A Novel Cache Architecture with Enhanced Performance and Security

Zhenghong Wang and Ruby B. Lee
Princeton Architecture Laboratory for Multimedia and Security (PALMS)
Department of Electrical Engineering, Princeton University
Princeton, NJ 08544, USA
{zhenghon, rblee}@princeton.edu

***Abstract*—Caches ideally should have low miss rates and short access times, and should be power efficient at the same time. Such design goals are often contradictory in practice. Recent findings on efficient attacks based on information leakage in caches have also brought the security issue up front. Design for security introduces even more restrictions and typically leads to significant performance degradation. This paper presents a novel cache architecture that can simultaneously achieve the above goals. Specifically, cache miss rates are reduced with dynamic remapping and longer cache indices, access-time overhead overcome with astute low-level circuit design, and information leakage thwarted by a security-aware cache replacement algorithm together with the performance enhancing mechanisms. We present both theoretical analysis and experimental results, using the SPEC2000 suite to evaluate the cache miss behavior, and CACTI and HSPICE to validate the circuit design. Our results show that the proposed cache architecture has low miss rates comparable to a highly associative cache and short access times and power efficiency close to that of a direct-mapped cache. At the same time it can thwart cache-based software side-channel attacks, providing both legacy and security-enhanced software a much higher degree of security. Additional benefits that the proposed cache architecture can bring, like fault tolerance and hot-spot mitigation, are also discussed briefly.**

Keywords- cache; computer architecture; security; side channel attacks; performance

I. INTRODUCTION

Cache memory is an essential processor component for overcoming the processor-memory speed gap. Ideally, caches should have both short access times and low miss rates to minimize average memory access delay. Unfortunately, caches which achieve the best access times, like direct-mapped (DM) caches, suffer from high miss-rates. Fully associative (FA) or set-associative (SA) caches achieve the best miss-rates, but at the cost of increased access times and power consumption. Power efficiency is also a critical issue in cache design. Lower power dissipation means longer battery life for mobile devices. Higher power consumption causes heating and reliability problems, which have become a limiting issue for achieving performance. Increasing faults and hot-spots are also concerns – especially in the deep-submicron era. Due to the shrinking of technology feature sizes, process variations increase the number of faulty devices with excessive delay or leakage power. Also, current densities become higher, heating up the chip and causing hot spots more easily. Both impact chip yields and device lifetimes.

Another new and important aspect for cache design is security. Recent software cache-based side-channel attacks show that caches are highly vulnerable to leakage of critical information such as cryptographic keys. They rely only on the timing difference between cache hits and misses, and therefore are effective on all caches, impacting a wide range of platforms and users. Since security solutions often lead to very restrictive design, they typically result in severe performance degradation. In this paper, we show that security, performance, as well as power efficiency, can all be achieved together. Our proposed cache architecture can also provide additional benefits such as fault tolerance and hot-spot mitigation. Our main contributions include:

- A novel cache architecture that can enhance security, performance and power efficiency, simultaneously.
- New insights on achieving low conflict misses without increasing set-associativity or cache capacity, based on the analysis of the miss behavior of our proposed architecture.
- Detailed low-level circuit design showing that our proposed architecture can be implemented with short access times and power efficiency.
- Identifying, for the first time, that conflict misses can be largely independent of the cache capacity – a property of our proposed architecture and the basis of its many benefits.

Section II gives a brief overview of the information leakage problem in caches. It provides necessary background information on this new security problem and points out its impact on cache design. In section III, we present high-level organization and architectural features of our proposed cache architecture. We also discuss the implementation issues and propose a new low-level circuit design. In section IV, we analyze and evaluate our new cache architecture in terms of cache miss rates, access times, power efficiency and security. In section V, we discuss additional benefits including fault tolerance, hot-spot mitigation and further optimization for low power. We review related past work in section VI and conclude in section VII.

This work was supported in part by NSF Cybertrust and DARPA under grants CNS-0430487 and CNS-0752961.

978-1-4244-2837-3/08/$25.00 ©2008 IEEE

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

II. The Impact of Security on Cache Design

A. Information Leakage in Caches

Recent attacks [1-6] have shown that, in spite of software protections such as address space isolation or secure Virtual Machines, hardware caches in processors introduce interference between programs and users. For example, one process can evict cache lines of other processes, causing them to miss in cache accesses. Such interference happen all the time and seemed harmless to most people in the past. However, as demonstrated by the recent cache-based side channel attacks, critical information (e.g., cryptographic keys) can easily be leaked out due to such common cache behavior. In contrast to traditional cryptanalysis, these cache-based attacks allow the recovery of the *full* secret cryptographic key and require much less time and computation power. Furthermore, these attacks can succeed on almost all processors with caches, since they rely only on hits and misses that occur in all caches. Such attacks are also very easy to launch: a remote computer user can become an attacker without the need for special equipment.

B. Impact on Cache Design

Both software and hardware techniques were proposed to mitigate the information leakage problem in caches. Software techniques mostly involve rewriting the code to prevent known attacks from succeeding. One software solution is to avoid using memory accessing operations [3] (e.g., replacing AES table lookups with arithmetic and logical operations). The performance overhead however can be very high and the method is not generally applicable. Another software countermeasure preloads objects into the cache before any use of them so that all subsequent accesses hit in cache, thus leaking no information [1,3]. This approach however is not really secure since the preloaded objects could be evicted by other memory references at a later time, which indeed often occurs. Researchers also proposed to use alternative tables [3,7], table permutation [7] and algorithmic masking [3] to mitigate cache-based attacks. Such methods however normally lead to significant performance degradation, e.g., 2~4x slow down in the case of AES [7]. In general, we observe that software countermeasures normally incur significant performance degradation, and are often not secure enough due to the behavior of the underlying hardware cache. Software methods alone are not sufficient to provide secure yet high performance mitigation of cache-based information leakage.

Hardware methods were also proposed. Cache partitioning (Partitioned cache[8]) and cache line locking (PLcache[9]) prevent undesirable cache evictions if the objects are put into a private partition or locked in cache, respectively, thus helping to achieve constant execution time. RPcache [9] used a randomization-based approach, allowing interference but randomizes it so that it carries no information. The drawback of cache partitioning and cache line locking is cache under-utilization. Cache lines that are locked or belong to a private partition can not be used by other processes even when they are unused. The randomization-based approach can avoid cache underutilization.

In summary, the information leakage problem in caches introduces a new challenge in cache design. In addition to performance, power efficiency, reliability, etc., cache designers have to also take security into account, which typically introduces even more restrictions in cache design and compromises other design goals.

III. The Proposed Cache Architecture

In this paper, we show that a single cache architecture can provide performance, security, power efficiency as well as many other benefits simultaneously. The core idea of our new cache architecture is to adopt the *direct-mapped* architecture to inherit its fast cache access time and high power efficiency, and extend this with *dynamic memory-to-cache remapping* and a *longer cache index*, which enable it to achieve lower miss rates and also improved security. With a new *security-aware cache replacement algorithm* (SecRAND), our proposed cache architecture can achieve the same degree of security as RPcache[9]. The performance-enabling features also allow cache partitioning and locking mechanisms to be implemented efficiently without incurring the performance problems as in traditional caches.

A. Dynamic Re-mapping and Logical Direct Mapping

The proposed cache implements *dynamic memory-to-cache remapping*, meaning that a memory block can be mapped to any desired cache line at run time. Logically, this can be achieved by using a level of indirection. The index bits of the address are first used to lookup a ReMapping Table (RMT), which returns the index of the real cache line that the address is mapped to. By changing the contents of an RMT entry, an address can be remapped to an arbitrary cache line. The RMTs are updated seamlessly by the cache replacement algorithm – whenever a cache line replacement occurs, the corresponding RMT entry is updated. In practice, however, an extra level of indirection is undesirable. In section III.D, we show that the indirection can be avoided by clever circuit implementation.

The proposed cache also adopts the direct-mapped architecture to inherit its fast access time and power efficiency. To avoid excessive conflict misses, a *longer cache index* is introduced. Unlike in traditional direct-mapped caches where using more index bits exponentially increases the cache size, *the proposed cache exploits the dynamic memory-to-cache mapping to achieve low conflict misses without increasing its physical size*. This is illustrated in Fig.1. Assuming that the cache contains 2^n physical cache lines, it uses $n+k$ index bits rather than n index bits as in a traditional direct-mapped cache. This is conceptually equivalent to mapping the memory space to a large *logical* direct-mapped cache with 2^{n+k} lines, referred to as the *LDM cache* in the rest of this paper. Note that the LDM cache does not physically exist and is introduced only to facilitate the analysis and discussion of the proposed cache architecture. The dynamic remapping mechanism enables the proposed cache to adapt to store the most useful 2^n lines at run time, rather than holding a fixed set of cache lines and missing on others. To remember which lines in the LDM cache are stored in the real cache, each physical cache line is associated with a Line Number register (LNreg), which stores the $(n+k)$-bit line number of the corresponding logical cache line in the LDM cache. An LNreg physically implements an entry of the ReMapping Table (RMT), and changing the line numbers

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

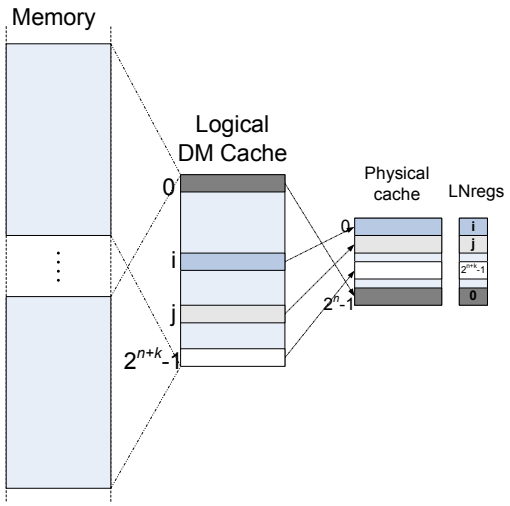

Figure 1. Mapping memory space to the physical cache

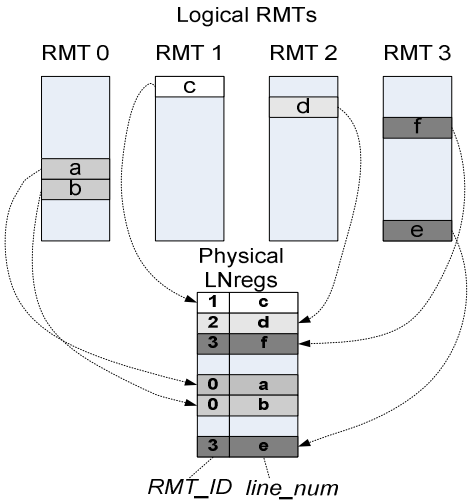

Figure 2. Supporting multiple logical RMTs

stored in an LNreg maps another logical cache line to the physical cache line. Although we assume 2^n cache lines in the above discussion, the number of cache lines s in our proposed cache can be any number – not necessarily a power of two – as long as $s < 2^{n+k}$.

An RMT stores a memory-to-cache mapping. For security as well as performance reasons, it is desirable to have multiple mappings, each of which may be used by one or more processes. Note that although logically multiple RMTs are required, they are physically implemented with one set of LNregs. This is because at any time, for each physical cache line storing a logical cache line, only the entry of the RMT associated to the logical cache line needs to be stored in the LNreg. The corresponding entries in all other RMTs are invalid since these logical cache lines of the other RMTs are not mapped to the physical cache line. Fig.2 shows how a single set of LNregs implement multiple logical RMTs. To distinguish which RMT the entry in an LNreg belongs to, an *RMT_ID* field is included in each LNreg in addition to the *line_num* field.

B. A Summary of the Proposed Cache Architecture

Our proposed cache architecture (Fig.3) is very similar to the traditional direct-mapped cache architecture, with some significant differences summarized below:

- The address decoder of the proposed cache is modified to implement dynamic memory-to-cache mapping. The LNregs are integrated into the address decoder.
- More address bits, $n+k$, are used as index to access a cache of size $s < 2^{n+k}$. A memory address is mapped into a Logical Direct Mapped cache of size 2^{n+k}, and dynamically re-mapped into the real cache of size s.
- The number of cache lines is not necessarily a power of two; it can be any $s < 2^{n+k}$.
- Each process is attached to a context RMT ID which specifies the Re-Mapping Table (RMT) it will use. Different processes therefore can have different memory-to-cache mappings if they are attached to different context RMT IDs.
- Each LNreg contains a *RMT_ID* field of d bits and a *line_num* field of $n+k$ bits.
- Each cache line also has a P flag bit, indicating protected cache lines. Each Page Table Entry (and/or segment descriptor, if implemented) also has a PP flag bit, indicating a Protected Page. This memory marking mechanism is similar to RPcache [9]
- A replacement algorithm is needed on cache misses.

Context RMT_ID: This identifies a hardware context, specifying which RMT is used by a process. A process that needs to be protected against information leak from other processes should use a different RMT. The OS is in charge of associating a process with a RMT_ID when the process is assigned a hardware context for execution.

Address decoder and LNregs: In a traditional cache, the address decoder in essence tests a set of conditions (*index* == 0?), (*index* == 1?), … (*index* == 2^n-1?) that compare the index with a series of constants (0 through 2^n-1) and selects one cache line based on the outcome of these comparisons. In the proposed cache, the address decoder tests a similar set of conditions, except that the condition is a variable, viz., the contents of the i^{th} LNreg, [$LNreg_i$], for i = 0, 1, …, s-1. The address decoder activates a cache line if the *RMT_ID field* in $LNreg_i$ matches the d-bit Context RMT_ID *and* if the *line_num* field in $LNreg_i$ matches the $n+k$ index bits. The LNregs are updated when cache line replacements occur. The new line's context RMT_ID and index bits are written to the *RMT_ID* field and *line_num* field respectively.

C. The Security-Aware Random Replacement Algorithm

Unlike in traditional direct mapped caches, a cache replacement algorithm is necessary in the proposed cache due to the dynamic remapping. During a cache miss, the replacement algorithm determines which physical cache line should be selected for holding the new logical cache line. Since

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

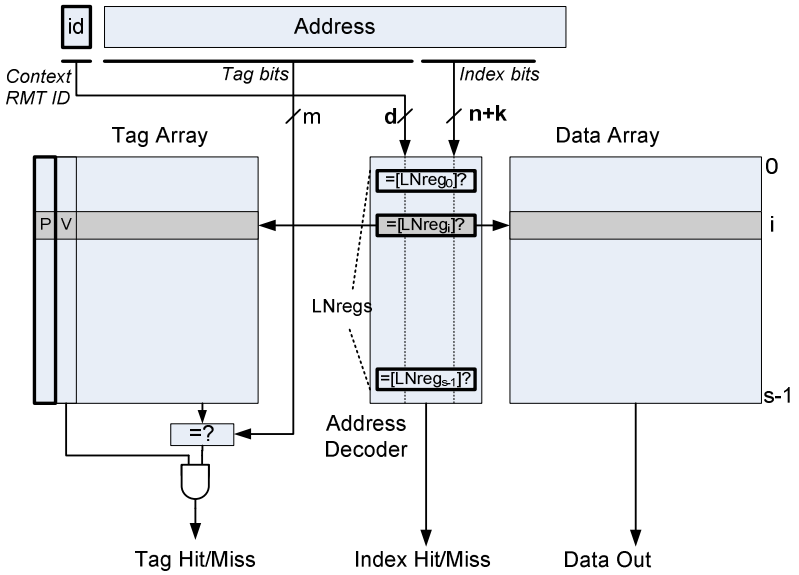

Figure 3. Our proposed cache architecture

replacing the logical cache line that the physical cache line holds normally means mapping a new memory address to the physical cache line, the LNreg (i.e., the physical realization of the logical RMT entry, which stores the corresponding memory-to-cache mapping) of the selected physical cache line needs to be updated accordingly. There are two types of misses, *index misses* and *tag misses*, in the proposed cache. An *index miss* occurs if none of the LNregs matches the given RMT_ID and index. None of the cache lines is selected in an index miss. A *tag miss* occurs if the index hits in one LNreg, but the tag of the selected cache line does not match the address tag. A tag miss essentially is the same as an ordinary miss in a traditional direct-mapped cache, whereas the index miss is a unique type of miss in our proposed cache. Since an index hit means the match of the RMT ID, tag misses only occur within the same process or among processes using the same RMT. Index misses occur early in the hardware pipeline during address decoding, before the tag is read out and compared, and this early miss signal could be used by the pipeline control logic to improve performance.

The replacement policies for the two types of misses are different as we show in Fig.4. The tag misses are conflict misses in the LDM cache since the addresses of the incoming line and the line in cache have the same index (as well as the same RMT ID) but different tags. Because in a direct-mapped cache at most one cache line can be selected at any time, no two LNregs can contain the same index (and the same RMT_ID). Therefore either the original line in cache is replaced with the incoming line or the incoming line is not cached. For index misses, the new memory block can replace any cache line. While various replacement policies can be used to choose the desired victim line to be replaced, we propose a new modified random replacement policy, which we call SecRAND, for the proposed cache, which provides improved security as well as excellent performance. Fig.4 shows the SecRAND replacement algorithm. The cache lines involved and the procedures used in the replacement algorithm are described in Table I.

TABLE I. DEFINITIONS AND NOTATIONS

Notation	Description
C	The cache line selected by the address decoder (during a cache hit or a tag miss).
D	The memory block that is being accessed.
R	The cache line selected for replacement (victim).
P_x	The protection bit of X. If X is in a cache line, it is the P bit of the cache line. Otherwise it is determined by the PP bit of the page/segment that X belongs to.
cache_access(C)	Access C as in a traditional Direct Mapped cache.
victim(C)	Select C as the victim line to be replaced.
victim(rand)	Randomly select any one out of all possible cache lines with equal probability.
replace(R,D)	Replace R with D, update LNreg.
evict(R)	Write back R if it is dirty; invalidate R.
mem_access(D)	Access to D without caching it.

Cache hits (1st column in the flow chart) are handled as in a traditional cache. When a cache miss occurs, if the LNreg of a cache line C matches the Context RMT_ID and index of the memory block D, then this is a tag miss. As a tag miss always indicates a matching RMT_ID, lines C and D must use the same RMT, which usually means that they belong to the same process. We call this interference *internal* to a process or processes in the same security group. If neither the incoming line (D) nor the selected line (C) is protected (2nd column), meaning that the interference is harmless, the miss is handled normally like in a traditional cache. If either C or D are protected (3rd column), meaning that the interference may leak out critical information, the replacement algorithm randomizes the cache interference due to the conflict between C and D. To avoid information-leaking interference, D does not replace C, and since in a tag miss D can not replace cache lines other than C, D is sent directly to the CPU core without being put in the cache. On the other hand, since a miss should normally cause

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

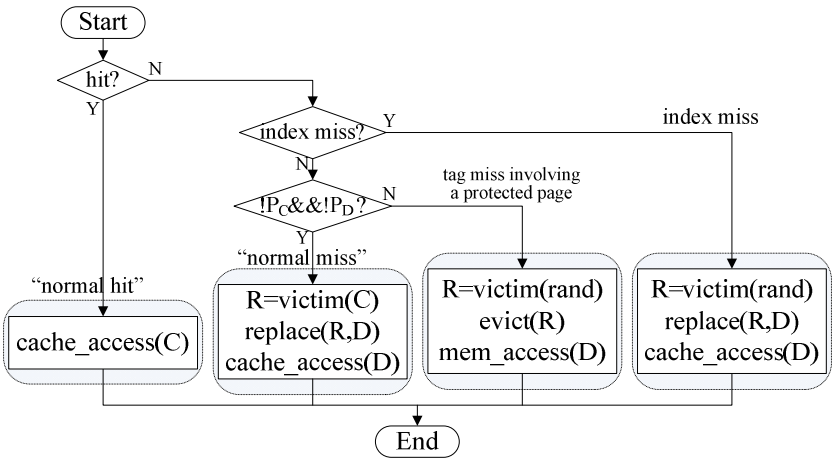

Figure 4. New security-aware random cache replacement algorithm

an eviction, a random line is evicted which "substitutes" for the eviction of C as well as randomizes the interference. Otherwise the old cache lines tend to stay in cache and new cache lines will not get cached. If the miss is not a tag miss, it is an index miss (4^{th} column) – none of the LNregs match the RMT_ID and index of D. In this case, C and D may or may not belong to the same process. Since for an index miss the new memory block D can replace any cache line, a cache line is randomly selected (with equal probability as in the normal RAND) and evicted. The interference caused by an index miss therefore is always randomized. Detailed security analysis of the SecRAND algorithm will be given in section IV.D.

D. Implementation Issues

The proposed cache is very similar to a traditional direct-mapped cache in implementation, except for the new address decoder and the new SecRAND replacement algorithm. We now discuss their implementation issues.

1) The new address decoder design:

According to section III.B, the address decoder of the new cache essentially performs an associative search of the LNregs, looking for a match of the index bits of the address and the RMT_ID (n+k+d bits in total). A traditional way to implement associative search is through Content-Addressable Memory (CAM) [15][16], i.e., the LNregs are implemented as a CAM array. CAM search however is slow and consumes more power. As shown in [16], the word length of each CAM entry is limited to 6 bits to achieve a delay comparable to that of a traditional decoder. Furthermore, having a separate CAM array requires routing its output to the main cache array. This could impact cache access time even more since routing delay has become a dominant factor of the overall cache access latency.

We now discuss two alternative implementations: our previous RPcache address decoder design [9] and our new proposed design. Both designs integrate the comparison logic into the traditional address decoder and avoid the drawbacks of the CAM-based design. The core idea is to make use of the existing logic and routing of the traditional address decoder and maintain similar timing characteristics. To implement the dynamic memory-to-cache mapping, the logic in the traditional address decoders that generate word line selection signals are made *flexible* by using switches. The switches can be controlled to connect different address bits or predecoded bits to the inputs of the logic gates, thus making the logic flexible. Fig.5 shows a comparison of the traditional address decoder, the RPcache decoder and our proposed decoder designed for an example cache configuration. In a real implementation, the exact circuitry may vary depending on the cache organization, the circuit style (e.g., static logic or domino logic) and the fabrication technology, but the same principles still apply.

In the RPcache, the static connections in the traditional address decoder between the outputs of the 3-to-8 pre-decoders and the inputs of the final NOR gates in the address decoder are replaced with dynamic connections via switches controlled by its *permutation registers* (PR). For each switch, a NAND3 is used to generate its control signal. For every 3 address bits, 8 switches and 8 NAND3 gates are needed. The more heavily loaded predecoded lines, due to the drain capacitance of the switches, may be segmented with duplicated drivers.

In our new design, rather than controlling the connections between the predecoded lines and the inputs of the final NOR gates, we control the connections between the address lines and the inputs of the decoder. The 3-8 predecoders are removed and their logic corresponding to each row – a NAND3 gate, is moved to sit beside each word line driver. The switches control how address bits are connected to the NAND3 gate, and thus control which cache line is activated given an index. This implements the dynamic memory-to-cache mapping. As shown in Fig.5, the hardware required in our proposed design, i.e., the switches and logic gates, is less than in the previous RPcache design. In actual circuit implementations, the load on the long wires as well as the input lines of the logic gates that the long wires drive is also lower in our proposed design because of fewer switches and smaller switch sizes. Since our cache has longer index bits, the output of the NAND3 gate corresponding to the extra address bits needs to be ANDed with the output of the NOR gate. This is done by replacing the first inverter in the word line buffer string with a NAND2 gate. By properly adjusting the transistor sizes of the NAND2 gate, no extra delay is introduced. Compared with the RPcache address decoder design, our new approach requires less hardware for implementing switches, has lower loading of the long wires

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

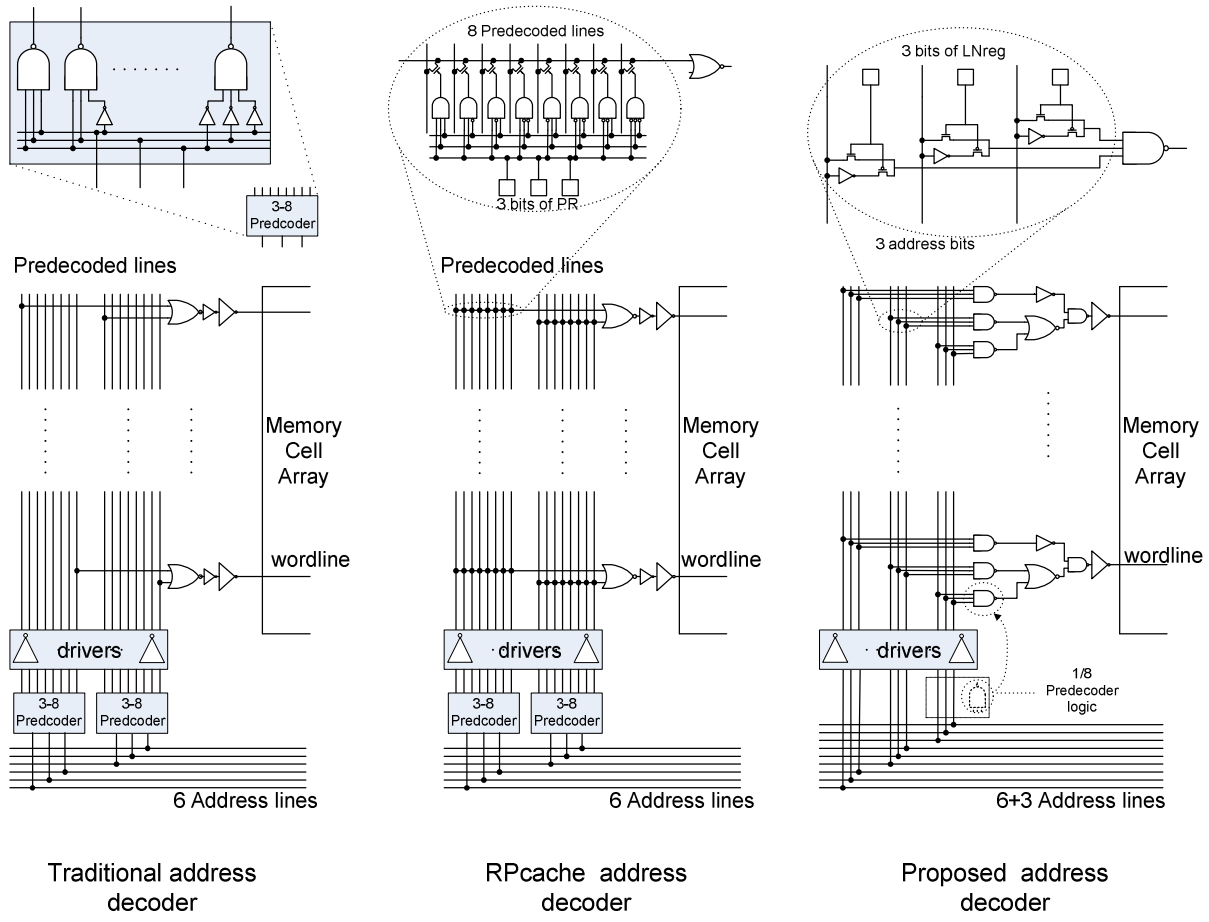

Figure 5. Comparing address decoder circuits

and routes address lines instead of predecoded lines along the edge of the memory cell array, reducing the number of long wires and improving power efficiency.

Since our new proposed decoder design is largely based on the traditional decoder design, the extra implementation overhead is minimized. The extra overhead for combinational logic is very low. In the example shown in Fig.5, for each cache line that probably contains several hundreds of memory cells and port switches, the extra circuits required only include 3 NAND3 gates, 10 inverters and 18 switches, and all these devices are about the minimal size since they are all minimally loaded. The overhead for storage, i.e., the LNregs, is also low. We assume that the LNregs are laid out beside the memory cell array and implemented with the same memory cells. Since each cache line is associated with one LNreg, the overhead of LNregs relative to the overall cache storage is $(n+k+d)/M$, where n,k,d are defined as in Fig.3, M is the total number of memory cells in each cache line including data, tag, flags and ECC bits. For example, in a 64KB cache with 64-bit address and 64-byte cache line size, n=10. The value of M varies since the numbers of tag, flags and ECC bits are implementation dependent. As a rough estimation, we assume there are ~50 bits in total for tag/flag/ECC bits and therefore M≈64x8+50=562. If we allow 4 RMTs and wish to achieve good performance, we can choose d=2 and k=4. The relative overhead of storage will be 16/562 ≈ 2.9%. In some cache implementations the tag array and the data array may be separated, requiring two sets of address decoders. The overhead will be 5.8% in this case.

2) Implementation issues of SecRAND:

Compared with other commonly used replacement algorithms such as LRU, pseudo LRU and FIFO, the random replacement algorithm requires the least hardware cost to implement, due to its stateless nature [25]. Similarly, our SecRAND is stateless and enjoys the same advantage. Although SecRAND requires condition checks, these checks are simple and stateless, thus can be trivially implemented with simple combination logic. The security of SecRAND relies on the quality of the random source. This requires a true or pseudo random number generator (RNG or PRNG) on chip. The design of these is out-of-scope for this paper. Furthermore, we assume that for any system interested in security, a good RNG or a PRNG (e.g., [26]) is already implemented.

IV. Analysis and Evaluation

A. Performance: Cache Access Time

The performance of a cache architecture depends on short access times and low miss rates. We use CACTI 5.0 [10] to explore the design space and find the optimal access times and power consumption. The code corresponding to the address decoder is modified to model the logic shown in Fig.5. More

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

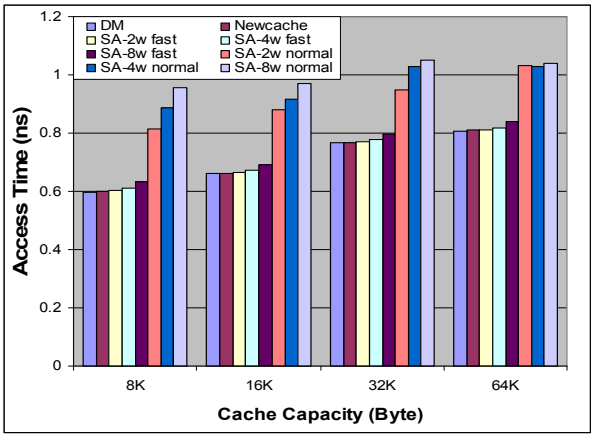

Figure 6. Cache access time comparison

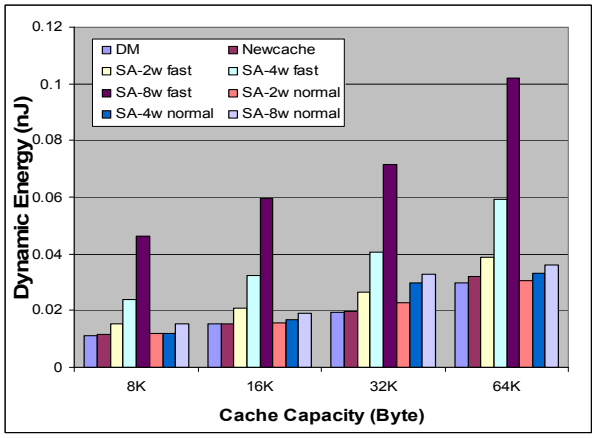

Figure 7. Dynamic read energy

accurate transistor level simulation is also performed using HSPICE. The transistor netlists corresponding to the circuit used in CACTI are constructed with the 65nm Predictive Technology Model (PTM) [17]. To accurately model the long wires in the decoder circuitry, we manually extract the parameters of long wires based on the geometrical information generated by CACTI. We focus on fast L1 caches since these are more impacted than L2 and L3 caches. Fig.6 shows the results on overall cache access time generated by CACTI. The extra delay introduced by our proposed cache, referred to as "Newcache" in the discussion below, is always within 1% range of the access time of a traditional direct-mapped (DM) cache. We also compared the access times of commonly used set-associative (SA) caches that are 2-way, 4-way or 8-way set-associative. The "fast" caches are optimized for speed whereas the "normal" caches are optimized for both speed and power efficiency. The data are generated by configuring CACTI with fast mode and normal mode, respectively. Although a fast SA cache could have an access time close to that of our cache, the power consumption is significantly higher – up to 4 times higher than our Newcache, as shown in Fig.7. Table II shows the HSPICE results for a traditional direct-mapped cache versus our proposed Newcache. In all cases, the extra delays are no greater than 5ps, which is less than 1% of the overall access times.

TABLE II. HSPICE RESULTS ON ADDRESS DECODER DELAY

	8KB	16KB	32KB	64KB
Traditional DM	0.149ns	0.149ns	0.226ns	0.192ns
Newcache	0.151ns	0.151ns	0.230ns	0.197ns

B. *Performance: Cache Miss-rate Analysis*

1) *Theoretical analysis*

Cache misses have been classified as compulsory misses, capacity misses or conflict misses. Compulsory misses (e.g., on a cold start) are common to all caches. Capacity misses (e.g., when the program's working size exceeds the size of the cache) only depend on cache size. Conflict misses have been shown to depend on the cache organization (e.g., set-associativity) and capacity. To reduce conflict miss rate, a traditional way is to increase associativity. However, this impacts cache access time and power efficiency. Increasing capacity can reduce capacity misses as well as conflict misses. However, this is often not feasible in practice due to the limited silicon real estate budget.

In contrast, we show, for the first time, that conflict misses can be largely independent of cache capacity. Our analysis shows that, regardless of its real capacity, our proposed Newcache with an $(n+k)$-bit index has *less conflict misses* than a traditional direct-mapped cache with 2^{n+k} cache lines. The total number of misses in our Newcache has the following bounds:

$$|Miss(Newcache,2^n)| \le |CompulsoryMiss| + |CapactiyMiss(2^n)| + |ConflictMiss(DM,2^{n+k})| \quad (1)$$

$$|Miss(Newcache,2^n)| \ge \max\{|Miss(DM,2^{n+k})|,|Miss(FA,2^n)|\} \quad (2)$$

where *Miss*(*Arch*, *Size*) denotes the set of misses in a cache of type "Arch" with a capacity of "Size" and |A| is the number of elements in set A. Detailed analysis can be found in Appendix A. In equation (1), the left side of the equation can be decomposed to the same first 2 terms as the right side plus a third term: *ConflictMiss*(*Newcache*,2^n). Hence, (1) shows that the conflict misses of our new cache is less than or equal to that of a direct-mapped cache with 2^{n+k} cache lines. Indeed, as verified in the next section, this bound is asymptotically tight and is a good approximation of the true miss rate in real configurations. This means that *the conflict misses of our proposed Newcache are largely independent of its actual cache capacity.* The conflict misses are indeed dependent on the size of the larger LDM cache, 2^{n+k}, rather than on the actual cache size, 2^n. This property of our proposed cache gives cache designers the ability to control the conflict miss rate at the desirable level by choosing the proper number of index bits, while choosing the capacity independently based on cost or other needs. This avoids the speed and power penalty due to higher associativity and allows finer-grained control on allocating capacity to the cache and making the best use of the resource. This property also enables other benefits that traditional caches can not provide, as we will show in section V.

2) *Simulation results*

For experimental confirmation of miss rates, we simulated our proposed Newcache and traditional direct mapped (DM), set-associative (SA) and fully-associative (FA) caches on a cache simulator derived from sim-cache and sim-cheetah of the

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

simplescalar toolset [24]. We run all 26 SPEC2000 benchmarks for 1 billion instructions with appropriate fast forward counts ranging from 2 million instructions to 3 billion instructions. Fig.8 illustrates the accuracy of the bounds we derived in equations (1) and (2). The bounds are normalized to the real miss rate to show the relative accuracy. The simulation is done for our proposed caches with 64-byte lines for n = 6 to 10 (i.e., 4K bytes to 64K bytes capacity), with cache indices that are k=3 to 4 bits longer. Except for one point, the bounds are always within the 10% range of the real miss rate, and when $n+k$ or k gets larger, the accuracy increases. Indeed, the derived bounds are asymptotically tight, meaning that the equality in (1) holds when k and $n+k$ are large.

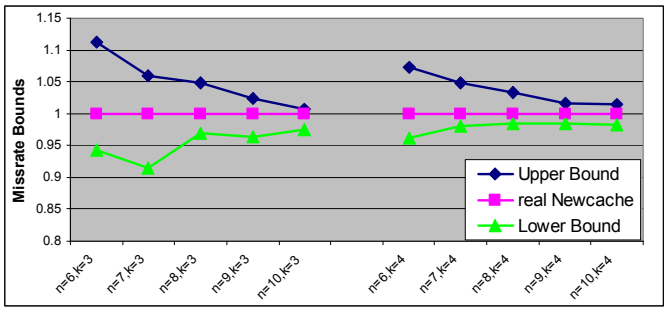

Figure 8. Accuracy of miss rate bounds

Table III compares the miss rates of our Newcache with the DM cache and the 2-way and 4-way SA caches with LRU replacement. FA caches and 8-way SA caches with RAND replacement are also included to show the effectiveness of our SecRAND replacement algorithm. The lowest miss rate in each column is highlighted in bold (and normalized to 1 in parenthesis). The miss rates of our new caches are in the last 2 rows – our Newcache almost always achieves the lowest miss rates achieved in each column by traditional caches.

C. *Power Efficiency Analysis*

We analyze the power efficiency of the proposed cache with regard to two aspects: the per access energy of the cache and the overall power consumption. The cache miss rates are obtained from simulation of all SPEC2000 benchmarks. The power penalty of misses, i.e., the per access energy of L2 cache is obtained using CACTI 5.0. Modern caches are usually organized as a set of subarrays to achieve fast timing and low power dissipation. The main sources of dynamic power include the power for routing address bits in and data bits out via H-trees, and the power on word lines and bit lines since they are heavily loaded. As our Newcache is direct-mapped, only a minimum number of subarrays need to be activated in each access, which minimizes the power consumed on word lines and bit lines, giving the low per access energy.

Fig.7 shows the dynamic read energy data generated by CACTI. The impact of the changes on the overall power consumption compared to DM caches is very low – less than 2%. This is because the percent of energy consumed by the modified structures in our proposed Newcache architecture is low. The new address decoder (excluding word lines since they are not changed) consumes just a few percent more than a traditional DM cache, and the whole decoder power consumption is normally less than 5% of the overall dynamic power. The LNregs consume little power because they are a small amount of memory compared with the size of the cache and have low switching activities – the contents of LNregs need to be changed only during an index miss. Furthermore, unlike accesses to other memory cells, most accesses to LNregs do not involve power-consuming bit-line charging and discharging. Only writes to LNregs require bit-line operations, which occur only when index misses happen. The increase in leakage power in our Newcache is mainly due to the memory cells in LNregs, which is small relative to the overall cache. Hence, the leakage power increase is also very low.

Fig.9 shows the results comparing the overall power consumption normalized to our Newcache. We compare traditional SA caches as well as advanced low power SA caches – the way-predicting (wp) SA cache. For example, "SA 4w LRU wp0.7" means a 4-way set-associative way-predicting cache with prediction accuracy of 0.7, and LRU replacement algorithm. All caches are 32KB with 64Byte cache lines. The miss rates of the cache impact the overall system power consumption. A higher miss rate means more accesses to the larger caches or the main memory which consume more power. Our Newcache is more power efficient than the others due to its low miss rate and low per access energy. On average, the 4-way SA cache consumes 61% more power than our Newcache, the 2-way SA cache 20% more, the DM cache 8% more, the 4-way way-predicting cache 16% and 6% more with 0.7 [22] and 0.85 accuracy[23], respectively.

D. *Security Analysis*

The proposed cache adopts the randomization approach used in RPcache on cache misses to mitigate information leakage. The SecRAND replacement algorithm essentially achieves the equivalent randomization effect on cache misses. We therefore show a similar analysis to prove the security of

TABLE III. CACHE MISS RATES WITH DIFFERENT ARCHITECTURES AND REPLACEMENT ALGORITHMS (NORMALIZED RESULTS IN PARENTHESIS)

	4KB	8KB	16KB	32KB	64KB
DM	0.133	0.093	0.068	0.055	0.048
SA-2way, LRU	0.101	0.075	0.057	0.045	0.041
SA-4way, LRU	0.096	0.068	**0.053 (1)**	**0.042 (1)**	**0.040 (1)**
SA-8way, RAND	0.095	0.071	0.054	0.044	0.041
FA, RAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044	**0.040 (1)**
Newcache k=4, SecRAND	0.093 (1.033)	0.068 (1.015)	0.054 (1.019)	0.044 (1.048)	0.041 (1.024)
Newcache k=6, SecRAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044 (1.048)	**0.040 (1)**

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

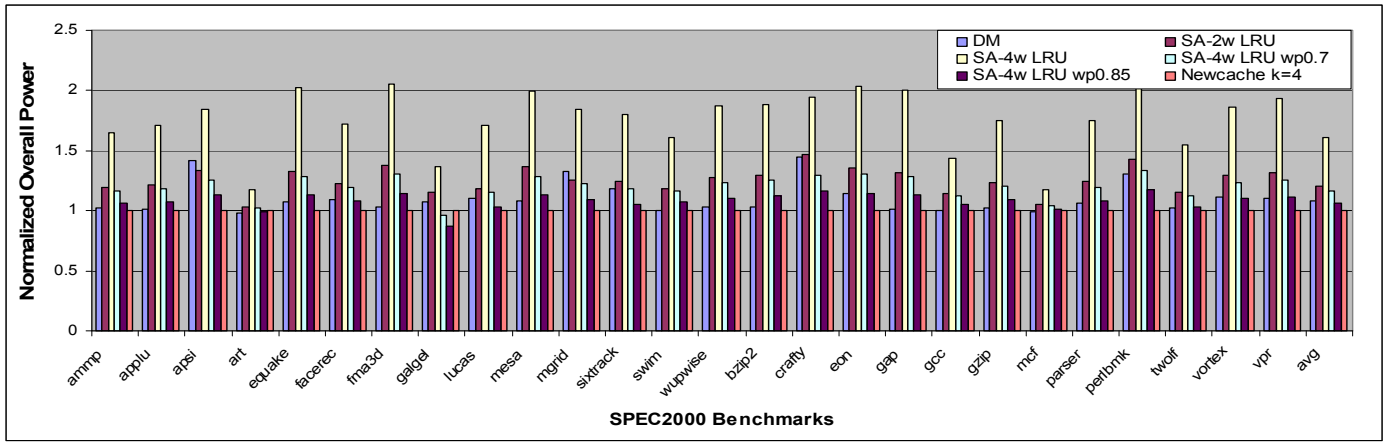

Figure 9: Comparison of the overall power consumption

our Newcache architecture. Similar to the analysis of RPcache, we model the information leakage channel as a classic discrete time synchronous channel. The input symbol of the channel is the line number of the cache line accessed by the victim that would cause an eviction and the output symbol is the line number of cache line for which the attacker observes an eviction. Note that the same physical cache line may have different line numbers from the victim and attacker's points of view (e.g., in the proposed cache, they may use different RMTs). To make the capacity of this channel zero, the randomization should meet the following requirement for all protected cache lines:

$$P(j|i) = P(j'|i), \forall\ i,j,j' \qquad (3)$$

where $P(j|i)$=Pr($output=j|input=i$). In other words, given an access at line i by the victim that would cause an eviction, the attacker can observe an eviction at any line number with equal probability. From the attacker's point of view, although the attacker can observe a cache eviction, he has no idea which cache line was accessed by the victim. Below we show that the proposed cache meets this condition. Given a cache miss that causes eviction, the following cases need to be considered.

a) The miss is an index miss. According to Fig.4 (4th column), a random cache line R is selected for eviction with equal probability. In other words, for any victim's access that would cause an eviction, all cache lines have the same probability to be evicted, i.e., $P(j|i) = P(j'|i), \forall\ i,j,j'$.

b) The miss is a tag miss that involves protected cache lines. As shown in Fig.4 (3rd column), the line to be evicted is also randomly selected with equal probability, i.e., $P(j|i) = P(j'|i), \forall\ i,j,j'$.

Clearly, the proposed SecRAND randomization mechanism satisfies (3), and thus achieves zero channel capacity.

V. Additional Benefits

Fault tolerance: Memory-to-cache remapping is a common technique used in fault-tolerant cache design. In traditional caches, a memory block mapped to a faulty line/set is statically remapped to another good line/set [12-14]. Such schemes increase the number of conflict misses since the remapped cache line/set is now shared by more memory addresses. They also increase the number of capacity misses since the faulty lines reduce cache capacity. The proposed cache architecture can provide fault tolerance in a similar manner using remapping, but with better performance. As shown in section IV.B, due to the dynamic memory-to-cache mapping of our Newcache architecture, a cache of size s with p faulty cache lines is equivalent to a cache of size s-p, which has the same conflict miss rate as shown by (1). In other words, faulty cache lines in our proposed cache only increase capacity misses, but not conflict misses.

Hot-spot mitigation: Due to spatial and temporal locality, the references to a small number of cache lines account for a majority of the total cache references. The more frequently accessed cache lines generate more heat, causing hot spots. Such unevenly distributed cache line accesses however are mostly avoided in our proposed Newcache. The SecRAND replacement algorithm maps memory blocks to randomly selected physical cache lines, which avoids clustering of frequently accessed cache lines.

Optimization for power efficiency: With the ability of mapping memory blocks to arbitrary physical cache lines, our Newcache architecture can also facilitate low power design. For example, by adaptively turning off cache lines based on a program's working set, the power efficiency of the cache can be further improved with minimal impact on performance. An analysis similar to that in the discussion of fault tolerance can show that turning off cache lines in the proposed cache will cause fewer additional cache misses than in traditional caches.

Benefits for cache partitioning and locking: In traditional caches such as set-associative caches, cache partitioning is not trivial and has many restrictions [27]. A set-associative cache can be partitioned in two ways: horizontal partitioning and vertical partitioning. Horizontal partitioning divides cache sets into subgroups, each of which forms a partition. One issue with this scheme is that the number of cache sets in each partition has to be a power of 2. This severely limits the flexibility of choosing a partition size. In addition, the address decoder has to be redesigned so that it can be reconfigured to index different numbers of cache sets. Vertical partitioning partitions cache "ways" (degrees of associativity) into subgroups. As most caches have limited associativity, the number of partitions

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

can be very limited. In addition, the partitions have lower associativity than the original cache, thus incurring higher conflict miss rates. Cache line locking is a more flexible way to "partition" a cache, as in PLcache [9]. It however also suffers from higher conflict miss rates. In a set-associative cache, the locked line(s) in a cache set reduce the effective associativity of the set, thus incurring more conflict misses. In contrast, as shown in section III, our Newcache does not have restrictions on the number of physical cache lines in a cache. Therefore cache partitioning and locking mechanisms built upon our proposed cache has the highest flexibility in allocating cache lines to a partition. Moreover, as shown in the discussion of fault tolerance, partitioning a cache incurs fewer additional cache misses in our Newcache than in traditional caches, thus providing better performance.

VI. PAST WORK

The past work on mitigating cache based side channel attacks have been reviewed in section II. With respect to performance and power efficiency, past research efforts focused mainly on the following aspects. To reduce conflict misses, the victim cache [18] allows conflicting lines to be stored in a small FA cache. The hash-rehash cache [19] and the column-associative cache [20] use a different indexing function when a conflict miss occurs such that the missing line can be put into an alternative location. These approaches can maintain fast cache access time, but the performance improvement is limited. The adaptive group-associative cache [21] can dynamically allocate cache holes for conflicting addresses to reduce conflict misses. Cache holes are lines with indices that have little usage. However, it incurs penalty for accessing reallocated cache lines even for hits, unlike our solution. The B-cache [16] employs a programmable address decoder with long cache index, but its address decoder can only be partially programmable, up to 6 bits, limited by its CAM-based implementation. In contrast, our address decoder allows full dynamic mapping, for security and performance. To reduce power consumption, way-predicting techniques [22][23] avoid accessing all ways of set-associative caches in every access – but with less effectiveness than our proposed architecture.

Unlike most of the previous work, a major advantage of our proposed cache is that, with a single cache architecture, many challenging and even conflicting design goals such as security and high performance can be achieved at the same time. Our proposed Newcache architecture has low miss rates comparable to highly associative caches, and is as fast and power efficient as direct-mapped caches. The Newcache architecture is also secure, yet requires lower hardware cost. Our novel multiple logical RMTs remapped into a single set of physical LNregs is much more efficient than the multiple physical permutation tables in RPcache, and our address decoder design needs less hardware and is more power efficient. Moreover, our cache architecture allows cache partitioning and cache line locking to be efficiently implemented yet avoids cache underutilization, providing better security and performance than existing partitioned cache and PLcache [9] architectures. Our proposed architecture also provides the nice property of capacity independent conflict misses, and can bring additional benefits including fault tolerance and hot-spot mitigation.

VII. CONCLUSIONS

In addition to high performance and low power dissipation, security has become a critical issue for cache design due to its susceptibility to software side-channel attacks. While design for security and design for performance are usually at odds, this paper shows that it is possible to have a cache architecture that simultaneously improves both security and performance – and also improves on power consumption, fault-tolerance and hot-spot mitigation. We hope this motivates a rethinking of cache and computer design, based on improving security without sacrificing other design goals like performance and power consumption.

We presented a novel cache architecture which combines dynamic remapping of multiple larger Logical Direct Mapped caches to a single smaller physical cache using extra index bits, hardware context awareness, a security-aware random replacement algorithm and a physical design that does not increase cache access latency. Our results show that our proposed Newcache architecture is as fast as a direct-mapped cache, has a miss rate as low as that of a highly-associative cache, and is as power efficient as a direct-mapped cache. At the same time it can thwart cache-based software side-channel attacks, providing both legacy and security-enhanced software a much higher degree of security. It can also bring additional benefits such as fault tolerance and hot-spot mitigation. Further, we show that the conflict miss rate of our new cache is independent of its physical capacity, enabling efficient, fine-grained cache resource utilization.

REFERENCES

[1] D.J. Bernstein, "Cache-timing Attacks on AES," available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[2] C. Percival, "Cache Missing for Fun and Profit," available at: http://www.daemonology.net/papers/htt.pdf

[3] D. A. Osvik, A. Shamir and E. Tromer, "Cache attacks and Countermeasures: the Case of AES", *Cryptology ePrint Archive*, Report 2005/271, 2005.

[4] Onur Aciçmez, Werner Schindler, and Çetin Kaya Koç, Cache Based Remote Timing Attack on the AES, in *RSA Conference 2007*, Cryptographers' Track.

[5] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. Cryptographic Hardware and Embedded Systems -- CHES 2006, pages 201-215, 2006.

[6] Onur Aciiçmez, and Cetin Kaya Koc. Trace driven cache attack on AES, IACR Cryptology ePrint Archive, Report 2006/138, April 2006.

[7] Ernie Brickell and Gary Graunke and Michael Neve and Jean-Pierre Seifert. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. *IACR ePrint Archive*, Report 2006/052, Feb 2006.

[8] D. Page, "Partitioned Cache Architecture as a Side-Channel Defense Mechanism", *Cryptology ePrint Archive*, Report 2005/280, 2005.

[9] Zhenghong Wang, Ruby B. Lee, "New Cache Designs for Thwarting Software Cache-based Side Channel Attacks", Proceedings of the 34th International Symposium on Computer Architecture, San Diego, CA, pp. 494 - 505, June 2007.

[10] Thoziyoor, Shyamkumar; Muralimanohar, Naveen; Jouppi, Norman P., CACTI 5.0, *PHL Techincal Report* HPL-2007-167.

[11] Ooi, Y., M. Kashimura, H. Takeuchi, and E. Kawamura, "Fault-Tolerant Architecture in a Cache Memory Control LSI," *IEEE J. of Solid-State Circuits*, Vol. 27, No. 4, pp. 507-514, April 1992.

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.

[12] Philip P. Shirvani, Edward J. McCluskey, "PADded Cache: A New Fault-Tolerance Technique for Cache Memories," 17th IEEE VLSI Test Symposium, p. 440, 1999.

[13] Mutyam, M. and Narayanan, V., Working with process variation aware caches. In *Proceedings of the Conference on Design, Automation and Test in Europe* (DATE'07), 2007.

[14] Lee, H., Cho, S., and Childers, B. R., Performance of Graceful Degradation for Cache Faults. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, p.409-415, 2007.

[15] Michael Zhang and Krste Asanovic, "Highly-Associative Caches for Low-Power Processors", Kool Chips Workshop, MICRO-33, Monterey, CA, December 2000.

[16] Zhang, C., Balanced Cache: Reducing Conflict Misses of Direct-Mapped Caches. In *Proceedings of the 33rd Annual international Symposium on Computer Architecture*, p.155-166, 2006.

[17] Predictive Technology Model. http://www.eas.asu.edu/~ptm

[18] Jouppi, N. P., Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual international Symposium on Computer Architecture*, p.364-373, 1990.

[19] Anant Agarwal , John Hennessy , Mark Horowitz, Cache performance of operating system and multiprogramming workloads, ACM Transactions on Computer Systems, v.6, p.393-431, Nov. 1988.

[20] Agarwal, A. and Pudar, S. D., Column-associative caches: a technique for reducing the miss rate of direct-mapped caches. ISCA '93, p.179-190.

[21] Peir, J., Lee, Y., and Hsu, W. W., Capturing dynamic memory reference behavior with adaptive cache topology. In ASPLOS-VIII. P.240-250, 1998.

[22] Powell, M. D., Agarwal, A., Vijaykumar, T. N., Falsafi, B., and Roy, K., Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the 34th Annual ACM/IEEE international Symposium on Microarchitecture*, 2001.

[23] Koji Inoue , Tohru Ishihara , Kazuaki Murakami, Way-predicting set-associative cache for high performance and low energy consumption, Proceedings of the 1999 international symposium on Low power electronics and design, p.273-275, 1999.

[24] http://www.simplescalar.com

[25] Al-Zoubi, H., Milenkovic, A., and Milenkovic, M. Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite. In *Proceedings of the 42nd Annual Southeast Regional Conference*.

[26] Fischer, V. and Drutarovský, M. 2003. True Random Number Generator Embedded in Reconfigurable Hardware. In the 4th international Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), 2002.

[27] Parthasarathy Ranganathan, Sarita V. Adve, Norman P. Jouppi: Reconfigurable caches and their application to media processing. ISCA 2000: 214-224.

APPENDIX A

In this analysis, we consider three cache types, our proposed Newcache, DM and FA, as explained in Table A. We assume LRU replacement policy for FA caches and Newcache for the ease of analytical analysis. RAND and SecRAND algorithm should lead to similar properties in a statisitcal sense since they statisically approximate LRU: even though each cache line can be evicted with equal probability in each individual cache miss, statistically the more frequently accessed lines have higher probability of residing in the cache, and a cache line residing in the cache without being accessed for a long time has a higher probability of being evicted.

Proof of the upper bound (1): The proof of the upper bound is based on three facts. We first define the terms that will be used. The *reuse distance d* is the number of *distinct* block addresses between two consecutive appearances of the same block address. *Compulsory misses* are those due to the first access of the data and have a resue distance $d = \infty$. *Capacity misses* are those due to insufficient cache capacity. In a cache with m blocks, a miss is a capacity miss if the block address has a reuse distance $d > m$. *Conflict misses* are those that are neither compulsory misses nor capacity misses, i.e., $d \le m$.

TABLE A. CACHE ARCHITECTURES CONSIDIERED

Caches considered	Description
Newcache of size s	our Newcache, with more index bits referring to a larger Logical Direct Mapped cache than the size of the physical cache (for simplicity we assume $s=2^n$)
DM cache of size 2^{n+k}	Direct Mapped cache (which is the LDM cache in Fig. 1.) The physical cache of the proposed architecture holds a subset of the lines in the LDM cache
FA cache of sizes 2^n	Fully Associative cache (for calculation of capacity misses and compulsory misses)

Fact 1: An index miss in the proposed Newcache is either a compulsory miss or a capacity miss but the opposite is not necessarily true.

Proof: In an index miss, the index of the address is not found in the LNregs, which means that this index either never appeared before, or there were more than 2^n distinct indices since the last appearance of the current index. In other words, the reuse distance of the address is greater than 2^n, and hence an index miss is always a capacity miss or a compulsory miss. On the other hand, a capacity miss or a compulsory miss is not necessarily an index miss. For example, at the first time an address is accessed, the same index may already exist in one LNreg due to a previous access to another address with the same index, and hence leads to a tag miss. This proves Fact 1.

Fact 2: A conflict miss is always a tag miss in Newcache but a tag miss is not necessarily a conflict miss.This is indeed the contraposition of Fact 1.

Fact 3: Considering the misses that occur in the cache architectures we examined, the following relationship holds:

$$Miss(Newcache,2^n) \subseteq Miss(DM,2^{n+k}) \cup Miss(FA,2^n) \quad \text{(i)}$$

$$CompulsoryMiss \cup CapacityMiss(2^{n+k}) \subseteq Miss(DM,2^{n+k}) \cap Miss(FA,2^n) \quad \text{(ii)}$$

where $Miss(Arch, Size)$ denotes the set of misses in a cache of type "Arch" with a capacity of "Size".

Proof: In the proposed cache, the index misses are always misses in the FA cache, by Fact 1. The tag misses are always misses in the LDM cache since whenever an index conflict occurs in the physical cache it must occur in the LDM cache. This proves (i). To prove (ii), consider the compulsory misses and capacity misses in the LDM cache. They are first a subset of $Miss(DM,2^{n+k})$ that includes all misses of the LDM cache. Also, since they have reuse distances $d > 2^{n+k}$, they must also be misses in a FA cache of size 2^n. In other words, they belong to $Miss(DM,2^{n+k}) \cap Miss(FA,2^n)$.

With (i) and (ii), we have

$$\begin{aligned}
&|Miss(Newcache,2^n)| \\
&\le |Miss(DM,2^{n+k})| + |Miss(FA,2^n)| - |Miss(DM,2^{n+k}) \cap Miss(FA,2^n)| \\
&\le |Miss(FA,2^n)| + |Miss(DM,2^{n+k})| - |CompulsoryMiss \cup CapacityMiss(2^{n+k})| \\
&= |Miss(FA,2^n)| + |ConflictMiss(DM,2^{n+k})| \\
&= |CompulsoryMiss| + |CapacityMiss(2^n)| + |ConflictMiss(DM,2^{n+k})| \quad \square
\end{aligned}$$

Proof of the lower bound (2): As mentioned earlier in section III, at any time the real physical cache stores a subset of the cache lines in the conceptual *LDM* cache. Therefore, given an arbitrary memory address, if it hits in the physical cache, it must also hit in the *LDM* cache. On the other hand, if it hits in the *LDM* cache, it may not necessarily hit in the physical cache – it will miss if the line being accessed in the *LDM* cache is not yet mapped into the physical cache. We therefore have $|Miss(Newcache,2^n)| \ge |Miss(DM,2^{n+k})|$. On the other hand, The proposed cache should have higher miss rate than the fully associative cache of the same size since it has conflict misses that the fully associative cache does not have, i.e., $|Miss(Newcache,2^n)| \ge |Miss(FA,2^n)|$. $\square$

Authorized licensed use limited to: University of Michigan Library. Downloaded on May 27,2020 at 15:05:10 UTC from IEEE Xplore. Restrictions apply.