2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2235-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/HPCA51647.2021.00036

Trident: A Hybrid Correlation-Collision GPU Cache Timing Attack for AES Key Recovery

Jaeguk Ahn, Cheolgyu Jin, Jiho Kim, Minsoo Rhu, Yunsi Fei†, David Kaeli†, John Kim
KAIST †*Northeastern University*
Daejeon, Republic of Korea Boston, MA USA
{jaegukahn, talkingdeer12, jihokim, mrhu, jjk12}@kaist.ac.kr {yfei, kaeli}@ece.neu.edu

***Abstract*—Given the parallel processing capabilities of Graphics Processing Units (GPUs), many applications are exploiting GPUs and cryptographic systems have also begun to leverage GPUs to accelerate encryption/decryption. Recent work has identified how microarchitectural side-channel attacks can be carried out on AES (Advanced Encryption Standard) by exploiting the SIMT characteristics and memory coalescing of GPUs. In this work, we first show that previously proposed correlation-based side-channel attacks are not feasible on modern GPUs that support narrower data-cache accesses via a sectored-cache microarchitecture – resulting in memory accesses from different levels of the memory hierarchy. In comparison, we identify how *negative* timing correlation can occur in modern GPUs when data is fetched from different levels of the cache hierarchy. We then propose *Trident* – a hybrid cache-collision timing attack on GPUs that can fully recover all AES key bytes on modern GPUs. Cache collisions in GPUs present challenges due to the large number of threads and the number of samples required. To address these challenges, Trident consists of three different components – negative timing correlation, cache-collision attack, and chosen plaintext attack. We leverage the negative timing correlation to recover earlier key bytes of AES while exploiting cache-collision attacks for the latter AES key bytes. To enable GPU cache collision attacks, we exploit memory coalescing to control the number of memory accesses through chosen-plaintext attacks to significantly reduce the number of timing samples needed. Our proposed Trident attack results in over 10× reduction in the number of samples needed to recover the key bytes compared with prior work, while still being successful in full AES key recovery in modern GPUs. We also propose *TridentShield* – a latency-based countermeasure to the Trident attack that minimizes throughput degradation in GPUs.**

***Index Terms*—GPU, AES, side-channel attack, sectored-cache**

I. Introduction

As the cloud is becoming more widely used, data privacy and security in the cloud has become more important and cloud service providers need to safely store data using encryption. AES is a commonly used symmetric cipher that is highly effective in encrypting a large amount of data and current CPUs provide AES-NI instructions to accelerate AES while providing security features in the ISA [15]. However, CPUs cannot provide high throughput compared to GPUs when encrypting a large amount of data [47]. As a result, many recent studies have focused on AES acceleration using GPUs [1], [16], [19], [30]. However, as GPUs become more widely used, GPUs have also become a target for security vulnerability attacks and side-channel attacks. Recently, Jiang et al [21] demonstrated a timing side-channel attack on a GPU, based on the SIMT (single instruction multiple threads) architectural characteristics of GPUs – in particular, the timing information can be leaked based on the memory coalescing behavior (which we refer to as *SIMT leakage*). The GPU's coalescing unit can merge multiple requests generated by GPU cores into a single memory transaction when requests access the same or contiguous memory addresses. Since coalesced requests are handled one per cycle and serialized, SIMT leakage leads to a positive correlation between the number of coalesced requests and the total execution time.

In this work, we analyze how sectored-cache [11], [31], [44] in the cache hierarchy of the modern GPUs impact timing-based correlation attacks. In particular, we show how prior side-channel attack [21] is not applicable to modern GPUs. We show that even though SIMT leakage still exists (to a lesser degree) in modern GPUs, the positive correlation that was observed earlier no longer exists because of cache lines being accessed at the "sector" (i.e., 32B) granularity. Recent GPUs implement sectored-cache [43], [45] and even though the cache line size is 128 bytes, only the sector that is accessed is actually brought into the L1 cache. When data is accessed in 128B granularity and combined with a relatively small T4-table [1] size in AES, all of the table entries can be easily loaded into L1 cache. However, that is no longer the case with a 32B access granularity. As a result, memory accesses will hit in either the L1 or L2 cache and the positive correlation between the number of unique cache lines and overall execution time no longer exists. However, we identify how the *opposite* can occur (or negative correlation) as data that are brought into the L1 cache help to reduce the execution time for latter key bytes. Some of the earlier accessed key bytes can be recovered through negative correlation but negative correlation is not sufficient to recover all key bytes.

To enable a full AES key recovery on modern GPUs, we propose Trident – a novel *hybrid* correlation-cache collision attack on GPUs. It is hybrid as the earlier key bytes (e.g., 1^{st}-4^{th} key bytes) are recovered using negative correlation while the latter key bytes (e.g., 5^{th}-16^{th} key bytes) are recovered using a cache-collision attack. A cache collision attack to recover AES key bytes has been proposed for CPUs [6]; however, CPU approaches are not directly applicable to GPUs because of the large number of threads that are executed

[1]T4-table is the last round lookup-table used in AES encryption (Sec II-B).

978-1-6654-2235-2/21/$31.00 ©2021 IEEE
DOI 10.1109/HPCA51647.2021.00036

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

concurrently in GPUs. In this work, we exploit the timing difference between L1 and L2 cache accesses on a GPU, which can exceed 100 cycles [20], as the basis for our cache collision-based attack. However, GPU cache collision attacks come with unique challenges since a large number of parallel threads are executing concurrently (e.g., 32 threads in a warp) and large number of samples are needed. Cache collision attacks are based on obtaining different timing samples but with small data set size (i.e., 1KB for the T4-table), the entire data can be loaded into the L1 cache – making it infeasible to obtain enough timing samples to carry out a successful cache collision attack. Thus, Trident attack leverages negative correlation to recover earlier key bytes of AES but then, we exploit chosen-plaintext attack to control the amount of memory coalescing to modify the rate at which data is loaded into the L1 cache – thus, enabling a sufficient amount of timing data to be collected for the side-channel attack. With the Trident attack, we demonstrate that we can recover all AES key bytes on modern GPUs with less than 10× the number of samples, as compared to prior correlation-based timing attack [21]. We also propose TridentShield – a countermeasure that does not impact coalescing and has minimal impact on memory bandwidth but leverages memory hierarchy latency differences to prevent cache collision attack in GPUs. In summary, the novel contributions of this work include the following:

- We analyze the impact of sectored cache in GPUs on coalescing-based correlation AES timing attacks. In particular, we show how the reduced data access granularity in sectored-cache leads to *negative* correlation, where *fewer* coalesced accesses can result in faster execution time.
- We propose *Trident* – a novel *hybrid* correlation-collision GPU side-channel attack that leverages both correlation and cache collision, using chosen-plaintext attack, to enable full AES key recovery.
- To provide defense against Trident, we propose *TridentShield* – a latency-based countermeasure that has minimal impact on GPU memory bandwidth and minimizing performance impact while preventing cache-collision attacks.

II. Background

A. GPU Architecture

GPU operates in units of multiple threads, referred to as *warp* or *wavefront*, consisting of 32 or 64 threads. All threads in a warp execute the same instruction but operate on different data and enable significant parallelism. The threads access off-chip memory through on-chip memory partitions that consist of shared L2 cache and memory controller and are placed in front of the off-chip memory. To reduce the number of memory requests, modern GPUs can coalesce or combine multiple memory requests into a single memory transaction. The amount of coalescing is based on the memory access pattern – e.g., all threads from a single warp can be coalesced into a single memory transaction but in the worst-case, no coalescing can be done and result in 32 separate memory transactions for 32 threads in a warp. The amount of coalescing impacts overall performance or the execution time and prior work [21] exploited this behavior in a GPU side-channel attack.

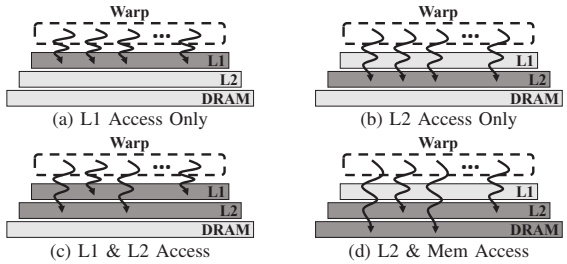


Fig. 1: Different memory access patterns across the GPU memory hierarchy for warp memory requests. For simplicity, the case where accesses can come from L1, L2, and main memory is not shown.

In Figure 1, we categorize different memory accesses that can occur within a single warp across the GPU memory hierarchy. `L1 Access Only` or `L2 Access Only` correspond to memory accesses being serviced from the same level of the memory hierarchy (Figure 1(a,b)). However, depending on prior memory access patterns, memory requests from the same warp can be serviced from different levels in the memory hierarchy (Figure 1(c,d)). In this work, we identify how these different access patterns impact GPU side-channel attacks.

Another factor that impacts GPU performance is the cache line granularity. It is known that the cache line size in modern GPUs is 128B [10]. Unlike traditional latency-sensitive CPU cache, the memory hierarchy in GPUs needs to provide high throughput; thus, modern GPUs often employ a sectored cache [45] where the cache line size is 128B but the data is brought into the cache in "sectors" (i.e., 32B granularity) to prevent data overfetch and improve DRAM bandwidth utilization. It is interesting to note that for the Kepler generation of NVidia GPUs, the implementation differed. While it is believed to have sectored cache, some implementation of the Kepler architecture had L1 caching of global loads disabled (i.e., K20 [39]). Other implementation of the Kepler architecture (i.e., K40) which had L1 cache enabled, L1 cache line granularity was 128B – "Kepler serviced loads at a granularity of 128B when L1 caching of global loads was enabled and 32B otherwise ... [with all subsequent NVidia GPUs] data access unit is 32B regardless of whether global loads are cached in L1 or not" [40].

B. AES Encryption in GPUs

Advanced Encryption Standard (AES) [9] is a symmetric cipher that is widely used to efficiently encrypt a large amount of data. AES input is divided into fixed blocks size (i.e., 128-bits) with key sizes of 128, 192, or 256-bits. In this work, we consider only the 128-bits version of Electronic Code Book (ECB) mode AES but this work is applicable to longer key sizes. AES-128 algorithm consists of 10 rounds

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

with each round consisting of lookup tables for fast encryption performance [14] – the earlier rounds use 4 lookup tables (T0-T3 tables) and the last round involves a single lookup table (T4 table), followed by a bitwise XOR with the last round key. The last round of AES is considered in this work (similar to prior work [21], [23], [24]) since it includes a single T-Table (T4) lookup and is invertible, while the earlier rounds do not have such characteristics. AES encryption on GPU proceeds in parallel across the multiple threads within a warp. The plaintext (i.e 512 bytes) received by the GPU is divided across the threads within a warp and each thread performs encryption in parallel. In this work, we use the AES implementation that compiled with the default optimization flag (-O3) in the GPU ($nvcc$ CUDA compiler). We also target recovery of the last round key bytes [21], [23], [24].

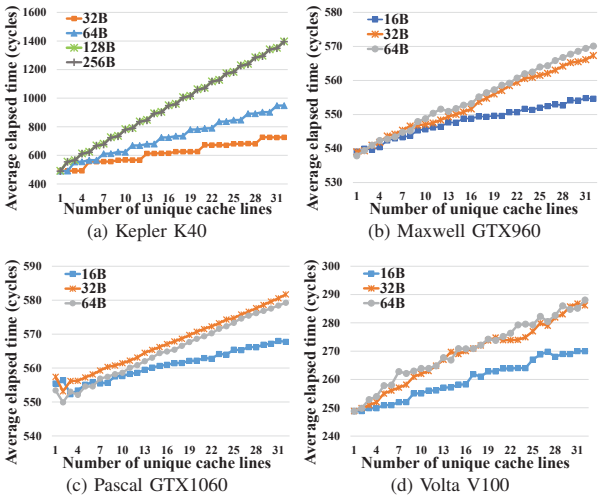
C. SIMT Leakage Side-channel Attack

In SIMT architectures such as GPUs, read requests from a warp are combined or coalesced to maximize memory bandwidth. Coalesced requests are processed one per cycle by *memory issue serialization* or *writeback serialization* [27]. Thus, the number of unique cache line requests generated by a warp and execution time are positively correlated, which we refer to as *SIMT leakage* [21]. In the last round of AES, each lookup table index can be calculated inversely from one guessed value of the key byte and the corresponding ciphertext byte [21], [23], [24]. Thus, the attacker can determine the number of unique cache line [2] requests under a key byte guess. Only the correct key byte guess would yield the highest correlation between the (calculated) number of unique cache lines and the observed timing. Pearson Correlation Value [42] is used such that with a correct guess of the N^{th} key byte, the number of unique cache line requests from the N^{th} load will result in the highest correlation with the execution time. However, for all the incorrect guesses, correlation will not exist since the calculated number of unique cache lines will differ from the actual number of unique cache lines. Therefore, the candidate key byte with the highest correlation value from all possible candidate keys represents the correct key byte that can be obtained from the side-channel attack. More details of the attack are presented in [21], [23], [24]. In this work, we demonstrate how the same correlation-based timing attack is no longer applicable to modern GPUs and provide analysis on the challenges in modern GPUs that prevent similar correlation-based timing attacks.

D. Threat Model

The attack setup is similar to prior work [21], [23], [24], where a victim GPU server provides AES encryption service, and a single attacker continuously sends 512B plaintext to the GPU for encryption and observes the output ciphertext. With the timing information and the ciphertext, the goal is to recover the 16-byte last-round key of the AES. We assume the attacker does not have any root privilege in the target system.

[2]For sectored-cache, the number of unique *sectors* can be determined, not necessarily the number of cache lines.



(a) Kepler K40 (b) Maxwell GTX960
(c) Pascal GTX1060 (d) Volta V100

Fig. 2: Timing measurements as the number of unique cache lines is varied with different strides across different GPUs.

The attacker sends 512B of plaintext to the victim server, and the server encrypts the received plaintext using the GPU. The encrypted ciphertext is then sent back to the attacker, and the attacker acquires timing information during this process. Similar to prior work [21], we use both *clean* and *noisy measurement* where clean measurement is defined as obtaining timing data directly from GPU kernel computation while noisy measurement refers to obtaining timing data from the host and includes CPU noise. Our attacks were conducted on Kepler, Maxwell, Pascal, and Volta GPUs. This work targets a particular application (AES) but the proposed side-channel attack captures timing-dependent behavior of an application on its secret-related data accesses and is applicable to other table-based ciphers such as triple-DES, sliding-window-based RSA [35], [36], and other software that access private/secret data. Identifying and exploiting such dependencies requires investigation of specific applications and we leave that as part of future work.

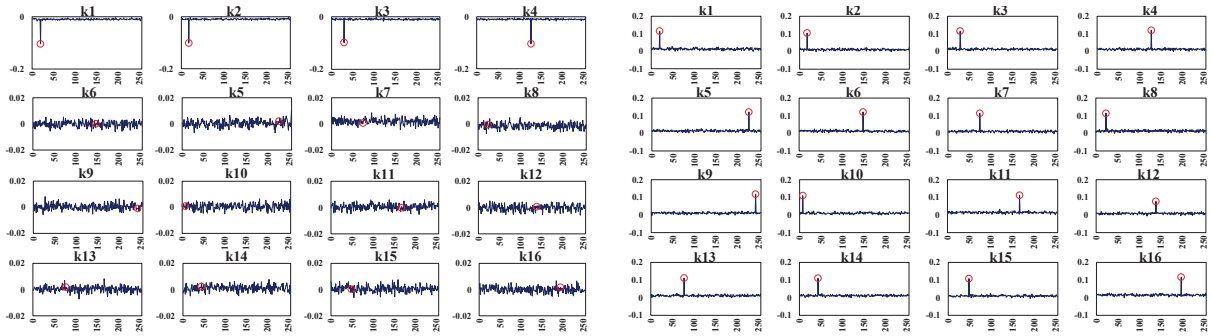
III. Impact of GPU μarch on SIMT Leakage

In this section, we demonstrate how SIMT leakage exists across different GPU architectures. Previously proposed GPU side-channel attack based on SIMT leakage [21] succeeds in some architecture; however, differences in recent GPU microarchitectures, in particular sectored cache microarchitecture, impact SIMT leakage.

A. SIMT Leakage Re-visited

We first evaluate SIMT leakage across 4 different GPU architectures, including Nvidia Tesla K40 (Kepler), GeForce GTX960 (Maxwell), GTX1060 (Pascal) and Tesla V100 (Volta). A simple CUDA kernel is written to measure the time to access the memory as the number of unique cache lines in a warp is varied [21] and the results are shown in Figure 2. Across 4 different generations of GPU architectures, SIMT leakage exists with positive correlation – i.e., increase

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.



(a) L1 enabled (b) L1 bypass

Fig. 3: AES key recovery attack results on V100 (Volta) NVIDIA GPU with (a) L1 enabled and (b) L1 bypassed. Key recovery fails when L1 is enabled but succeeds when L1 is not enabled or bypassed. Note that L1 bypass negatively impacts AES performance and is not normal execution but shown to illustrate the impact of L1 cache on the correlation-based timing attack. x-axis is the possible candidate key byte values and y-axis is the Pearson Correlation value.

in the number of unique cache lines proportionally increases the latency or the average elapsed time. The slope of the line varies across different GPU architectures. However, one key difference is the Kepler GPU results show that cache lines are accessed in 128B granularity but for the other GPUs, the cache lines are accessed in 32B granularity as described in Section II-A. [3] As we later discuss in Section IV, the access granularity has a significant impact on the side-channel attack.

We also re-evaluate the previously proposed GPU side-channel attack [21] across different GPUs with 1 million samples. While we were able to duplicate the side-channel attack on the Kepler GPU, we were not able to succeed on the other GPUs, including Pascal, Maxwell, and Volta GPUs. The results from AES key recovery for the Volta GPU are shown in Figure 3(a) and results from other GPUs (Pascal and Maxwell) follow similar trends. Each plot in Figure 3(a) represents the possible candidate key byte values on the x-axis and the y-axis represents the Pearson Correlation values. Interestingly, the first four key bytes can be recovered if *negative* correlation is assumed; however, *negative* correlation does not hold for the remaining 12 key bytes. To isolate the impact of the cache hierarchy, we re-evaluated the side-channel attack but disabled global caching in L1 cache by setting the *dlcm=cg* compiler flag. The results from the AES key recovery with L1 bypassed is shown in Figure 3(b). The results clearly show that all 16 key bytes of AES last round can be recovered using the previously proposed side-channel attack [21].

One reason why disabling L1 has a significant impact is because L2 is effectively warmed-up with the T-Tables. GPUs initially copy their data from the host memory to the GPU device memory and occurs through the GPU; thus, the DMA data pass through the L2 cache when routed to the GPU

[3]Note that prior work [21] states cache line size was 64B for Kepler GPU. It is our understanding that cache line size has always been 128B in NVidia GPUs but the implementation has varied, in terms of "sectors" and promotion policy between L2 and L1. For some of the older GPUs, promotion policy could have been set by the driver at 64B granularity and prior work effectively observed 64B cache line size. However, as our analysis in Section IV shows, the difference between 64B and 128B is relatively small but when data access is 32B, it has a significant impact on the side-channel attack.

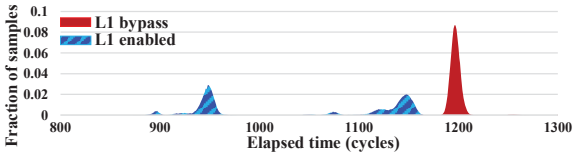


Fig. 4: Distribution of AES timing samples for the last round on Volta GPU when L1 is enabled and when L1 is bypassed.

device memory [12]. Since modern GPUs have several MBs of L2 cache and the T-tables used for AES is relatively small (approximately 5kB), the L2 cache is effectively warmed-up from the data copy from the host. As a result, initial memory accesses for the T-tables do not access the main memory but are accessed from the L2 cache – similar to what is shown in Figure 1b (`L2 Access Only`). However, when L1 caching is enabled with a sectored-cache microarchitecture, correlation attack based on SIMT leakage is no longer possible.

B. GPU Sectored Cache Impact on SIMT Leakage

Side-channel attack based on SIMT leakage is based on observing differences in memory access latency. As discussed earlier in Section III-A, when L1 is disabled or bypassed on the Volta GPU system, the side-channel attack based on SIMT leakage was successful (Figure 3(b)) since all the T4-table accesses come from the L2. However, if L1 is enabled, some data will be cached in L1 while other data will come from the L2 – thus, memory accesses will have a significant variance that is great than the SIMT leakage. The impact of this variance is shown in Figure 4 which plots 1 million measurements of the last round in AES with L1 enabled and L1 bypassed. When L1 is bypassed, a tight distribution of execution times is shown as all data for the T4-table is accessed from the L2. However, when L1 is enabled, significant performance variation is observed as some data can come from L1 while in other cases, data can come from L2. Thus, the access latency difference between L1 and L2 can be approximately 150 cycles in Volta [20] – much larger than the

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

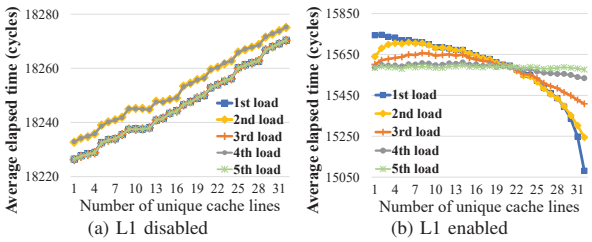


Fig. 5: Execution time of synthetic workload with 16 loads as the number of unique cache lines of each load instruction is varied. Results are measured on Volta GPU with (a) L1 disabled and (b) L1 enabled.

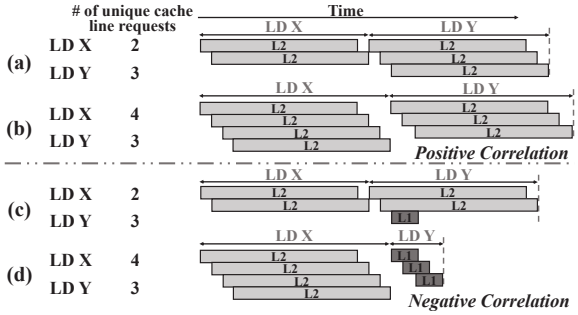


Fig. 6: A toy example to demonstrate (a,b) positive correlation and (c,d) negative correlation in GPU memory access. LD X / LD Y represent two consecutive memory accesses and L1/L2 show memory access latency to L1/L2 cache hierarchy.

SIMT leakage (Figure 2) and makes the side-channel attack infeasible.

The sectored cache and in particular, *32B data access granularity of the sectored cache has a significant impact on memory access latency since it determines the rate at which L1 cache is filled* for the last round of AES (or the T4-table). For example, with a 1024-byte T4-table and 128B data access granularity, only 8 unique accesses are needed to copy the entire T4-table from L2 to L1. But with 32B data access granularity, 32 unique accesses are needed. Given that the maximum number of unique cache lines from a single warp is 32, multiple load instructions are likely needed to copy the entire table from L2 to L1. If the entire T4-table is brought into the L1 cache, SIMT leakage is still valid since the difference in memory access latency is proportional to the number of unique cache lines; however, when access come from both L1 and L2 (i.e., Figure 1c), the difference in memory access latency for L1 and L2 has a bigger impact on execution time compared to SIMT leakage. In the following section, given the different possible memory access behavior, we analyze how *negative* correlation can occur and how we leverage it in our proposed Trident attack.

IV. Negative Correlation Leakage

In this section, we first demonstrate how negative correlation can occur through a simplified kernel evaluation and a toy example. We then provide a probabilistic model that describes how data granularity (or "sector") impacts the way T-table is loaded into the L1 cache. Based on this model, we describe how it impacts the performance of the last round of AES.

A. Overview

As shown earlier in Figure 3(a), earlier key bytes from the GPU side-channel attack resulted in a negative correlation. To better understand why negative correlation occurs, we created a synthetic CUDA kernel with 16 loads to evaluate the memory access during the last round of AES and average the execution time across 1000 samples with and without L1 caching of global loads on the Volta GPU (Figure 5). For each n^{th} load that we plot, we vary the number of unique cache lines for the n^{th} load (x-axis) while all other loads have a random number of unique cache line requests. For simplicity, only the earlier loads (i.e., $n < 6$) are shown. When L1 cache is bypassed, positive correlation from *SIMT leakage* is observed for all loads (Figure 5a). However, Figure 5b shows that ***L1 caching of global loads produces negative correlation for earlier loads and no correlation for latter loads*** – i.e. for 5^{th} load and beyond, the number of unique cache lines has negligible impact on overall execution time.

To understand how negative correlation can occur, a toy example is shown in Figure 6 for a system with consecutive two loads. Each load accesses the same data structure (e.g., T4-table) that is spread across four L2 cache lines. L1 (L2) in the figure represents latency to access L1 (L2) cache. In Figure 6(a,b), different possible numbers of unique cache lines are shown for the two loads and the example shows how positive correlation occurs – i.e., when the number of unique cache lines increases (from 2 to 4 for the first load - LD X), the execution time increases. In Figure 6(a,b), we assume L1 cache is bypassed.

However, an example of negative correlation is shown (Figure 6(c,d)) where we assume both L1 and L2 caches in the memory hierarchy are used. Negative correlation is defined as – increase in the number of unique cache lines *decreases* the overall execution time. In Figure 6(c), since only two unique cache line requests are made by the first load, the second load still results in some accesses to the L2 cache. In comparison, with more unique cache line accesses from the first load (Figure 6(d)), all of the data is loaded into L1 cache by the first load. Thus, all accesses from the second load will hit in L1 cache – and result in *faster* execution time to cause *negative* correlation. In the following section, we provide a probabilistic model to analyze the impact of cache access granularity on the negative correlation.

B. Probabilistic Model

The example shown in Figure 6 was a simplified example but we generalize the behavior by modeling the memory accesses to the T4-table during the last round of AES. The goal of the probabilistic model is to ***analyze how fast the entire T4-table is loaded into L1*** – i.e., if the T4-table is loaded into L1 at a slow rate, `L1 & L2 Access` pattern

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

can occur to cause negative correlation. [4] Assuming the 1024-byte T4-table is loaded into L2, and not in L1 (Sec II-A), and the data accesses from a warp to the T4-table are random, we provide a probabilistic model for our analysis. Without loss of generality, we define the following parameters.

w = # of unique cache lines in the T4-table
m = avg # of unique cache line requests for each load
X_N = # of unique cache lines in L1 after N^{th} load
Y_N = `L1 Access Only` event occurring during N^{th} load

The size of T4-table can be expressed as $w \times$ cache_line_size. $P(X_N = k)$ is defined as the probability that k *unique* cache lines (of the T4-table) are loaded into L1 after N^{th} load. The initial conditions ($N = 1$) of this probability are as follows:

$$P(X_1 = k) = \begin{cases} 1, & \text{if k = m} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Based on these initial conditions, we can calculate $P(X_N = k)$ using the law of total probability [8], which states that the probability of an outcome can be expressed as the sum of the probability of several distinct events. For $N \geq 2$, if we assume the number of unique cache lines loaded into the L1 cache *preceding* the N^{th} load is i (i.e., $X_{N-1} = i$), then $m \leq i \leq k$ and $(k - i)$ unique cache lines need to be loaded into the L1 cache from the L2 cache during the N^{th} load to guarantee $X_N = k$ event occurs. $P(X_N = k)$ can then be calculated by determining all possible events which is represented by all possible values of i between m and k and can be expressed as follows:

$$P(X_N = k) = \sum_{i=m}^{k} P(\{X_N = k\} \cap \{X_{N-1} = i\}) \quad (2)$$

$$= \sum_{i=m}^{k} P(X_{N-1} = i) \cdot P(X_N = k \mid X_{N-1} = i) \quad (3)$$

$$= \sum_{i=m}^{k} P(X_{N-1} = i) \cdot \frac{\binom{w-i}{k-i} \cdot \binom{i}{m-(k-i)}}{\binom{w}{m}} \quad (4)$$

Equation (2) is from the law of total probability and applying the definition of conditional probability results in Equation (3). The conditional probability in Equation (3) can be calculated by considering all different possible L1/L2 access combinations during the N^{th} load. The denominator lists all possible combination of m accesses during the N^{th} load while numerator is the product of the different combination of $k - i$ data coming from L2 and the different combination of the remaining $m - (k - i)$ data coming from L1, as shown in Equation (4). In particular, $\binom{w}{m}$ represents the total possible combination of selecting m unique cache lines from w possible cache lines of the T4-table while $\binom{w-i}{k-i}$ is the combination of requests that comes from L2 and the remaining $m-(k-i)$ coming from the i cache lines in L1 is represented as $\binom{i}{m-(k-i)}$.

[4]The model described below track actual measurements from real GPUs but due to difficulties in obtaining the necessary performance counters, the model was validated using cache simulations.

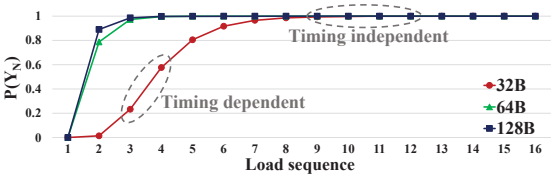


Fig. 7: `L1 Access Only` probability ($P(Y_N)$) for different cache line sizes.

The probability of `L1 Access Only` during the N^{th} load ($P(Y_N)$) can also be calculated using the law of total probability and leveraging Equation (4). If we assume $X_{N-1} = i$ or i cache lines are loaded into L1 before the N^{th} load, in order for `L1 Access Only` event to occur, the m unique cache line requests for the N^{th} load ***must*** be accessed from the i cache lines that are already in L1. All possible events (or values of i) range from m to w, or the maximum number of unique cache lines that can be loaded into the L1. When `L1 Access Only` event occurs, then no additional cache lines are brought into L1 from L2 and i cache lines remain in the L1 cache after N^{th} load – thus, $Y_N = \{X_N = i\}$ since $X_{N-1} = i$. Thus, $P(Y_N)$ can be expressed as follows:

$$P(Y_N) = \sum_{i=m}^{w} P(\{Y_N\} \cap \{X_{N-1} = i\}) \quad (5)$$

$$= \sum_{i=m}^{w} P(X_{N-1} = i) \cdot P(Y_N \mid X_{N-1} = i) \quad (6)$$

$$= \sum_{i=m}^{w} P(X_{N-1} = i) \cdot P(X_N = i \mid X_{N-1} = i) \quad (7)$$

$$= \sum_{i=m}^{w} P(X_{N-1} = i) \cdot \frac{\binom{i}{m} \cdot \binom{w-i}{0}}{\binom{w}{m}} \quad (8)$$

Equation (7) is obtained by substituting $Y_N = \{X_N = i\}$ from earlier definition. The same approach used in Equation (3), (4) can also be used to obtain Equation (8). $\binom{i}{m}$ is the number of possible accesses where the m unique cache lines come from the i cache line already loaded into L1. $P(X_{N-1} = i)$ can be calculated from Equation 4.

Figure 7 plots Equation (8) or the probability that `L1 Access Only` will occur at the N^{th} load for different data access granularity. [5] When data are accessed in 128B or 64B granularity, T4-table consists of 8 or 16 unique cache lines, respectively. Given that memory accesses are done in unit of a warp, there is a very high probability (approximately 90% or higher) that all of the loads starting with the 2^{nd} load will load data from the L1 cache. The `L2 Access Only` always occurs for the first load since the T4-table is cached in L2 but for all subsequent loads, there is a high probability of `L1 Access Only`. As a result, `L1 Access Only` or `L2`

[5]Prior to `L1 Access Only`, memory accesses will be 'L1 & L2 Access' or 'L2 Access Only', based on the memory access classification described earlier in Section II-A.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

`Access Only` lead to positive correlation (SIMT leakage) – i.e., a higher number of unique cache lines lead to higher execution time.

We classify loads that result in `L1 Access Only` or `L2 Access Only` as ***timing independent*** loads – i.e., the latency or timing of the load is independent of the other loads. The loads with cache lines of 64B or 128B result in all loads being timing independent loads; thus, SIMT leakage has a direct impact on the execution time and enables timing-based side-channel attacks. However, the behavior of 32B data access granularity differs (Figure 7). Since, for the 32B granularity case, the T4-table consists of 32 unique cache lines and is randomly accessed by a warp, the T4-table might not get loaded in the L1 cache entirely at 1^{st} and 2^{nd} load. Thus, loads such as 3^{rd} and 4^{th} loads, for 32B data access granularity, can be classified as timing *dependent* loads – i.e., timing or latency of these depend on the prior loads since the data can come from L1 and/or L2 (i.e., `L1& L2 Access`). As a result, if the earlier loads (i.e., 1^{st} or 2^{nd} loads) result in a smaller number of unique cache line requests, then the execution time will be longer since the probability of `L1 Access Only` decreases. In comparison, if the number of unique cache line requests are *higher* from the earlier loads, then the execution time will be shorter since `L1 Access Only` probability increases. As a result, the timing *dependent* loads with 32B cache line creates negative correlation, as described earlier in Figure 6, and results in the negative correlation for the earlier keys (Figure 3(a)).

C. Limitations of Correlation Timing Attack

While prior sub-section described the behavior of negative correlation, there is a limitation of this correlation-based timing attack in modern GPUs. Earlier key bytes can be recovered by negative correlation but results from Figure 3a showed that the remaining key bytes cannot be recovered using negative correlation. As shown in Figure 7, cache access granularity of 32B results in some timing dependent loads; however, for the latter loads (i.e., 5^{th} or 6^{th} load and beyond), loads become timing *independent* loads since all accesses come from L1 with high probability. Interestingly, these latter loads do not necessarily result in positive correlation but result in *no* correlation as shown by the flat line for the 5^{th} load in Figure 5. While positive correlation should occur for the 5^{th} load since `L1 Access Only` occurs, the latency of the ***earlier loads dominate overall execution time***. For the timing attacks, the timing measurement comes from the entire execution time of all 16 loads and not individual loads. The earlier, timing dependent loads have a significant impact on overall execution time since the latency difference between L1 and L2 is significant (e.g., ∼150 cycles) and dominates the latency difference from SIMT leakage – thus, positive correlation for the latter loads are effectively "hidden" and do not have a significant impact on overall performance. Thus, the number of unique cache lines does not result in a significant correlation with the overall execution time and the correlation-based timing attack is no longer applicable.

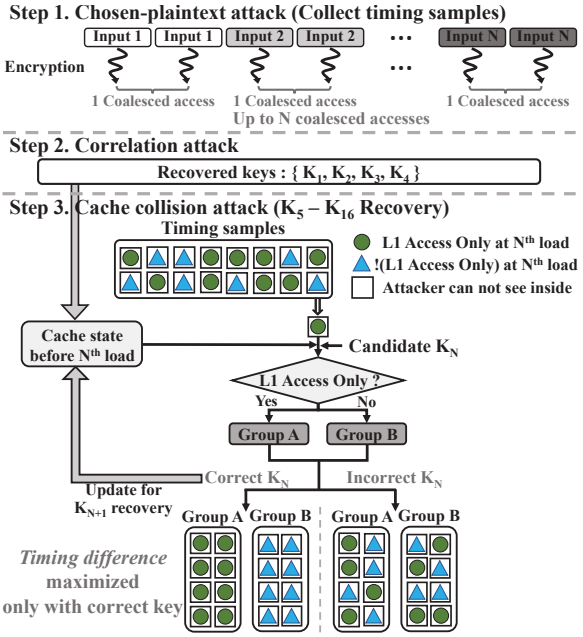


Fig. 8: Trident attack high-level overview diagram.

V. Trident Side-Channel Attack

In this section, we describe our proposed *Trident* GPU side-channel attack that enables full recovery of AES last round key on modern GPU microarchitecture with small data access granularity. In our *Trident* attack, we do not make any assumptions on the location of the data in the memory hierarchy. [6] Unlike prior work where each AES key byte can be recovered in parallel, Trident requires sequential key recovery – i.e., recover earlier key bytes and use that information for recovering latter key bytes. The first 4 key bytes are recovered in a batch while the rest of the key bytes are recovered one-by-one. The *same* timing samples are re-used for the key recovery and Trident requires a significantly smaller number of samples and results in significant speedup in recovering all key bytes. The threat model assumed is identical to what was described in Sec II-D. Figure 8 provides an overview of the Trident attack that consists of three steps. The first step is to apply the chosen-plaintext attack to reduce the number of timing samples required for our cache-collision attack by adjusting the amount of memory coalescing. This step provides the ciphertext and timing information from the victim GPU server. The second step is to recover the earlier key bytes through the negative correlation-based timing attack, and these key bytes are provided as cache state information for the GPU cache-collision attack. The final step is our novel cache-collision attack to recover the remaining key bytes, based on "difference-of-means attack" [26], exploiting the large latency difference between L1 and L2 cache in

[6] As discussed in Section III-B, by default data are pre-loaded into L2 but we also show our attack is effective when cache is flushed to begin with a cold cache in Section V-D.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

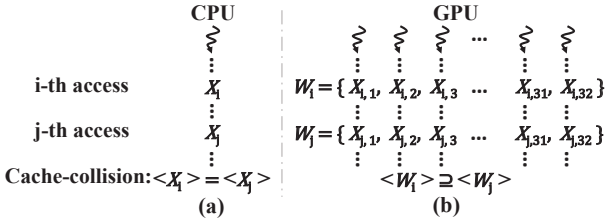


Fig. 9: Comparison of CPU and GPU cache-collision attacks.

GPUs. Unlike previously proposed CPU-based cache collision attack [6] that only recovers the "distance" between key bytes (XOR results), our proposed cache collision, as part of a hybrid Trident attack, enables the key bytes to directly recovered.

A. Motivation/Challenges in GPU Cache-Collision Attack

While cache-collision attacks have been demonstrated in CPUs [2], [5], [6], to the best of our knowledge, cache collision timing-based attacks in GPUs have not been demonstrated before. Prior CPU cache collision attacks are based on the assumption of a single thread accessing data and whether subsequent memory accesses hit or miss in the cache. An example of CPU cache collision attack is shown in Figure 9(a) – for two memory accesses (X_i and X_j), a cache collision can be determined if $\langle X_i \rangle = \langle X_j \rangle$ or access the same cache line.

However, the strict definition of cache collision in units of thread is not applicable to GPUs as it can lead to incorrect information. GPU operates in units of warp with multiple threads and even if a thread is accessing the data for the first time, a cache hit can occur based on a different thread's earlier access. In Figure 9, $X_{a,b}$ notation is used to refer to a^{th} memory access and b^{th} thread in a warp. Using this notation, if $\langle X_{i,1} \rangle = \langle X_{j,2} \rangle$, a cache "hit" occurs but the accesses come from different a thread. Thus, cache-collision detection in GPUs need to be defined in the unit of warps, and not individual threads, especially given that the number of memory requests generated can vary significantly based on the amount of coalescing. As shown in Figure 9(b), each warp generates a set of requests (i.e., W_i and W_j) that consists of multiple individual requests. In this work, we define GPU cache collision to occur if $\langle W_i \rangle \supseteq \langle W_j \rangle$ – i.e., all cache requests from W_j is a subset of requests in W_i. Unlike CPU cache collision where only $=$ condition needs to be met, GPU presents additional challenges since the number of accesses can vary and the number of timing samples needed increases exponentially.

Another challenge with cache collision attack is the relatively small size T4-tables (i.e., 1kB) and the large number of parallel accesses in GPUs. As shown earlier in Section IV-B, only a few loads or memory accesses are needed to load the entire table in the memory hierarchy and results in latter loads '`L1 Access Only`'. However, cache collision timing attacks are not only based on the difference between cache collision (i.e., $\langle X_i \rangle = \langle X_j \rangle$) as well as non-cache collision ($\langle X_i \rangle \neq \langle X_j \rangle$). With all data brought up to the first level of the memory hierarchy (i.e., L1 cache), it becomes difficult to obtain measurements or samples for "non-cache collision" for the cache collision attack. In the next sub-section, we describe how to overcome these challenges through Trident attack.

B. GPU Cache Collision Attack

In this work, we define GPU cache collision as when `L1 Access Only` occurs (Figure 1(a)) and resulting in short memory access latency while GPU cache non-collision is defined as all other access patterns (Figure 1(b,c,d)) – i.e., at least a single memory access from a thread within a warp does not hit in L1 but accesses L2, which we define as `!(L1 Access Only)`. Since access to L2 cache (and main memory) is significantly longer compared to L1 cache in GPUs, we use the timing difference between `L1 Access Only` and `!(L1 Access Only)` in the Trident attack.

With a CPU cache collision attack, the probability of generating cache collisions for an AES timing-attack is only $1/16$ (assuming 16 cache lines are needed for the T4-table with 64B cache lines in CPUs) and timing samples for cache collision and non-cache collision can be easily generated. However, the probability of generating cache collision on GPU decreases exponentially to approximately 10^{-5} (= $P(\langle W_i \rangle \supseteq \langle W_j \rangle)$), which makes it practically impossible to carry out the cache collision attack. As a result, we propose a *hybrid* timing side-channel attack where we exploit the negative correlation described earlier in Section IV to recover the earlier keys.

Assuming that 1^{st} through the N^{th} key bytes are recovered through negative correlation, we can use the recovered key bytes to determine which cache lines are accessed from the 1^{st} to the N^{th} loads – i.e., it is possible to determine which cache lines are already loaded into the L1 cache before $(N+1)^{th}$ load ($\cup_{i=1}^{N} \langle W_i \rangle$ or defined as the sum of all access from the 1^{st} load to the N^{th} load). Using this information in the GPU cache collision attack, we can determine exactly whether requests generated by the $(N+1)^{th}$ load are L1 cache hit or not ($\cup_{i=1}^{N} \langle W_i \rangle \supseteq \langle W_{N+1} \rangle$). As a result, the accessed cache lines information from the earlier key bytes significantly increase the probability of cache collision and becomes equivalent to the probability of `L1 Access Only` as shown in Figure 7.

Algorithm 1 provides an overall summary of the algorithm used to classify and calculate the difference of the execution time into two sets – `L1 Access Only` and `!(L1 Access Only)` when the attacker guesses a key byte in the 0-255 range. As shown earlier in the Figure 3a, the earlier key bytes can be recovered through negative correlation for the 1st - 4th key bytes. Using this information, for all 5th candidate key byte values ($0-255$), we can distinguish whether the requests generated by the 5th load instruction is `L1 Access Only` or `!(L1 Access Only)` based on the timing information. Trident categorizes all timing samples into two sets, `L1 Access Only` and `!(L1 Access Only)`, the correct guess of the key byte can be determined when the difference in the execution time of these two sets is the highest – and thus, correctly identifying the correct 5th key byte.

A key aspect of Trident attack is that cache-collision attack is *sequential* – i.e., the latter key bytes can be recovered one

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Calculating Difference between `L1 Access Only` and `!(L1 Access Only)`

```
1: Candi_k_i ⇐ i th key guess (0 - 255)
2: T_L1Only ⇐ 0
3: T_NotL1Only ⇐ 0
4: for k = 1 : N do % number of samples
5:     Cipher ⇐ ReadCipher(k)
6:     encrypt_time ⇐ ReadEncryptionTime(k)
7:     IsL1Hit ⇐ [ false, false, ... ]
8:     for p = 1 : i − 1 do
9:         for q = 1 : 32 do
10:            Index = T4^-1[Cipher[q][p] ⊕ Key_p]
11:            IsL1Hit[Index] ⇐ true
12:        end for
13:    end for
14:    L1Only_Flag ⇐ true
15:    for q = 1 : 32 do
16:        Index = T4^-1[Cipher[q][i] ⊕ Candi_k_i]
17:        if IsL1Hit[Index]! = true then
18:            L1Only_Flag ⇐ false
19:        end if
20:    end for
21:    if L1Only_Flag == true then
22:        T_L1Only += encrypt_time
23:    else
24:        T_NotL1Only += encrypt_time
25:    end if
26: end for
27: return Avg(T_NotL1Only) - Avg(T_L1Only)
```

at a time, based on the earlier key recovery. Thus, by using the 1st - 5th key bytes, the attacker knows the state of L1 cache (in terms of the T4-table that is loaded into L1) before the 6th load instruction and the 6th key byte can be recovered using the same approach described earlier. [7] By repeating this process, the attacker can recover the remaining key bytes by collecting sufficient timing samples of `L1 Access Only` and `!(L1 Access Only)`.

C. Chosen Plaintext Attack

One challenge in GPU cache collision attack is obtaining timing samples for both `L1 Access Only` and `!(L1 Access Only)`. As shown in Figure 11 (based on the probability model from Section IV-B), the probability of generating `!(L1 Access Only)` sample becomes very difficult for the latter loads since the entire T4-table will likely be loaded into the L1 cache (shown with the baseline data points in Figure 11). Therefore, the attacker must collect a significant amount of timing samples to obtain data for `!(L1 Access Only)`. [8] To address this challenge, we exploit memory coalescing in GPUs and exploit *chosen-plaintext attack* to

[7]Note that samples do not need to be collected sequentially but only the key recovery analysis is done sequentially.

[8]Based on probability in Figure 11, approximately 2 billion samples are needed for sufficient `!(L1 Access Only)` samples.

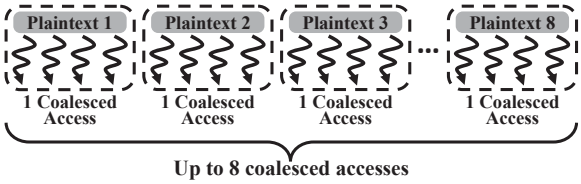


Fig. 10: 8-thread chosen-plaintext attack in *Trident*.

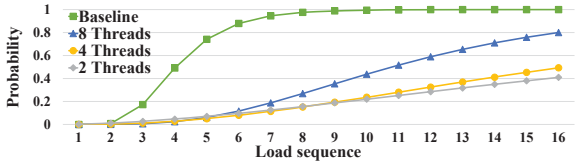


Fig. 11: Probability that `L1 Access Only` occurs for the 2, 4, 8 thread model. Baseline has 32 threads per warp.

effectively ***slow the rate of T4-table being loaded into the GPU L1 cache***. We propose to leverage chosen-plaintext attack by providing the *same* plaintext to different threads within a warp – which causes the threads to generate the same ciphertext and access the same T4-table element. As a result, T4-table read requests from different threads using the same plaintext are coalesced into one request. An example of an 8-thread model is shown in Figure 10 where group of 4 threads are provided the same plaintext – thus, effectively reducing 32-thread warp to an 8-thread warp, and resulting in up to only 8 memory accesses, instead of 32 memory accesses.

The chosen-plaintext attack enables the attacker to control the number of unique cache line requests that is generated and its impact on how fast L1 cache is loaded with the T4-table as shown in Figure 11. With the baseline (or 32 threads), the probability of `L1 Access Only` reaches approximately 100% by the 9th load – however, with an 8-thread model, the probability is significantly reduced to under 40%. With a 4-thread model, the probability is further reduced to approximately 20%. By controlling the degree of coalescing, the attacker can easily generate samples of `!(L1 Access Only)` for the latter loads.

D. Evaluation of Trident Attack

By leveraging the chosen-plaintext attack, the results from the Trident attack are shown in Figure 12. Trident attack can recover all 16 bytes of the last round key using only 100,000 samples, compared to correlation attack requiring one million samples [21]. As described earlier, the first 4 key bytes are recovered using negative correlation, and thus, the y-axis shows Pearson correlation values while for the other 12 key bytes, the y-axis shows the difference between the average time of `!(L1 Access Only)` and `L1 Access Only`. For $k5$ through $k12$, the key bytes are recovered at the lowest value while for the remaining key bytes, the key bytes are recovered at the highest value. It is interesting to note that while the y-axis shows, `!(L1 Access Only) - L1 Access Only`, the value can be less than 0 – i.e., `!(L1 Access Only)` performance is faster than `L1 Access Only`. As discussed

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

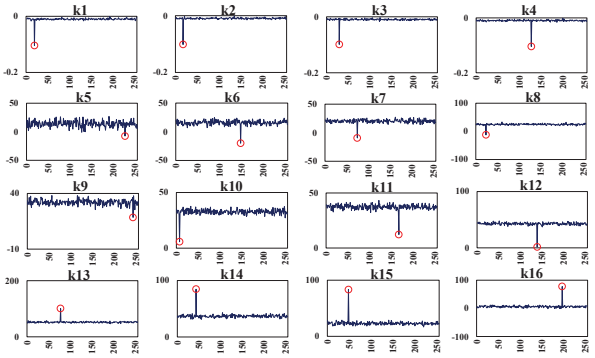


Fig. 12: AES key recovery on Volta GPU with 100,000 samples using 8 thread model under *clean* measurements.

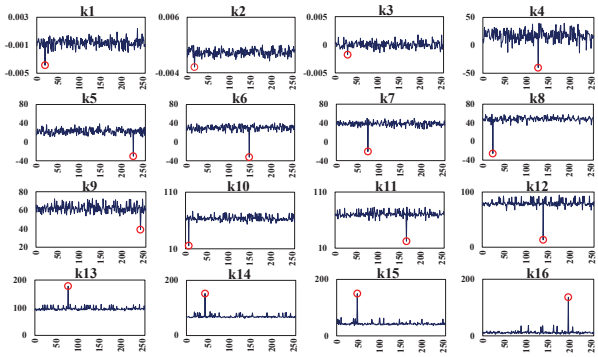


Fig. 13: AES key recovery on Volta GPU with 2 million samples using 8-thread model under *noisy* measurements.

in Section IV, negative correlation can occur which causes this non-intuitive behavior in latency. For the latter key bytes ($k9$ to $k12$), the key bytes are still recovered at the lowest point but the values are actually positive. For the last four key bytes, the impact of negative correlation disappears and the key bytes are recovered at the *highest* value. We also evaluate the Trident attack across different scenarios, including evaluations across different memory hierarchy and in a more noisy environment.

Different memory hierarchy: Since our side-channel attack exploits the latency difference across the memory hierarchy, we also evaluated a scenario where the entire cache is flushed – i.e., assume all of the data is located in the global memory (off-chip DRAM), including the T4-table. Due to space constraints, the results are not shown but we succeeded in complete key recovery using 200,000 timing samples with the 4-thread model alongside the Trident attack.

Noisy environment: The evaluations earlier assumed that GPU kernel execution time could be measured but it can be very difficult to have such GPU-only execution time measurements. To evaluate a more realistic scenario, we model a "noisy" environment, similar to prior work [21], where the execution time is measured from the host or the CPU. Thus, noise from the host to CPU transfer overheads is included. The full key recovery on the Volta GPU with the Trident attack using the 8-thread model is shown in Figure 13. More samples were required as 2 million timing samples were required to carry out the attack and the correlation values are lower than the clean measurements; however, all key bytes can still be recovered.

E. Discussion

Our baseline attack model, similar to prior work [21], [23], [24], was based on a single warp. GPU is designed to support parallelism and more warps and AES throughput can be improved with more warps; however, using more warps can be "unfavorable" to the attacker [21] (e.g., higher noise, etc.). In our Trident attack, we evaluated a 256-threads attack (8 warps) and was successful in recovery all key bytes. However, an 8-warps attack required 1 million samples or $10\times$ longer than the single-warp baseline Trident attack. In general, since the attacker controls the workload to "stress" test the GPU server, it is reasonable for the attacker to choose a "favorable" setup (i.e., minimize noise, minimize the attack time, etc.). Modern GPUs support Multiple-Process Service (MPS) [41] and side-channel attacks would become harder with MPS because of additional noise and uncertainty; however, an attacker will likely avoid MPS to minimize noise in their attack and make the environment more "favorable".

VI. TridentShield Countermeasure

In this section, we propose a countermeasure to the proposed Trident attack. To address the correlation-based timing attack [21], different solutions have been recently proposed including RCoal [23] and BCoal [24]. The limitations of RCoal have been addressed in [24] so we focus on comparison with BCoal in this section. BCoal [24] removes leakage from coalescing by creating *constant* amount of coalescing. However, the performance impact from BCoal is additional on-chip memory bandwidth from the additional requests generated. In particular, the performance impact from BCoal is higher in modern GPU system with 32B sectors because of the additional requests that are generated. The performance degradation increases when there are multiple warps executing because of the extra bandwidth demands. In this section, we propose TridentShield – a countermeasure that does not impact on-chip bandwidth or coalescing but exploits the difference in L1 and L2 cache access latency to prevent correlation-based timing attacks.

A. Latency-based Countermeasure

Prior defense [23], [24] removed correlation by restricting memory coalescing; however, memory coalescing is a critical component to maximize bandwidth in GPUs [25]. In this work, we observe that since leakage appears in terms of latency (or execution time), *leakage can be removed if the memory access latency correlation to the number of unique cache lines is removed by ensuring memory access latency for the T-table is made constant*. Given the relatively small size of the T-tables, we propose to initially prefetch the entire T-tables into the L1 and lock the content of the T-tables in L1 such that it is not evicted. As demonstrated in prior work [21], this creates

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

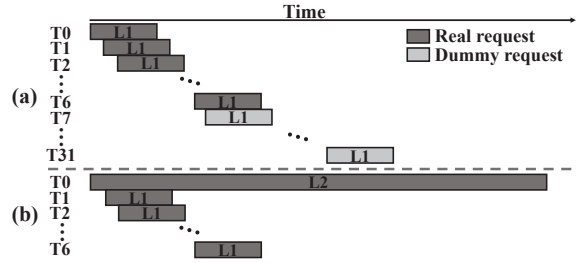


Fig. 14: Comparison of (a) BCoal and (b) TridentShield countermeasures. L1/L2 represents L1/L2 cache latency.

SIMT leakage (or positive correlation) based on the number of unique cache lines.

To remove this correlation, we propose that one memory read request within each warp that access the T-tables bypass L1 and access the data from L2. [9] Since L2 latency is significantly higher than L1 latency, the overall execution time of the load instruction is dominated by the L2 latency and the information leakage from L1 is effectively "hidden." A high-level overview comparing BCoal and TridentShield is shown in Figure 14. We assume that 7 unique cache lines are generated. With BCoal (Figure 14(a)), 25 additional dummy requests are generated (assuming BCoal(32) is implemented). In TridentShield(Figure 14(b)), only the "real" requests are sent but the first request bypasses L1 and the request is sent to L2 – thus, the positive correlation from L1 requests is effectively hidden.

However, this approach can degrade performance since the benefit of low latency L1 is lost and the latency of the warp is determined by the slowest thread (or the L2 access). To overcome this limitation while ensuring correlation is not leaked, we *probabilistically* determine whether a memory read request in a warp bypasses L1 and accesses L2 – i.e., a single request from each warp is sent to L2 with a probability of $P_{L2}(N)$ where N is the number of unique cache line requests. $P_{L2}(N)$ is adjusted based on N such that, regardless of the value of N, the average execution time for the memory access is constant. As N increases, $P_{L2}(N)$ decreases since the total L1 access latency increase (green line in Figure 15). If $P_{L2}(N) = 1$, a request is always sent to L2 (blue line in Figure 15). By probabilistically determining the number of requests sent to L2 (and bypass L1), on average, the overall latency matches the latency (shown with the red dotted line) – as if 32 unique cache lines were generated for L1. $P_{L2}(N)$ can be determined using the following parameters.

M = maximum # of unique cache line requests
G = slope of `L1 Access Only`
$L1$ = latency of L1 cache access
$L2$ = latency of L2 cache access

$$L1 + G \cdot M = P_{L2}(N) \cdot L2 + (1 - P_{L2}(N)) \cdot (L1 + G \cdot N)$$
$$\therefore \; P_{L2}(N) = \frac{G \cdot (M - N)}{(L2 - L1) - G \cdot N} \tag{9}$$

[9]We assume L1 is write-through and L2 is inclusive of L1 in GPUs.

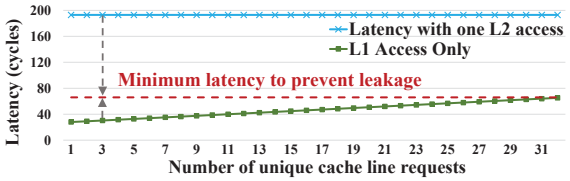


Fig. 15: Probabilistically determined L2 access rate.

TABLE I: GPGPU-Sim configurations used for simulations.

	Simulation Parameters
Resources/Core	1400MHz core clock, SIMT width = 32 (16x2) 1 SM, 48KB Shared Memory, 32KB Register File 16KB 4-way L1D Cache (32B sector size) 2KB 4-way L1I Cache, 128B cache block size
Interconnect	Crossbar, 1400MHz
Memory Subsystem	128KB L2 Cache per bank, 2 L2 banks per MC 6 MCs, 924MHz memory clock

To verify the proposed countermeasure prevents correlation-based timing attack, we use a similar analytical model as prior work [23], [24] where the number of timing samples needed to recover the entire AES keys can be determined by the following:

$$S \propto \left(\frac{\mu(U \times T) - \mu(U) \cdot \mu(T)}{\sigma(U) \cdot \sigma(T)}\right)^{-2}$$

where U is the vector of the number of memory requests and T is the vector of latency, μ and σ are the mean and standard deviation, respectively. Since $\mu(T|U = u) = L1 + G \cdot M$ for any u with proposed countermeasure, $\mu(U \times T)$ can be calculated as follows:

$$\begin{aligned}\mu(U \times T) &= \textstyle\sum_u P(u) \cdot \mu(U \times T|U = u) \\ &= (\text{L1 + G} \cdot \text{M}) \cdot \textstyle\sum_u u \cdot P(u) \\ &= (\text{L1 + G} \cdot \text{M}) \cdot \mu(U) = \mu(T) \cdot \mu(U) \\ \therefore \;\; S &= (0)^{-2} = \infty\end{aligned}$$

As a result, the required number of plaintext samples to break AES approaches infinity – i.e., timing attack based on correlation is no longer possible. In TridentShield, the main microarchitectural components/overheads are locking the tables in the cache and cache bypassing. For cache locking, additional lock bits are needed in the L1 cache line state to ensure the cache lines are not evicted. To support bypassing, a random number generator is needed to probabilistically determine which request to bypass L1. Both G and $P_{L2}(N)$ can be calculated ahead of time, based on the GPU architecture, thus, can be stored in a dedicated table. Since only 32 entries are required, only 128B of extra storage is required. N is determined during run-time but existing memory coalescing logic can be used to determine the number of unique cache line requests (e.g., "pending request count" within the memory coalescing logic [29]).

B. TridentShield Evaluations

To evaluate the performance impact from the proposed countermeasure, GPGPU-Sim [3] was used to implement both BCoal [24] and TridentShield. The configuration used in our evaluation is summarized in Table I. One key difference in

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

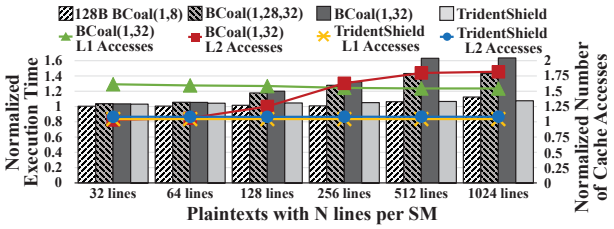


Fig. 16: Performance of BCoal and TridentShield for different plaintext sizes per SM. All results are normalized to the baseline GPU. (Each line means that 16 bytes plaintext)

our evaluation compared to prior work [24] is the cache line assumption – we assume 32B sectored-cache while BCoal [24] assumed 128B cache line (data access granularity) and the results are shown in Figure 16. Results are normalized to a baseline GPU without any support to limit correlation-based timing attacks and the number of lines or plaintext that is being encrypted is varied. As the number of lines increases, GPU does more work and more on-chip bandwidth is being consumed. BCoal(1,8), assuming 128B cache lines, has a small impact on performance, as performance degrades by only 11%. However, with 32B data accesses, BCoal(1,32) performance degradation is also negligible for small input data but as the input size increases, BCoal results in up to 50-75% additional accesses to L1/L2 as compared to the baseline. Thus, performance degradation exceeds 60%. A slightly improved implementation (BCoal(1,28,32)) reduces performance degradation but still results in approximately 43% performance degradation. In comparison, our proposed solution results in no bandwidth overhead for L1 and less than 10% overhead for L2 accesses. Thus, the impact as the number of lines increases is minimal and the overall performance degradation is under 10%. The implementation includes the overhead of the prefetching for the T-tables and we assume Equation 9 is used to determine the L2 access rate. As a result, our proposed latency-based TridentShield countermeasure provides a cost-effective countermeasure against correlation-based timing attacks in GPUs with sectored-cache.

VII. Related work

Different types of side-channel attacks have been proposed to target AES and other cryptosystems, including electromagnetic radiation [7], [33], power consumption [26], [32], timing [4], [48], and temperature [17]. Timing side-channel attacks are an attractive approach for side-channel attacks since they can be carried out without physical access to the system. Timing side-channel attack is based on timing variance that occurs in memory access across the on-chip memory hierarchy and many attacks have been proposed for CPU [2], [5], [6], [18]. Cache-based timing attack [4] assumes that the attacker can observe the aggregate profile of the encryption time. Because memory access latency is dependent on the cache, a cache hit or miss can be inferred based on the total encryption time. Bonneau et al. [6] used timing variation effect from cache-collisions in the sequence of lookups performed by the encryption. Key recovery using a wide collision that indicates that the five S-boxes process pairwisely equal values for a plaintext pair have also been proposed [5]. Most prior works were done on CPUs but for GPUs, all threads in warp or wavefront are executed in parallel and prior approaches are not directly applicable to GPUs.

There have been recent studies on security vulnerability in GPUs [28], [37], [46], [49] and a recent study [38] show that side-channel attacks on GPUs can be very realistic concerns. There have also been prior works on the AES side-channel attack/countermeasures using GPUs [21], [23], [24], [34], [35]. Prior work [13] demonstrated the GPU side-channel using electromagnetic behavior of the GPU. In addition to GPU side-channel attack [21] described in Section II-C, a similar side-channel attack on GPU has been proposed to exploit bank conflicts in shared memory [22]. Their attack demonstrated that several key bytes can be recovered even with 8192-thread execution. However, since they exploit latency difference from bank conflicts, it requires as much as 10 million samples for full key recovery – in comparison, Trident attack is able to achieve complete key recovery with only 100,000 samples.

VIII. Conclusion

In this work, we demonstrated how prior side-channel attacks on GPUs are not valid on modern GPUs because of sectored-cache that results in data from being accessed from both the L1 and the L2 cache in the memory hierarchy. Thus, the positive correlation between execution time and the number of unique cache lines being access does not exist. In comparison, we identified how the opposite or *negative* correlation can occur. We leveraged this leakage in our proposed *Trident* attack – a hybrid correlation-collision attack for GPUs. The negative correlation is used to identify the earlier AES key bytes while the cache collision attack is used to extract the latter key bytes where memory coalescing is exploited with chosen-plaintext attack to enable cache collision attack. We also proposed *TridentShield* – a latency-based countermeasure that minimizes bandwidth degradation compared to prior work while preventing the Trident side-channel attack.

Acknowledgment

We would like to thank the anonymous reviewers and the shepherd for their insightful comments, as well as Jaemoon Sim for his contributions to initial GPU evaluations. This work was supported in part by NRF-2020R1A2B5B0100168711, NRF-2020M3H6A1085498, IITP-20200013090012002, and NSF/SRC STARSS-1618379.

References

[1] A. A. Abdelrahman, M. M. Fouad, H. Dahshan, and A. M. Mousa, "High Performance CUDA AES Implementation: A Quantitative Performance Analysis Approach," in *2017 Computing Conference*, July 2017, pp. 1077–1085.

[2] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Cache Based Remote Timing Attack on the AES," in *Cryptographers' track at the RSA conference*, 2007, pp. 271–286.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.

[3] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 163–174.

[4] D. J. Bernstein, "Cache-timing attacks on AES," 2005.

[5] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke, "Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs," in *Cryptographers' Track at the RSA Conference*, 2010, pp. 235–251.

[6] J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks Against AES," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2006, pp. 201–215.

[7] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 163–177.

[8] K. L. Chung and K. Zhong, *A Course in Probability Theory*. Academic Press, 1974.

[9] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[10] Design Guide, "CUDA C Programming guide," NVIDIA, July 2013.

[11] P. G. Emma, R. K. Montoye, and V. Srinivasan, "Sectored cache memory," Apr. 28 2009, US Patent 7,526,610.

[12] R. Farber, *CUDA Application Design and Development*. Elsevier, 2011.

[13] Y. Gao, W. Cheng, H. Zhang, and Y. Zhou, "Cache-Collision Attacks on GPU-Based AES Implementation with Electro-Magnetic Leakages," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 300–306.

[14] J. Gong, W. Liu, and H. Zhang, "Multiple Lookup Table-Based AES Encryption Algorithm Implementation," in *International Conference on Solid State Devices and Materials Science*, 2012, pp. 842–847.

[15] S. Gueron, "Intel Advanced Encryption Standard(AES) New Instructions Set," Mobility Group, Israel Development Center, Intel Corporation, Tech. Rep., May 2010.

[16] O. Harrison and J. Waldron, "AES Encryption Implementation and Analysis on Commodity Graphics Processing Units," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2007, pp. 209–226.

[17] M. Hutter and J.-M. Schmidt, "The Temperature Side Channel and Heating Fault Attacks," in *International Conference on Smart Card Research and Advanced Applications*, 2013, pp. 219–235.

[18] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing – and Its Application to AES," in *2015 IEEE Symposium on Security and Privacy (SP)*, May 2015, pp. 591–604.

[19] K. Iwai, N. Nishikawa, and T. Kurokawa, "Acceleration of AES encryption on CUDA GPU," *International Journal of Networking and Computing*, vol. 2, no. 1, pp. 131–145, 2012.

[20] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[21] Z. H. Jiang, Y. Fei, and D. Kaeli, "A Complete Key Recovery Timing Attack on a GPU," in *2016 IEEE International symposium on high performance computer architecture (HPCA)*, 2016, pp. 394–405.

[22] ——, "A Novel Side-Channel Timing Attack on GPUs," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, p. 167–172.

[23] G. Kadam, D. Zhang, and A. Jog, "RCoal: Mitigating GPU Timing Attack via Subwarp-based Randomized Coalescing Techniques," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 156–167.

[24] ——, "BCoal: Bucketing-based Memory Coalescing for Efficient and Secure GPUs," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 570–581.

[25] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[26] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Annual International Cryptology Conference*, 1999, pp. 388–397.

[27] A. Lashgar, E. Salehi, and A. Baniasadi, "Understanding Outstanding Memory Request Handling Resources in GPGPUs," in *proceedings of The Sixth International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART'2015)*, 2015.

[28] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing Webpages Rendered on Your Browser by Exploiting GPU Vulnerabilities," in *2014 IEEE Symposium on Security and Privacy (SP)*, May 2014, pp. 19–33.

[29] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, p. 487–498.

[30] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and Analysis of AES Encryption on GPU," in *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, June 2012, pp. 843–848.

[31] J. S. Liptay, "Structural aspects of the System/360 Model 85, II: The cache," *IBM Systems Journal*, vol. 7, no. 1, pp. 15–21, 1968.

[32] O. Lo, W. J. Buchanan, and D. Carson, "Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA)," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 88–107, 2017.

[33] J. Longo, E. De Mulder, D. Page, and M. Tunstall, "SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2015, pp. 620–640.

[34] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli, "Side-Channel Power Analysis of a GPU AES Implementation," in *33rd IEEE International Conference on Computer Design (ICCD)*, Oct 2015, pp. 281–288.

[35] C. Luo, Y. Fei, and D. Kaeli, "GPU Acceleration of RSA is Vulnerable to Side-channel Timing Attacks," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[36] ——, "Side-channel Timing Attack of RSA on a GPU," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 3, pp. 1–18, 2019.

[37] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and Characterizing Covert Channels on GPGPUs," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*, 2017, pp. 354–366.

[38] H. Naghibijouybari, A. Neupane, Z. Qian, and N. B. Abu-Ghazaleh, "Rendered Insecure: GPU Side Channel Attacks are Practical," in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[39] "Kepler Tuning Guide," https://docs.nvidia.com/pdf/Kepler_Tuning_Guide.pdf, NVIDIA.

[40] "Pascal Tuning Guide (PDF) - v10.1.168," https://docs.nvidia.com/cuda/pascal-tuning-guide/index.html, NVIDIA, 2019.

[41] NVIDIA, NVML, "Multi-Process Service," NVIDIA, June 2020.

[42] K. Pearson, "Notes on Regression and Inheritance in the Case of Two Parents Proceedings of the Royal Society of London, 58, 240-242," 1895.

[43] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*, 2013, pp. 86–98.

[44] A. Seznec, "Decoupled Sectored Caches: conciliating low tag implementation cost and low miss ratio," in *Proceedings of 21st International Symposium on Computer Architecture (ISCA)*, 1994, pp. 384–393.

[45] G. Thomas-Collignon and P. Micikevicius, "VOLTA Architecture and performance optimization," http://on-demand.gputechconf.com/gtc/2018/presentation/s81006-volta-architecture-and-performance-optimization.pdf, NVIDIA, 2018.

[46] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted Execution Environments on GPUs," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.

[47] C. Wang and X. Chu, "GPU Accelerated AES Algorithm," *arXiv preprint arXiv:1902.05234*, 2019.

[48] M. Weiß, B. Heinz, and F. Stumpf, "A Cache Timing Attack on AES in Virtualization Environments," in *Financial Cryptography and Data Security*, February 2012, pp. 314–328.

[49] Z. Zhu, S. Kim, Y. Rozhanski, Y. Hu, E. Witchel, and M. Silberstein, "Understanding The Security of Discrete GPUs," in *Proceedings of the General Purpose GPUs (GPGPU-10)*, 2017, pp. 1–11.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:57:07 UTC from IEEE Xplore. Restrictions apply.