Comparing Modern Microtagging with Other Pseudo-Associative Schemes

Nathaniel Young
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA
nateyoung@berkeley.edu

Tsang-Yung Wu
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA
tywu13@berkeley.edu

***Abstract*—In order to reduce power and bank conflicts in the L1 cache, AMD CPUs use microtagging. This technique dramatically reduces the pressure on the cache, but introduces more conflicts and the potential for aliasing. In this report, we provide empirical miss rate results comparing microtagging to lower-associativity and skew-associative caches, as well as a qualitative discussion of numerous schemes with similar tradeoffs.**

***Index Terms*—cache, AMD microtagged cache, low power, pseudo-associative**

I. Introduction

AMD CPUs from the 15h, 17h, and 19h families use microtagging in what AMD calls a "linear address microtag/way-predictor" [1]. These microtags are stored in a 512-byte microtag array, which is accessed before the tags and data are accessed in the cache itself; this allows the cache to access only the one cache way which has a matching microtag, or to declare an "early miss" if there is no microtag match. Since only one way out of 8 needs to be accessed, this both reduces cache power and increases cache throughput as fewer discrete SRAM banks are touched by any one access. In effect, microtagging is an attempt to retain a large amount of effective associativity while only accessing one cache entry for any access.

However, there are numerous cache organization strategies which have been described in research which hope to increase effective associativity for the same total work done in accessing the cache. Therefore, a natural question is whether microtagging is actually effective compared to these other "pseudo-associative" strategies. We provide a qualitative analysis of several such strategies (way prediction, column-associativity, skewed-associativity, victim cache) and use the gem5 simulator [4] to compare the miss rate of a microtagged cache with that of a standard set-associative cache and that of one of these other schemes (skewed-associativity).

II. Description of Schemes

We will describe several pseudo-associative caching schemes, including the microtagging scheme used by AMD as well as several alternatives.

A. Set-Associativity

For completeness, we will describe the attributes of standard set-associative caches which are relevant to our discussion.

Set-associative caches are divided into many 'sets,' each with a constant number of entries (the 'associativity'). For a given access, the lower bits of the memory address define an 'index' which identifies the set in which the data should reside; it may reside anywhere in this set, but nowhere else. In this way, several pieces of data with different addresses but the same index can reside in the cache at the same time.

The downside of a set-associative cache is that it requires many entries to be accessed in the cache for every request, since there are numerous locations in which the data could reside. This requires a lot of power and makes it likely that independent requests will require accessing the same SRAM banks. A tradeoff therefore develops between conflict misses (which decrease with increasing associativity) and cache power and utilization (both of which increase with increasing associativity). One way to improve this tradeoff is with pseudo-associative schemes such as those described below; another way is to build a specialized content-addressable memory, or CAM, which retrieves only the entry in the cache for which the tag matches, while not being quite as slow as serializing the entire data access behind the tag check [11]. CAMs, however, require difficult custom design to implement effectively.

B. Microtagged

In order to reduce cache power (and the likelihood of independent accesses requiring access to the same SRAM bank), caches in some AMD architectures use microtags. Figure 1 shows the operation of this cache, as near as we can decipher from the AMD software optimization guide [1] (which is quite thorough but does not provide all details).

In this scheme, an 8-bit hash value is computed from the address of every load and store. On a cache fill, the index and microtag are computed, and all 8 microtags currently stored for that index are checked. If there is no match, the data is not present and an entry is chosen to be replaced by the new data as usual; the microtag is written to the microtag array in the corresponding way. If there is a match, that way alone can be used for the new data. When the cache is accessed for a load, the same checking process takes place. If no microtags match, the cache declares an 'early miss.' Otherwise, the one cache way with the matching microtag is accessed.

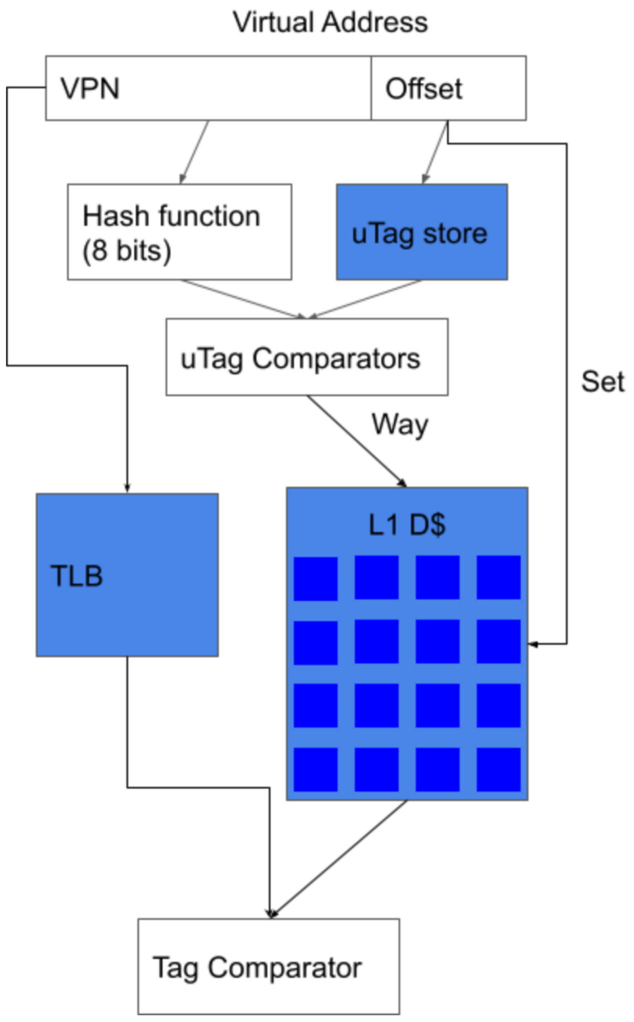

Fig. 1. A diagram of a cache access in the AMD microtagged scheme

AMD calls this ‘microtagged way prediction,’ but this is something of a misnomer, as it does not involve ‘predicting’ a way of the cache – at least, there is no possibility of ‘misprediction.’ Either the data resides in the way indicated by the microtag array, or it does not reside in the cache at all (except in cases of aliasing, which we will discuss later).

One might imagine a scheme which does actually use microtagging to implement way prediction, by relaxing the requirement that only one way of the set has a given microtag, but we will not discuss this approach in this report.

C. *Way Prediction*

A way-predicting cache avoids the cost of parallel access to many cache ways by simply serializing these accesses [5]. When an access happens, a single cache way is checked to see if the data is present. If it is not, a second cache way is checked on the next cycle, and so on. A ‘way predictor’ is included before the cache which indicates in what order the ways should be accessed for each set (usually by recording the most-recently-used way).

D. *Skewed-Associativity*

Conflict misses occur in caches when whole cache sets fill up with useful data (that is, when numerous addresses all have the same index bits); skewed-associative caches mitigate this issue by separating the ways of the cache, and accessing each with a different hash function [9]. In this way, in order for several addresses to conflict in the cache, they would have to collide with each other under several different hash functions, instead of just having the same index bits.

E. *Column-Associativity*

Column-associative caches are similar to skewed-associative caches in that the location of data in the cache is determined by a hash of the address, but the ways of the cache are not separated [2]. Instead, all hash functions produce locations in the same unified cache memory. We will assume for our purposes that the accesses corresponding to these hash functions happen serially so as to not require two independent accesses of the same memory in one cycle (although with banked structures, this requirement could be relaxed).

F. *Victim Cache*

A victim cache is a small fully-associative cache which is filled with recently-evicted entries from the main cache [6]. If the main cache has high conflict misses in only a few sets, the evicted entries are likely to be accessible in the victim cache.

III. Discussion of Schemes

All the schemes described above attempt to improve the tradeoff described in the introduction, between the effective associativity of the cache and the cost of accessing it. In so doing, however, they incur other costs. We will discuss these costs for the various schemes, and provide comparisons between them.

A. *Microtagging*

The microtagging scheme described by AMD has two major drawbacks: firstly, because only one entry with a given microtag can reside in any set, it introduces extra conflicts, and secondly, the microtag array itself increases area, access latency, and power. In particular, since the AMD cache has 512 entries (32KB at 64B cache lines), and the microtags are 8 bits long, the microtag array must be 512 bytes [1]. In addition, the cache supports two loads and one store operation every cycle (assuming no bank conflicts), meaning that the microtag array needs three independent ports. This means the array is almost certainly implemented using latches or flip-flops, rather than SRAM. This will lead to a fairly large area cost for implementing microtagging, and the array likely takes a cycle or two to access. The software optimization guide quotes a 4-5 cycle latency through the cache for integer memory operations [1].

In our quantitative analysis below, we explore the effect of microtagging on the miss rate of the cache, but we have not investigated the effect of the increased latency. For a cache designed with a latency budget in mind, it might be the case that the microtag array becomes the limiting factor in the size of the cache.

B. Way Prediction

The one major drawback of way prediction is the obvious: sometimes the predicted way will be incorrect, thereby increasing the apparent latency of many memory operations. The way predictor itself may also increase latency somewhat, although this effect is not likely to be as dramatic as for the microtag array. Suppose for instance that the exact order in which the ways should be accessed were stored in an array for an 8-way cache with 512 entries; this would require 3 bits (for a number from 0 to 7) for each entry, or 192 bytes.

C. Skewed-Associativity and Column-Associativity

We combine our discussion of skewed-associative and column-associative caches because these schemes are very similar. Both are "index using a hash function" approaches, and one might view a serial-accesses column-associative cache as just the way-predicting version of a skewed-associative cache.

Both of these approaches add a small amount of latency and complexity in order to calculate the hash function used to access the cache, but have no other major drawbacks.

D. Victim Cache

The main drawback of a victim cache is, again, obvious: a cache with high associativity must be implemented in addition to the main cache and accessed, if not on every memory operation, then at least on every cache miss.

Note that accessing the victim cache every cycle could easily be just as bad for power usage as having a highly-associative main cache, since in both cases the high power utilization comes from the fact that a large amount of data is read from memories at once.

E. Comparison

Direct comparison among these schemes is difficult, since most are attempts to decrease conflict misses while paying a small extra cost, whereas microtagging is an attempt to decrease cost while slightly increasing conflict misses. Additionally, the actual miss rate of these various schemes would be difficult to estimate – we have done simulation work to find the miss rate of microtagged and skewed-associative schemes. So, Table I summarizes the discussion above; bear in mind that this discussion is of course imprecise, and that the important metric of a scheme is the relationship between the measures recorded, since their actual values could change with the baseline features of the cache in which they are implemented.

TABLE I
CHARACTERISTICS OF PSEUDO-ASSOCIATIVE CACHING SCHEMES

*As compared to a set-associative cache of the same associativity; - - means much lower, - means lower, ~ means negligible difference, + means higher, + + means much higher

Scheme	Miss Rate	Latency	Area	Power
Microtagged	+	+ +	+	- -
Way-Predicting	~	+ +	+	-
Skewed-Associative	-	+	~	~
Column-Associative	-	+ +	~	-
Victim Cache	-	+	+	+

F. Hashing

It should be noted that three of the schemes discussed here (microtagging, skewed-associativity, and column-associativity) depend on hash functions, and therefore their performance depends heavily on which hash function is chosen. In our simulations of a microtagged cache, we used the hash function used by AMD in the Zen, Zen+, and Zen 2 architectures as reverse-engineered by Lipp et al. [7]. In our simulations of skewed-associative caches, we used the much simpler default hash function provided by gem5, based closely on the hash function described in the original paper of Seznec and Bodin [9]. This is perhaps not fair, since the quality of these hash functions could be very different. Indeed, in a short experiment, not described in this report, we found that a slight simplifying change to the hash function used by the microtagged cache had a fairly large effect on miss rate. Regardless, more research needs to be done to find good hash functions for all schemes which use them, and this research could have a large effect on the miss rates of these schemes.

G. Combining and Refining Schemes

A natural question about the schemes described here, since they have varying strengths and weaknesses, is whether they could be combined in a useful way. This is not always easy. For instance, combining microtagging with skewed-associativity naively would break the guarantee that only one element of a set has any particular microtag, since indices do not correlate between ways (and there is no real notion of "set" in a skewed-associative cache). However, one can see a column-associative cache as almost the same thing as a skewed-associative cache with way prediction; the original column-associativity paper describes a technique for repositioning data in the cache so that as many accesses as possible will hit under the first hash function instead of the second [2].

Here we should also mention the ZCache, which was proposed by Sanchez and Kozyrakis and combines hash functions for indexing (as in a skewed-associative cache) with a sophisticated replacement policy designed to move conflicting blocks on a miss and increase the number of candidates for eviction [8].

H. Aliasing and Page Coloring

The reader may have noticed that one consideration is absent here: that of virtual memory aliasing. Any cache which uses bits of the address beyond the page offset (bits 11 through 0 for the common 4KB page size) in order to determine where the data resides in the cache runs the risk of placing the same data into the cache twice at two different places, if two locations in virtual memory are mapped to the same location in physical memory. The microtagging approach used by AMD uses bits from the virtual page number in the address in the hash function to compute the microtag, and a fair amount of space in the software optimization guide is dedicated to discussing this problem and the anti-aliasing scheme implemented to mitigate it. We have omitted any discussion about aliasing in this report for two reasons: firstly, aliasing is rare, making anti-aliasing schemes not performance-critical, and secondly, the need to consider aliasing is not unique to pseudo-associative caches. The ARM Cortex-A76, for instance, has a 64KB 4-way set-associative L1 data cache (thus 16KB ways) despite supporting a 4KB page size. This cache is described as being virtually indexed, but behaving as though it were physically indexed [3].

Another issue which arises in similar circumstances is page coloring. With cache index ranges larger than the page size, operating systems must be careful in where in the pages are allocated so as not to introduce bias in which parts of the cache are actually used. Again, we ignore this because it is not unique to pseudo-associative schemes but is necessary for large physically-indexed caches, and because hash functions, distributing traffic across the cache indices, may even lessen the burden on page coloring.

IV. Quantitative Analysis

A. Methodology

We compare the microtagged cache with set-associative and skewed-associative caches using the gem5 simulator [4]. We simulate using the ARM ISA since it the best-supported RISC architecture in gem5. gem5 supports full-system and system emulation simulation modes. We would like to know the miss rate and conflicts incurred when using different schemes. We choose system emulation mode to just emulate the system call to make simulation faster, while still preserve most of the traces in programs that we could get meaningful data. gem5 natively supports set-associativity and skewed-associativity; we implemented AMD microtagging scheme in gem5 ourselves. The set-associative cache we use as a baseline has 32KB and 8-way set associative. For AMD microtagging caches, we use the same configuration. For the skewed-associative cache, we simulate with 32KB and 8-, 4-, and 2-way configuration. We also the simulate normal set-associative cache with 32KB and 4-way. The benchmark we use is SPEC CPU2017 [10]. There are a total of 43 benchmarks in it. However, we remove 12 benchmarks which didn't run, and 2 outliers. Also, for time's sake, we simulate on the test inputs. This gives us a rough idea of how they perform differently in a short period of simulation time.

B. Results

- The result for SPEC CPU2017 intspeed benchmark suite is shown in Figure 2. *620.omnetpp* and *648.exchange2* are removed since they cannot run on gem5 using ARM ISA. For AMD microtagging scheme, it has a little higher miss rate in most of the benchmarks except for *641.leela*. Table II shows a 0.66% more miss rate than the normal set-associative cache calculated by the geometric mean of the intspeed benchmarks that can run. The skewed-associative cache is actually quite a bit better than the rest of the schemes, with 3.05% less miss rate than the normal set-associative cache. Moreover, the 4-way skewed-associative cache is over 5% less miss rate than the 4-way set-associative cache in terms of the geometric mean of miss rate ratios.

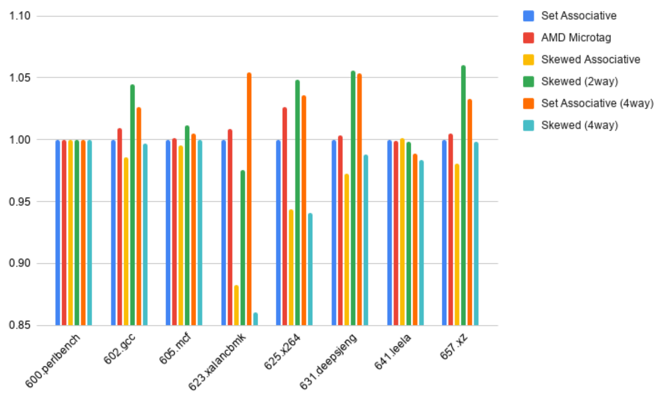

Fig. 2. Miss rate ratio to 8-way set associative cache for intspeed benchmark suite.

- The result for SPEC CPU2017 intrate benchmark suite is shown in Figure 3. *520.omnetpp* and *548.exchange2* are also removed since they cannot run on gem5 using ARM ISA. The miss rate ratios are similar to intspeed benchmark suite. For AMD microtagging, Table II shows a 0.85% more miss rate than that of the normal set-associative cache calculated by the geometric mean of the intrate benchmarks that can run, while the skewed-associative cache is almost 4% less miss rate than that of the normal set-associative cache.

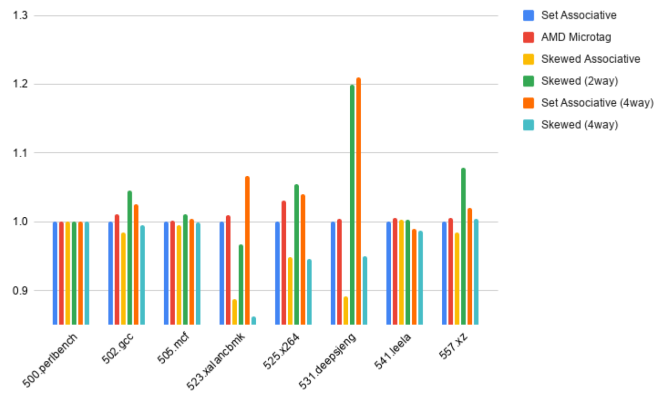

Fig. 3. Miss rate ratio to 8-way set-associative cache for intrate benchmark suite.

- The result for SPEC CPU2017 fpspeed benchmark suite is shown in Figure 4. We remove *603.bwaves*, *621.wrf*, *627.cam4*, and *654.roms* since those cannot run on gem5. Though *607.cactuBSSN* can be executed, the miss rate difference of that benchmark is quite large between the skewed cache and the normal set-associative cache. We remove it from the calculation of the geometric mean of miss rate ratios to prevent huge bias incurred from that benchmark. According to Table II, the 4-way set-associative cache has only 0.01% more miss rate, better than that of the skewed-associativ and AMD microtagging cache.

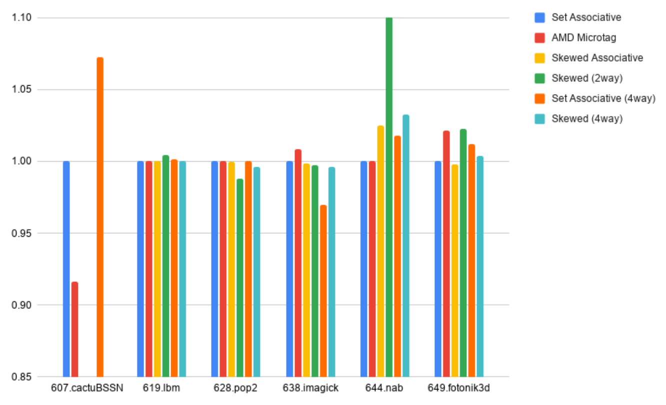

Fig. 4. Miss rate ratio to 8-way set-associative cache for fpspeed benchmark suite.

- The result for SPEC CPU2017 fprate benchmark suite is shown in Figure 5. We remove *503.bwaves*, *521.wrf*, *526.blender*, and *554.roms* since those cannot run on gem5. Also, *507.cactuBSSN* is removed from the calculation of the geometric mean of miss rate ratios due to the huge difference miss rate between different schemes. However, unlike fpspeed benchmark suite, 8-way skewed-associative cache has 2% less miss rate than that of the set-associative cache. The AMD microtagging cache has its worst miss rate ratio, 1.05% more miss rate than that of the set-associative cache, compared with the results in other benchmark suites.

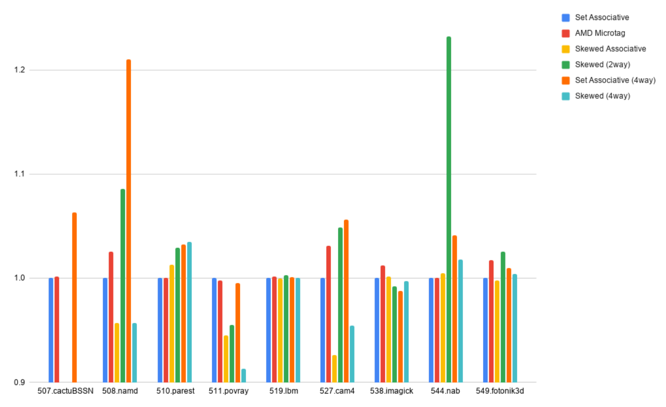

Fig. 5. Miss rate ratio to 8-way set-associative cache for fprate benchmark suite.

The overall miss rate ratio results are shown in Table II. The AMD microtagging cache has only 0.81% more miss rate than that of the set-associative cache. This means that AMD microtagging cache incurs less than 1% conflict misses compared with the normal set-associative cache. Also, the use of the skewed-associative cache can reduce miss rate. The skewed-associative cache has 2.42% less miss rate than that of the set-associative cache.

TABLE II
GEOMETRIC MEAN OF MISS RATE RATIO TO 8-WAY SET-ASSOCIATIVE CACHE

Schemes	**intspeed**	**intrate**	**fpspeed**	**fprate**	**overall**
Set-Associative	-	-	-	-	-
AMD Microtag	+0.66%	+0.85%	+0.61%	+1.05%	**+0.81%**
Skewed	-3.05%	-3.94%	+0.42%	-2.00%	**-2.42%**
Skewed (2way)	+2.38%	+4.29%	+2.17%	+4.36%	+3.41%
Set-Associative (4way)	+2.44%	+4.26%	+0.01%	+3.96%	+2.93%
Skewed (4way)	-3.02%	-3.32%	+0.57%	-1.59%	-2.10%

V. Future Work

In our simulation results, AMD's microtagging scheme can offer lower power and larger bandwidth at the cost of a little more conflict misses. This offers a great opportunity for cache designs. However, lots more work still needs to be done to have a good understanding of this space. Beside microtagging, we see that skewed-associative cache can provide way better miss rate compared with traditional set-associative caches. More work is needed even to strengthen our results: simulation on reference inputs instead of on the test inputs of SPEC CPU2017, simulation of other benchmarks, testing other hash functions, testing other cache sizes and baseline associativities, etc. However, even with good data on the miss rate of these specific schemes, more work will need to be done to really characterize and understand the space of pseudo-associative caches. For instance, the exact impact of the microtagging approach on latency, power, and other cache properties needs to be found in order to evaluate it properly. Exploration of other schemes entirely, including those mentioned in sections II and III and combined schemes, is also needed to provide more insights into this space. Implementing those schemes in gem5 or using other simulators will help us get the results.

VI. Conclusion

Microtagging, as used by AMD, appears to work very well: it incurs only a few extra cache misses, and it dramatically reduces the number of bits which need to be accessed in the cache. This, coupled with the fact that AMD was able to implement it in a practical way, gives us hope that other pseudo-associative schemes could be used to similar effect – improving the tradeoff between miss rate and cache resources – in future designs.

References

[1] Advanced Micro Devices (2017). Software Optimization Guide for AMD Family 17h Processors.

[2] Agarwal, A., Pudar, S. D. (1993, May). Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In Proceedings of the 20th annual international symposium on Computer architecture (pp. 179-190).

[3] Arm Limited (2020). Arm Cortex-A76 Core Technical Reference Manual, Revision: r4p1

[4] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., ... Wood, D. A. (2011). The gem5 simulator. ACM SIGARCH computer architecture news, 39(2), 1-7.

[5] Inoue, K., Ishihara, T., Murakami, K. (1999, August). Way-predicting set-associative cache for high performance and low energy consumption. In Proceedings of the 1999 international symposium on Low power electronics and design (pp. 273-275).

[6] Jouppi, N. P. (1990). Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. ACM SIGARCH Computer Architecture News, 18(2SI), 364-373.

[7] Lipp, M., Hažić, V., Schwarz, M., Perais, A., Maurice, C., Gruss, D. (2020, October). Take A Way: Exploring the Security Implications of AMD's Cache Way Predictors. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (pp. 813-825).

[8] Sanchez, D., Kozyrakis, C. (2010, December). The ZCache: Decoupling ways and associativity. In 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 187-198). IEEE.

[9] Seznec, A., Bodin, F. (1993, June). Skewed-associative caches. In International Conference on Parallel Architectures and Languages Europe (pp. 305-316). Springer, Berlin, Heidelberg.

[10] SPEC CPU2017 home page: www.spec.org/cpu2017

[11] Zhang, M., Asanovic, K. (2000, December). Highly-associative caches for low-power processors. In Kool Chips Workshop, MICRO (Vol. 33).