WORK

In this section we look at related work on how to minimize number of bit-reads, and how cache lines can be tracked through course-grained data structures.

Course-grained tracking. The idea of tracking cache lines through courser grained data structures (e.g., a page) is not new. Zebchuk et al. [21] use a Region Vector Array (similar to our eTLB), to locate where cache lines are located in the L2 cache, to reduce tag area, eliminate snoops and for prefetching.

In contrast to our their work, we focus on how course-grained tracking can be applied to first level caches, and how any accompanying issues (e.g., virtual addresses, synonyms, etc) can be solved. Moreover, the key benefit and goal of our work is lower dynamic energy, which is much larger for L1 caches than L2/L3 caches, due to L1 filtering.

Pseudo-associative caches. Most of the data-array energy in a parallel VIPT cache is due to reading all ways. One way to reduce the energy is to simply use fewer ways. However, that typically decreases performance by incurring more cache misses. A significant amount of work has addressed this issue by adding pseudo-associativity on top of direct-mapped or low-associative caches [18, 6, 15, 16]. That is, the cache behaves like it has more associativity than it actually has. This is typically done by using different hashing functions for different ways or sets and by combining it with the ability to move cache lines between sets instead of evicting them.

However, these techniques are limited to physically indexed caches (e.g., PIPT, and last level caches), since the associativity typically determines the L1 cache size for virtually indexed caches such as VIPT and TLC (i.e., cache size = associativity * page size). This makes it hard to reduce the associativity without reducing the cache size.

TLB optimizations. Much work has also focused on reducing the TLB energy by using virtual caches (e.g., VIVT). The TLB is then only accessed on L1 cache misses. However, this typically introduces problems with synonyms etc. To deal with this, various software [11] and hardware [14, 1] techniques have been proposed. Instead of reducing the TLB energy, we amortizes the TLB energy by using some of that energy to reduce the data-array energy and to remove the tag-array.

Way-prediction. Way-prediction is used to reduce data-array energy by predicting which way to read [5, 2, 13, 9]. On a correct prediction, only the correct way is read. However, miss-predictions incur extra energy and latency.

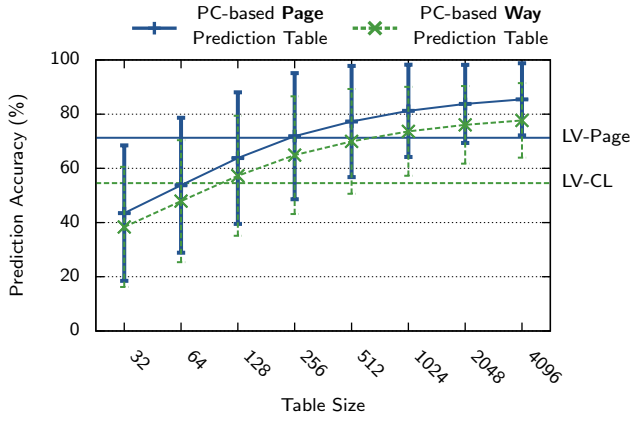


Figure 13: The average prediction accuracy for last-value prediction, and PC-based table predictors as a function of table size.

To predict the correct way, cache line location information is cached in a separate prediction table. A prediction is then made by looking up the right way in the table. However, the prediction must be made before the effective address is known. To do so, it must base the prediction on information from earlier pipeline stages (e.g., program counter).

Way-prediction is complementary to our work and we can use it to reduce parts of the latency and energy. However, we already know which way a cache line is located in by reading the cache line location information in the eTLB, so there is no benefit in predicting cache lines. We demonstrate how to leverage prediction to reduce the eTLB's latency and energy in the next section.

8. FURTHER TLC OPTIMIZATIONS

The TLC design requires a way index lookup in the eTLB before the data can be accessed. To save more energy (and potentially reduce the latency), we can predict the eTLB entry with the techniques from way-prediction. However, there are two downsides to this. First, the prediction table wastes energy since it is read on every memory access, and it needs to be updated on miss predictions. Second, the processor's core pipeline must be integrated with the cache to provide information from earlier pipeline stages.

Instead, we propose a simple last-value predictor that has similar accuracy, but is more energy efficient and much easier to implement. A last-value register is added to the design, which holds the last accessed page's cache line location information (CLT + tag + permission bits) in each bank. This allow us to bypass the eTLB whenever the same page is used again, which eliminates eTLB read energy and latency. Furthermore, we can handle several different access streams if they access different banks. For example, up to 8 access streams for an 8-bank eTLB.

Figure 13 shows the prediction accuracy. The two horizontal lines shows the accuracy for last page prediction (LV-Page) and last cache-line prediction (LV-CL). These structures use one register per bank and do therefore not have a table size parameter. The results show that a simple last-value page prediction has the same accuracy as a 512 entries PC-based way-predictor. However, it is more energy efficient (no prediction table reads and fewer eTLB reads) and much easier to implement. Overall, predicting the page is better than predicting the way due to spatial locality.

The TLC-LV bars in the results section show the result for a TLC with last-value eTLB prediction. We only use this technique for reducing the eTLB energy to avoid complications with variable access times. The results show that TLC-LV is slightly slower than TLC in Figure 9 because it has to first check if the eTLB can be bypassed. However, it reduces the energy beyond the Optimized TLC as seen in Figure 12.

9. CONCLUSIONS

In this paper, we have presented a new Tag-less Cache (TLC) design. It augments the TLB with extra way index information, thereby enabling us to: 1) eliminate extra data-array reads, 2) eliminate the tag-array), 3) filter out cache misses, and 4) amortize the TLB lookup energy.

Several performance and energy improvements were proposed and evaluated in order improve the effectiveness by reducing number of cache line evictions due to eTLB replacements, and to improve the efficiency by reducing the communication cost between the eTLB and the data-array. Finally, a simple, yet effective, last-value page predictor was proposed to reduce eTLB energy further. The TLC design efficiently handles synonyms and coherency in order to make it a drop-in replacement for a standard VIPT cache. The final results show that the TLC design can reduce the dynamic energy by 78% without affecting performance (CPI and miss ratio).

10. REFERENCES

[1] A. Basu, M. D. Hill, and M. M. Swift. Reducing Memory Reference Energy with Opportunistic Virtual Caching. In *Proc. International Symposium on Computer Architecture (ISCA)*, 2012.

[2] B. Batson and T. N. Vijaykumar. Reactive-Associative Caches. In *Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 Simulator. *SIGARCH Comput. Archit. News*, 2011.

[4] 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. 2005.

[5] B. Calder, D. Grunwald, and J. Emer. Predictive Sequential Associative Cache. In *Proc. International Symposium on High-Performance Computer Architecture (HPCA)*, 1996.

[6] E. G. Hallnor and S. K. Reinhardt. A fully associative software-managed cache design. In *Proc. International Symposium on Computer Architecture (ISCA)*, 2000.

[7] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News*, 2006.

[8] Intel Corporation. *Intel® Architecture Instruction Set Extensions Programming Reference*, August 2012.

[9] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. 2008.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated

Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2009.

[11] W. L. Lynch. The Interaction of Virtual Memory and Cache Memory.

[12] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. Tech. rep., Hewlett Packard Labs, 2009.

[13] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2001.

[14] X. Qiu and M. Dubois. The Synonym Lookaside Buffer: A Solution to the Synonym Problem in Virtual Caches. *IEEE Trans. Comput.*, 2008.

[15] M. K. Qureshi, D. Thompson, and Y. N. Patt. The V-Way Cache: Demand Based Associativity via Global Replacement. In *Proc. International Symposium on Computer Architecture (ISCA)*, 2005.

[16] D. Sanchez and C. Kozyrakis. The ZCache: Decoupling Ways and Associativity. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2010.

[17] A. Sembrant, D. Eklov, and E. Hagersten. Efficient Software-based Online Phase Classification. In *Proc. International Symposium on Workload Characterization (IISWC)*, 2011.

[18] A. Seznec. A Case for Two-way Skewed-associative Caches. In *Proc. International Symposium on Computer Architecture (ISCA)*, 1993.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proc. Internationl Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[20] A. Sodani. Race to Exascale: Opportunities and Challenges. In *MICRO 2011 Keynote*, 2011.

[21] J. Zebchuk, E. Safi, and A. Moshovos. A Framework for Coarse-Grain Optimizations in the On-Chip Memory Hierarchy. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2007.

[22] C. Zhang, X. Zhang, and Y. Yan. Two Fast and High-Associativity Cache Schemes. *Micro, IEEE*, 1997.