ZCache Skew-ered

Richard Sampson Thomas F. Wenisch
Advanced Computer Architecture Lab
University of Michigan

Abstract

Reducing the number of last-level cache misses in multicores while maintaining low on-chip hit latencies remains a critical challenge. The recently proposed ZCache improves on-chip hit rates by indexing via a sophisticated H_3 hashing function and selecting victims through a multi-level replacement scheme. By using different hashing functions in each way, ZCache, like earlier skewed-associative caches, ensures that different subsets of addresses conflict in each way for a given index. The diversity of replacement candidates allows ZCache to provide an effective associativity greater than the number of physical ways while still having low look-up costs.

In this paper, we deconstruct and re-examine the various differences between ZCache and earlier proposals for skewed-associative caches using commercial server applications, multi-threaded applications, and scientific workloads. We find that: (1) ZCache's H_3 hash functions are unnecessarily complex—previously proposed skewed indexing schemes provide similar performance with less overhead; (2) two-level replacement provides useful energy/performance gains for commercial server applications, but additional replacement levels may not be justified; and (3) ZCache's timestamp-based replacement scheme is its most critical innovation—when this replacement scheme is integrated into a skewed-associative cache, the design achieves most of ZCache's performance gains without the need for multi-level replacement.

1. Introduction

Reducing the number of last-level cache misses in multicores while maintaining low on-chip hit latencies remains a critical challenge. Modern multicores typically have highly-associative (16-way or higher) last-level caches to mitigate conflict misses [7]—misses that arise when numerous frequently-accessed blocks map to the same set in the cache. However, high associativity comes at a price in static power (due to larger tags) and access latency/energy (due to set-associative lookup).

To mitigate conflict misses in low-associative caches, Seznec proposed a new cache indexing scheme called *skewed associativity* [20]. By using simple hashing functions that mix index and tag bits to map an address to distinct sets in each way of the cache, skewed-associative caches ensure that addresses that conflict in one way do not conflict in others. Over time, blocks that conflict in one way settle into non-conflicting locations in other ways, reducing conflict misses and increasing the effective associativity of the cache. However, skewed-associative caches complicate cache replacement; least-recently-used (LRU) replacement is difficult to implement and past skewed-associative designs have relied on simple not-recently-used (NRU) schemes [20, 21].

ZCache [17] further builds on this body of work to provide even greater effective associativity from a low-associative (e.g., 4-way) cache. ZCache combines sophisticated H_3 hash-based indexing, a new bucketed pseudo-LRU replacement algorithm, and a multi-level replacement scheme inspired by cuckoo hashing [13]. Instead of evicting a victim block from the cache, in ZCache, a victim block can displace yet other victims to alternate locations, thereby allowing a large number of replacement candidates to be considered. This cascading replacement is possible because the hash function maps each block to distinct alternate sets, enabling better cache utilization without increasing the number of ways. However, this flexibility comes at a cost in implementation complexity and additional energy to shift blocks between locations.

In this paper, we deconstruct the ZCache design and evaluate each of its differences from prior skewed-associative designs. In particular, we contrast ZCache's H_3 based hash with simpler skew functions, we examine the hit rate and energy trade-offs of multi-level replacement, and we consider the impact of ZCache's bucketed LRU replacement. First, we show that the H_3 hashing functions used by ZCache are unnecessarily complex and equivalent performance is possible using the simpler skew function. Second, we demonstrate that, for commercial applications, two-level replacement provides a useful power/performance trade-off, making it a good candidate for designs geared towards server workloads; however, additional replacement levels provide negligible improvement for the added complexity and energy. Moreover, we find that multi-level replacement is not necessary for scientific workloads. Third, we show that ZCache's bucketed LRU with skew provides substantial benefit over Seznec's 2-bit NRU scheme, and is a preferred pseudo-LRU

algorithm for skew-associative caches. Finally, we propose small enhancements to skewed-associative indexing to improve its impact in large caches.

The rest of this paper is organized as follows. In Section 2, we describe skewed-associativity and ZCache. In Section 3, we present our evaluation methods, and discuss our results in Section 4. In Section 5, we discuss related work and finally, we conclude in Section 6.

2. Background

2.1 Skewed Associativity

Skewed-associative caches [20,21] were proposed to reduce conflict misses by using different mappings for each way of a set-associative cache. Each mapping function mixes low order tag bits into the index under a different permutation, ensuring that addresses conflicting in one way do not conflict in the other ways. As a result, more distinct combinations of blocks can be concurrently cached. Over many accesses, conflicting blocks tend to spread out to alternative non-conflicting locations, resulting in a significant reduction of conflict misses over conventional set-associative lookup [3].

The main drawback of skewed-associative caches is that one cannot readily implement a least recently used (LRU) replacement algorithm for them. Because the set of candidate victim blocks varies for each lookup, exact LRU can only be implemented by maintaining large last-access timestamps for each block, an impractical solution. Instead, Seznec proposed a complexity-effective 2-bit not-recently used (NRU) replacement scheme [21].

The skewed indexing scheme adds minimal hardware, requiring only static bit rotations and a single level of XOR gates. An access address is divided into four segments, A_3, A_2, A_1, A_0, where A_0 are the block offset bits, A_1 the conventional n-bit index, A_2 the n low-order bits of the tag, and A_3 the remaining tag bits as shown in Figure 1(a). The index skewing function $f_i(A)$ of way i is given by:

$$f_i(A) = \sigma^i(A_2) \oplus A_1$$

where σ is the perfect shuffle operation shown in Figure 1(b). Since binary perfect shuffles require only a permutation of the wires, the skewed lookup introduces only a single level of XOR gates in the critical path of each way.

Skew lookup in large caches. Skewed lookup relies on variability in A_2 to spread otherwise-conflicting blocks across sets of the cache. It is most effective when there is a great deal of variability in all bits of A_2. However, present-day L2 caches have far more sets than when skewed lookup was proposed, and higher-order tag bits tend to vary little. To introduce better distributions in the skewing, we modify $f_i(A)$ to duplicate the lower half of A_2 into its upper half, yielding A_2'. We find that this modified skewing function gives consistently better performance given the footprints (up to 1GB) of the applications we study.

2.2 ZCache

ZCache [17] at its core is similar to skewed-associativity, as it uses a hashing function to create distinct indexing functions for each way. In addition, ZCache introduces a multi-level replacement scheme, inspired by cuckoo hashing [13], to consider more candidates for replacement than there are ways in the cache, without affecting the critical path of cache lookup. Finally, ZCache makes use of a bucketed-LRU replacement algorithm that provides near-LRU performance with reasonable overhead.

ZCache uses sophisticated H_3 hash functions [4, 15] to create distinct index functions for each cache way. In the H_3 hash, shown in Figure 1(c), n address bits select multiple rows of a pre-generated n by m binary matrix, where n is the desired number of selection bits, and m is the length of the index. The matrix is generated randomly with each bit having a 50% probability of being set [19]. The selected rows (for which the corresponding address bits are set) are passed to an n by m XOR array, which sums each column to produce an m-bit index. Distinct index functions are created by using a different random matrix for each way. The H_3 hash generally performs better for larger n; we use all available address bits (all bits above the block offset) to maximize variation. The hardware overheads of a generic H_3 algorithm are considerable; each way requires matrix storage, selection logic, and an XOR array, all of which lie on the cache access critical path. By pre-selecting and hard-coding the hash functions for each way, these hardware overheads can be reduced (e.g., [19] demonstrates an implementation with a five-level tree of two-input XOR gates). In our work, we assume the hard-coded design; nonetheless, even these optimized H_3 implementations have greater hardware overhead than skewed indexing.

ZCache also introduces two-level and three-level replacement mechanisms, which consider more candidates for eviction as shown in Figure 1(d). The key idea of this multi-level replacement scheme is to displace (rather than evict) a victim block to one of its alternate locations, which in turn might displace another block, until the best candidate for eviction is found. When a victim block must be selected, an initial set of victims are identified as in conventional single-level replacement. In a 4-way cache, there are four such possible victims. Then, a second level of possible victims is identified by hashing each of these first four candidate addresses, providing 12 more candidates. These can be hashed a third time, yielding 52 total replacement candidates. The eldest block of all of the candidates is evicted from the cache, and

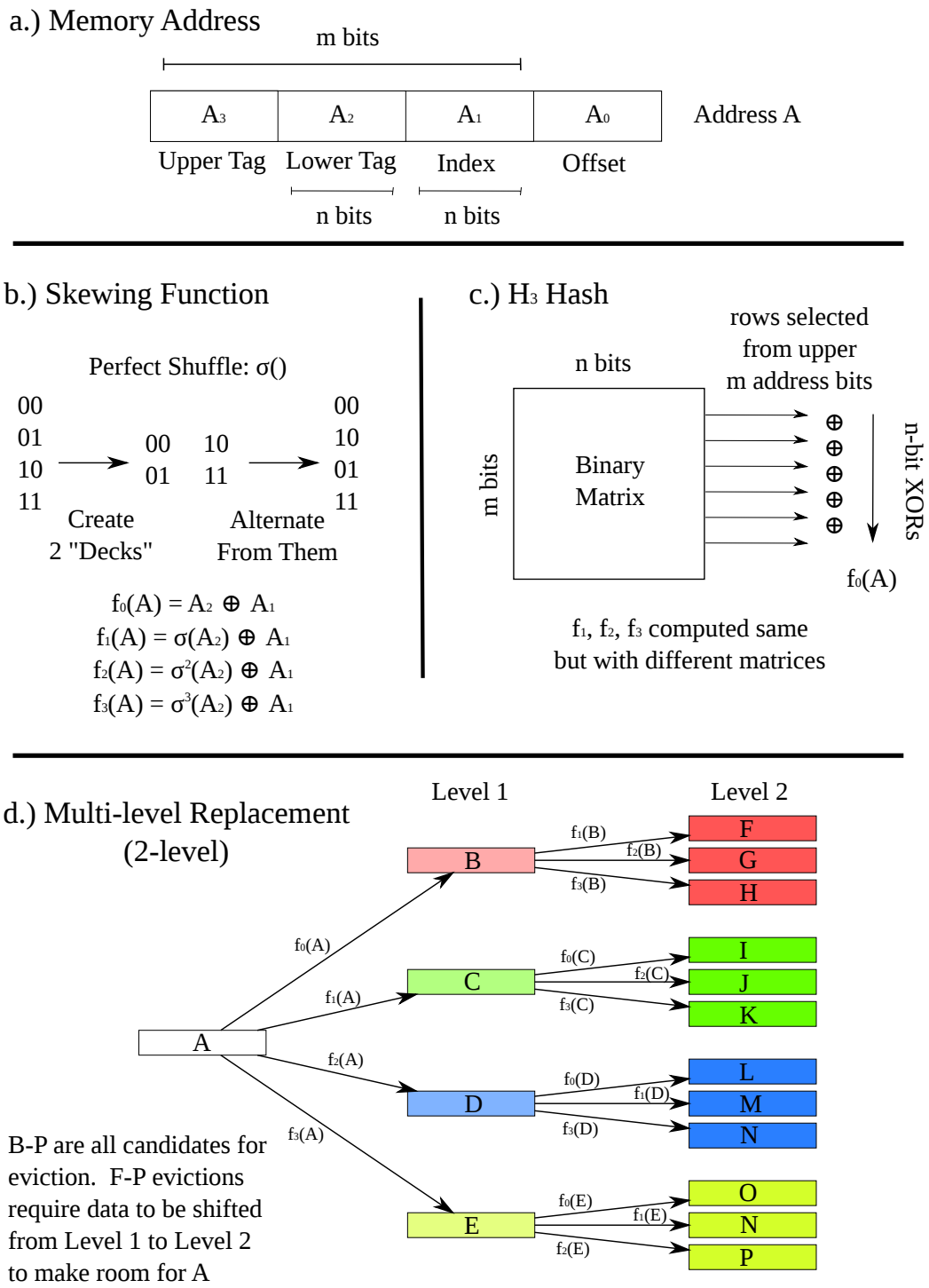

Figure 1: *a.) Decomposition of address bits for skew and hash algorithms, b.) Description of skewing with "perfect shuffle" function, c.) Description of H_3 hashing, d.) Diagram of multi-level replacement, which can be used with either indexing method.*

the remaining blocks are shifted as needed. The search for a victim and swap operations are performed while an off-chip miss is pending, and hence are off a miss's critical path.

Finally, much like skewed-associativity, ZCache cannot practically implement LRU replacement. Instead, it introduces a novel bucketed-LRU replacement scheme, where a small n-bit last-access timestamp is maintained per cache line. Every k accesses, the current time is incremented. When a victim block is needed, the blocks are partially-ordered according to their timestamps and a victim is selected from the eldest group. We use $n = 8$ and $k = 5\%$ of cache size as suggested in [17].

3. Methodology

3.1 Infrastructure

We evaluate ZCache and skewed-associative designs using trace-based simulations with Flexus 3.0 [24]. We simulate a 16-core system with 8MB shared L2 cache and 64kB private L1 caches. In our trace-driven model, each core executes at a fixed IPC of 1; we do not model any stalls within the simulation. For energy calculations, we determine the total stall time due to L1 and L2 cache misses from average miss rates (i.e., assuming in-order cores). Hence, our evaluation focuses on L2 miss rates (reported in misses-per-1000-instructions) and average power rather than exact performance estimates. Our traces include both L2 instruction and data accesses. Table 1 summarizes our simulation parameters. We include baseline results for 4-way, 16-way and fully-associative L2s and consider ZCache and skewed-associative designs that are implemented with 4-way lookup, to ensure our results are comparable to prior work.

3.2 Workloads

We study a selection of multithreaded applications including commercial server applications, scientific applications (em3d and moldyn) and PARSEC [2]. Table 2 lists our workload suite. Prior work on ZCache [17] also considered multi-programmed workloads. Numerous recent studies have considered schemes for staticly and dynamically partitioning cache capacity in multi-programmed workloads [10, 16, 22]; we limit our study to single-application multi-threaded programs to simplify the analysis. However, recent work has suggested that multi-level replacement allows for more intelligent partitioning schemes [18]. We execute each benchmark for either 400-million or 800-million instructions per CPU, or, in the case of the scientific applications, seven to nine iterations. We warmed caches for 200-million instructions per CPU or three iterations prior to measurements.

3.3 Replacement Policy

Unless otherwise specified, we use bucketed-LRU replacement with n = 8 and $k = 5\%$ of cache size (419430 bytes) for both skewed-associative and ZCache designs. In Section 4.1, we establish that this bucketed-LRU scheme is superior to the NRU scheme described by Seznec. For the set-associative baselines, we use exact LRU replacement. We include results for a fully-associative LRU design to estimate the opportunity to reduce conflict misses in each workload.

3.4 Energy and Power Models

We estimate static and dynamic power for each L2 cache design using the leakage power and per-access energies reported in [17]. These per-access figures were obtained from McPAT [9] and CACTI 6.5 [11]. We reproduce these power/energy estimates in Table 3.

To generate our full system energy estimates, we first estimate the additional runtime due to L2 and main memory accesses. We assume a 10-cycle delay for L2 accesses and a 200-cycle delay for main memory accesses. Under these assumptions, we can calculate the total L2 energy (considering both static and dynamic power) using the values in Table 3. To assess full system power, we assume a fixed 2W per core, and calculate main memory energy assuming a single DIMM with a 2.78 W background power and 51 nJ per-access energy (derived from the main memory power model in [5]). We present the full system energy as the sum of the calculated core, L2, and main memory energies, neglecting other system components. We have selected these core and memory estimates to approximate the system assumed in [17]; that is, a large number of Atom-class cores with a comparably small on-chip cache and low memory footprint.

4. Results

We begin our evaluation by first demonstrating that ZCache's bucketed-LRU replacement is superior to Seznec's NRU scheme for skewed-associative caches. Then, we contrast this enhanced skewed-associative design with ZCache to evaluate its hashing function and multi-level replacement scheme with respect to performance and energy.

4.1 NRU versus Bucketed-LRU

Figure 2 contrasts the miss rate of three version of skew associativity and a conventional 16-way set-associative design, normalized to the miss rate of a 4-way set-associative cache. Each bar shows percent improvement in misses-per-1000-instructions (MPKI) over the 4-way cache. The left-most bar in each group shows the 16-way set-associative design. *Skew(NRU)* represents a skewed-associative design with Seznec's "enhanced NRU" replacement policy [21], which uses the full A_2 address segment. *Skew(Bucketed LRU)* adds the replacement policy proposed for ZCache. Finally we show *Skew(Bucketed LRU + 3-level)* which shows the skew algorithm with all of additional contributions of ZCache. Compared to the bucketed-LRU, 3-level design, on

Component	Configuration
Cores	16 In-order Cores @ 2.0 GHz
Architecture	UltraSPARC III ISA
L1 I & D Caches	64KB each with 64-byte blocks
L2 Cache	8MB, Non-inclusive, Shared, 4-way (unless otherwise specified), 10-cycle access latency
DRAM	1 DIMM, 200-cycle access latency

Table 1: *System Configuration for Simulations*

Benchmark	Configuration
apache	16K connections, fastCGI, 12.8 billion instr.
oracle	100 warehouses (10 GB), 16 clients, 1.4 GB SGA, 6.4 billion instr.
db2	100 warehouses (10 GB), 64 clients, 2 GB buffer pool, 6.4 billion instr.
zeus	16K connections, fastCGI, 12.8 billion instr.
blachscholes	native workload, 6.4 billion instr.
dedup	native workload, 6.4 billion instr.
facesim	native workload, 6.4 billion instr.
fluidanimate	native workload, 6.4 billion instr.
vips	native workload, 6.4 billion instr.
x264	native workload, 6.4 billion instr.
em3d	768K nodes, degree 2, span 5, 7 iterations
moldyn	19652 mo., boxsize 17, 2.56M iteractions, 9 iterations

Table 2: *Benchmark Configurations*

Cache Type	Serial lookups			Parallel lookups				
	Bank latency	Bank E/hit	Bank E/miss	Bank latency	Bank E/hit	Bank E/miss	L2 area	L2 leakage
SetAssoc 4-way	4.14 ns	0.61 nJ	1.26 nJ	2.91 ns	0.71 nJ	1.42 nJ	42.3 mm^2	535 mW
SetAssoc 8-way	4.41 ns	0.75 nJ	1.57 nJ	3.18 ns	0.99 nJ	1.88 nJ	45.1 mm^2	536 mW
SetAssoc 16-way	4.74 ns	0.88 nJ	1.87 nJ	3.51 ns	1.42 nJ	2.46 nJ	46.4 mm^2	561 mW
SetAssoc 32-way	5.05 ns	1.23 nJ	2.66 nJ	3.82 ns	2.34 nJ	3.82 nJ	51.9 mm^2	588 mW
ZCache 4/16	4.14 ns	0.62 nJ	2.28 nJ	2.91 ns	0.72 nJ	2.44 nJ	42.3 mm^2	535 mW
ZCache 4/52	4.14 ns	0.62 nJ	3.47 nJ	2.91 ns	0.72 nJ	3.63 nJ	42.3 mm^2	535 mW

Table 3: *Area, power, and latency for 8MB, 8-banked L2 caches with different organizations. Values reproduced from [17].*

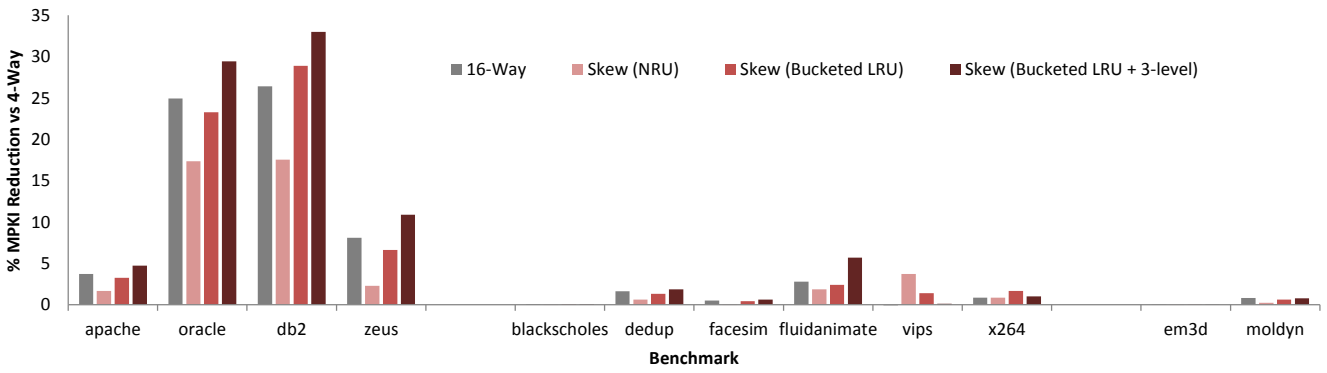

Figure 2: *Comparisons of misses-per-1000-instructions (MPKI) improvements over 4-way set associative caches for 16-way set associative, skewing with A_2 and NRU (NRU), bucketed LRU (Bucketed LRU), and bucketed LRU with 3-levels of replacement (Bucketed LRU + 3-level)*

average, 42% of the benefit is from the skewing algorithm with NRU, 20% comes from the addition of the bucketed-LRU algorithm, and the rest is from the 3-levels of replacement.

For workloads that are sensitive to replacement policy, the bucketed-LRU scheme substantially outperforms NRU. Moreover, in nearly all cases, the performance of the best skewed-associative design matches or comes close to the performance of a 16-way set-associative cache.

4.2 H_3 versus Skewing

Figure 3 contrasts MPKI across several set-associative, ZCache, and skewed-associative designs. All results are again reported as improvements over a 4-way set-associative cache with LRU replacement. Both skewed-associative and H_3-based designs are shown for 1-level (4 replacement candidates), 2-level (16), and 3-level (52) replacement schemes.

The figure demonstrates that there is a negligible difference between the H_3 hash and our skew function. As skew requires less hardware overhead and is conceptually simple, we see little reason to use the H_3 hash.

4.3 Performance versus Energy

Figure 3 also shows the gain in cache miss rate from ZCache's multi-level replacement scheme. Several of the workloads are insensitive to replacement policy, gaining no benefit from multi-level replacement. For many workloads, single-level replacement (with 4-way lookup) is sufficient to match the performance of 16-way associative caches. The commercial applications and fluidanimate do gain some benefit from two-level lookup, closing the gap between 16-way associative and fully-associative designs. Three-level lookup provides at best marginal gains for the commercial workloads, which (as we show below) do not justify its energy and complexity overheads. The diminishing returns in performance are a limitation of the replacement policy being unable to make effective use of the added associativity. If the additional associativity could be better utilized with an improved replacement policy, three-level lookup might provide more substantial gains.

The benefits of multi-level lookup come at a significant cost in average L2 power, as shown in Figure 4. In this figure, we show the percent increase in total L2 power relative to a 4-way set associative cache (higher is worse). However, most of this added cache power is mitigated by energy savings from improved performance when considering the full-system energy. Figure 5 shows that in a system with 16 2W cores and a 1-DIMM main memory, two-level replacement provides the best trade-off for most of the commercial workloads. Three-level replacement provides a marginal gain for oracle and fluidanimate, but the benefit is unlikely to warrant the additional implementation complexity. Three-level replacement results in a net energy loss for the remaining workloads when compared with fewer levels of replacement. Our results also demonstrate that multi-level lookup is not justified for scientific applications—these applications thrash the cache, invoking the multi-level replacement algorithm frequently (at a high power overhead), but gain no benefit from the additional replacement candidates.

It is important to note that the L2 cache-to-core power ratio assumed in this analysis is atypical for a server-class system. Under the set of assumptions discussed here, the L2 cache is small (only 8 MB for a 32nm process assumption) and contributes only a couple of percent of the total system power (the last-level cache has been reported as roughly 20% of chip-level power for Niagara 2 [12] and Nehalem [1]). Hence, our L2 cache power overheads are often dwarfed by core power and main memory energy savings from reduced miss rates in our analysis. We have selected these assumptions to provide consistency with the results presented in [17]. However, we have also provided MPKI reductions for a 32 MB L2 shown in Fig. 6. This graphs shows the 8 MB results in the left set of bars and 32 MB results in the right set of bars. Not only does the 32 MB cache provide a significant improvement over the smaller design, but the advantage of multiple levels of replacement shrinks, most notably between one-level and two-level. On balance, we conclude that two-level replacement is likely justified for server-class systems, but three-level replacement will result in energy losses.

5. Related Work

Hash Function Analysis. Previous work has examined the effectiveness of various hash functions in skewed-associative designs [23]. In particular, this work has examined XOR-based hashes and their matrix representations to gain insight into conflicts via null space and column space analysis. Whereas the prior study focuses on theoretical analysis of the functions, we compare the skew function and H_3 hashes empirically.

Cuckoo Directories. Another recent proposal similar in spirit to ZCache is Cuckoo Directories [6]. A Cuckoo Directory uses skewed-associative lookup, and upon a miss, considers allocations that involve multiple displacements. ZCache limits the number of levels searched during an eviction operation. In contrast, the Cuckoo Directory performs a depth-first search and does not limit search by levels. Instead it continues to iterate until a valid insertion location is found, and data is then shifted as needed.

V-way Cache. The V-way cache [14] also seeks to increase effective associativity by using a variable number of ways in each set. It exploits the observation that in a set-associative cache, the working set that maps to one cache set may be larger than that of another. By redistributing the data, V-way caches improve utilization.

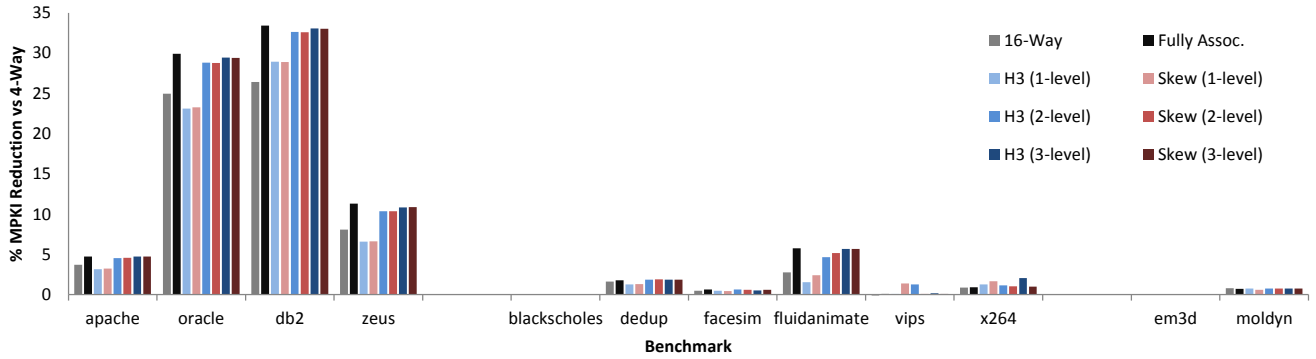

Figure 3: *Comparisons of misses-per-1000-instructions (MPKI) improvements over 4-way set associative caches for 16-way set associative and fully associative using LRU and ZCache and Skewing with 1, 2, and 3 levels of replacement*

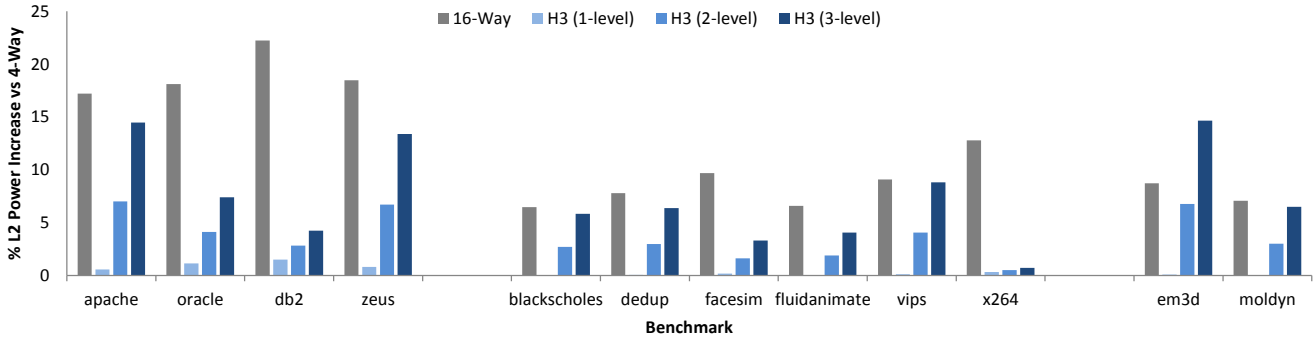

Figure 4: *Power increase over 4-way set associative design generated from power values in Table 3*

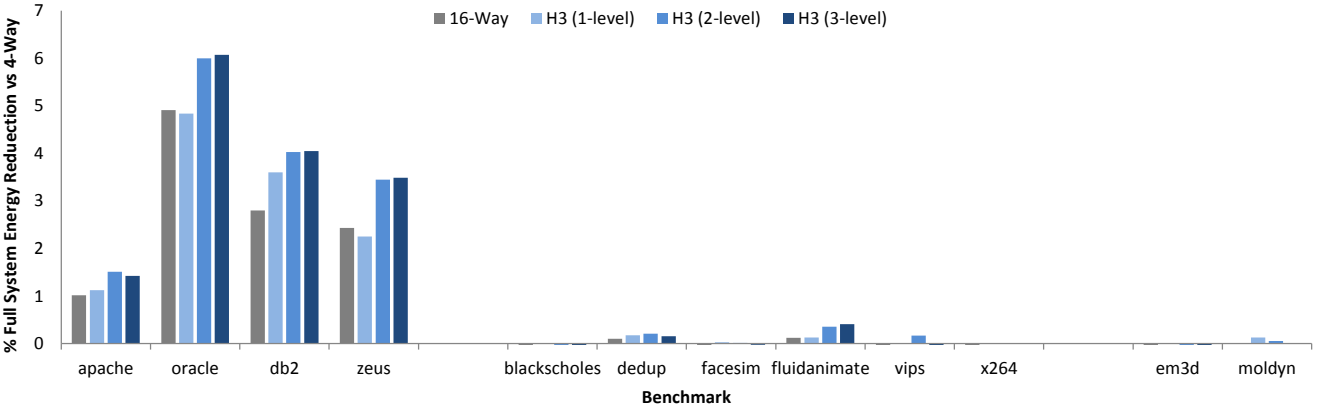

Figure 5: *Full system energy reduction over 4-way set associative design assuming 2W per core*

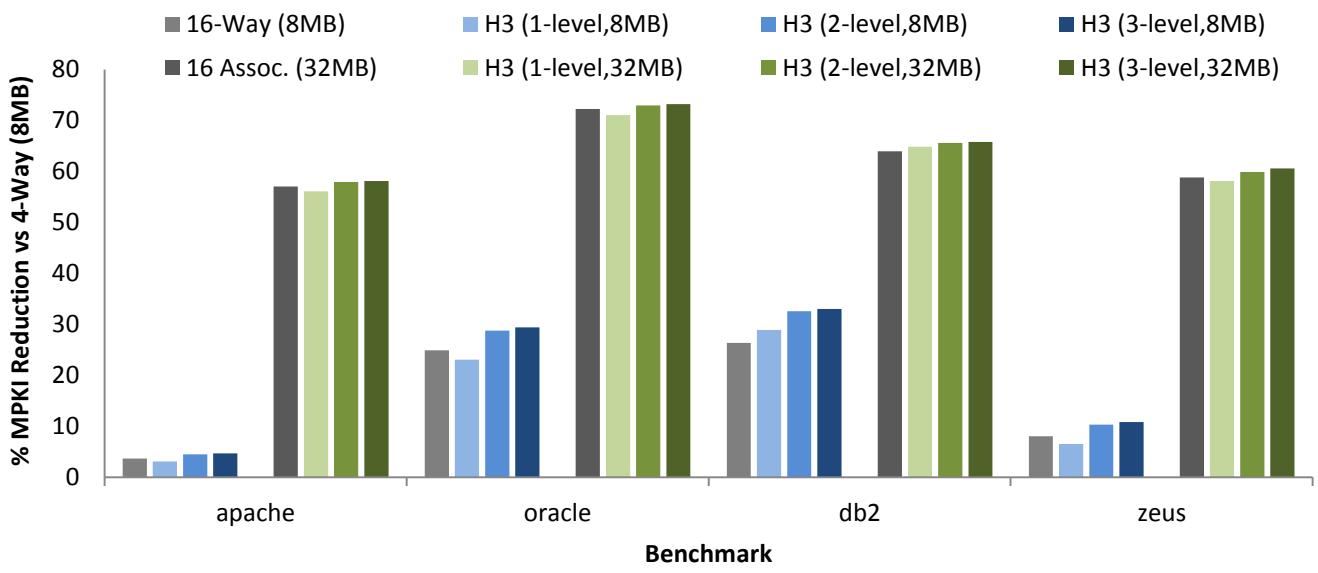

Figure 6: *Comparisons of misses-per-1000-instructions (MPKI) improvements over 4-way set associative caches for 16-way set associative using LRU and ZCache and Skewing with 1,2, and 3 levels of replacement. For each benchmark, the left set of graphs has 8 MB L2 caches and the right set of graphs has 32 MB L2 caches*

RRIP. RRIP [8] describes a replacement scheme that is closely-related to Seznec's pseudo-LRU. RRIP is motivated by the observations that effective replacement algorithms, such as LRU, are expensive and difficult to implement. Additionally, LRU keeps no information about how often data is accessed, and is oblivious to the fact that data recently put into the cache may not be needed again. RRIP uses a simple mechanism that starts data at a low priority, and upon future accesses, raises its priority. Victim blocks are selected from the lowest priority group; if no blocks have minimum priority, then the priority of all blocks is reduced and the search is repeated. By using two bits to keep track of priorities, RRIP can achieve excellent performance in set-associative caches.

6. Conclusions

In this paper, we have deconstructed ZCache, a new design for caches that uses both hashed indexes and multi-level replacement to achieve greater associativity and performance without increasing the number of ways. We have shown that previously proposed skewed-associative caches use a simpler indexing algorithm that is able to achieve comparable performance to ZCache's H_3 hashes with lower hardware overheads. We have also shown that, whereas a two-level replacement scheme might benefit commercial workloads, its energy overheads are not justified for scientific applications. Furthermore, we have shown that three-level replacement pays an even larger power cost and while it may be beneficial overall in systems with small caches, server-class designs are more likely to favor two-levels of replacement. Finally, we have shown that applying a simple modification to the skewing function and adopting ZCache's bucketed LRU replacement policy significantly improves skewed-associative cache performance, matching that of ZCache.

Acknowledgements

The authors would like to thank Daniel Sanchez and Christos Kozyrakis for extensive comments on earlier drafts of this work. This work was supported in part by grants from ARM and NSF grant CCF-0815457.

References

[1] "Intel Xeon Processor 5600 Series. Datasheet, Vol. 1," 2010.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," pp. 72–81, 2008.

[3] F. Bodin and A. Seznec, "Skewed associativity enhances performance predictability," *ISCA '95: 22nd Annual International Symposium on Computer Architecture*, pp. 265–274, 1995.

[4] J. L. Carter and M. N. Wegman, "Universal classes of hash functions (extended abstract)," *STOC '77: Ninth Annual ACM Symposium on Theory of Computing*, pp. 106–112, 1977.

[5] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," *ASPLOS '11: 16th international conference on architectural support for programming languages and operating systems*, vol. 46, pp. 225–238, 2011.

[6] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," *HPCA '11: International Symposium on High-Performance Computer Architecture*, 2011.

[7] M. D. Hill and A. J. Smith, "Evaluating associativity in cpu caches," *IEEE Trans. Comput.*, vol. 38, pp. 1612–1630, 1989.

[8] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ISCA '10: 37th annual international symposium on Computer architecture*, pp. 60–71, 2010.

[9] 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," *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, 2009.

[10] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," pp. 367–378, 2008.

[11] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," *40th Annual International Symposium on Microarchitecture*, pp. 3–14, 2007.

[12] U. Nawathe, M. Hassan, K. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, "Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 6–20, 2008.

[13] R. Pagh and F. F. Rodler, "Cuckoo hashing," *ESA '01: 9th Annual European Symposium on Algorithms*, 2001.

[14] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The v-way cache: Demand based associativity via global replacement," *ISCA '05: 32nd annual international symposium on Computer Architecture*, pp. 544–555, 2005.

[15] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. Comput.*, vol. 46, pp. 1378–1381, 1997.

[16] P. Ranganathan, S. Adve, and N. P. Jouppi, "Reconfigurable caches and their application to media processing," pp. 214–224, 2000.

[17] D. Sanchez and C. Kozyrakis, "The zcache: Decoupling ways and associativity," *43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 187–198, 2010.

[18] D. Sanchez and C. Kozyrakis, "Vantage: scalable and efficient fine-grain cache partitioning," *ISCA '11: 38th annual international symposium on Computer architecture*, 2011.

[19] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam, "Implementing signatures for transactional memory," *40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 123–133, 2007.

[20] A. Seznec, "A case for two-way skewed-associative caches," *ISCA '93: 20th Annual International Symposium on Computer Architecture*, pp. 169–178, 1993.

[21] A. Seznec, "A new case for skewed-associativity," *Internal Publication No 1114, IRISA-INRIA*, 1997.

[22] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," *The Journal of Supercomputing*, vol. 28, pp. 7–26, 2004.

[23] H. Vandierendonck and K. De Bosschere, "Xor-based hash functions," *IEEE Trans. Comput.*, vol. 54, pp. 800–812, July 2005.

[24] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, pp. 18–31, 2006.