Ghost Arbitration: Mitigating Interconnect Side-Channel Timing Attacks in GPU

Zhixian Jin
KAIST
Daejeon, Republic of Korea
kimjihyun@kaist.ac.kr

Jaeguk Ahn
KAIST
Daejeon, Republic of Korea
jaegukahn@kaist.ac.kr

Jiho Kim
KAIST
Daejeon, Republic of Korea
jihokim@kaist.ac.kr

Hans Kasan
KAIST
Daejeon, Republic of Korea
hanskasan@kaist.ac.kr

Jina Song
KAIST
Daejeon, Republic of Korea
s_jina@kaist.ac.kr

Wonjun Song
Kangwon National University
Chuncheon, Republic of Korea
wjsong@kangwon.ac.kr

John Kim
KAIST
Daejeon, Republic of Korea
jjk12@kaist.ac.kr

2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 979-8-3503-5057-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MICRO61859.2024.00086

***Abstract*—Network-on-chip (NoC) is a critical shared resource in scalable multicore processors; however, it is well-known that shared resources can lead to side-channel attacks. In this work, we demonstrate how contention for on-chip bandwidth in GPUs can lead to *fine*-grain information leakage and enable side-channel attacks. As a case study, we demonstrate how RSA key bit information can be leaked on a real GPU. We also describe how interconnect characteristics from the side-channel or an *interconnect-gram* can be used to fingerprint kernels executing on the GPU. To defend against such fine-grain side-channel attack, we propose *secure* arbitration that prevents information leakage while minimizing performance impact during normal execution. In particular, we present a novel *ghost arbitration* that prevents interconnect contention from being leveraged to leak information by keeping track of "ghost" requests or requests when other nodes receive free arbitration to enable least-recently-used priority. However, if the attacker reverse engineers the arbitration, a naïve implementation of ghost arbitration can still lead to information leakage. Thus, we propose a *weighted* ghost arbitration that exploits "malicious" communication patterns to prevent information leakage with minimal loss in performance. Compared to previously proposed arbitration that is secure (e.g., strict time-division multiplexing), ghost arbitration is able to improve performance by up to 4×.**

***Index Terms*—GPU, side-channel attack, network-on-chip**

I. Introduction

Microarchitectural side channel attacks occur when an attacker exploits unintended information leakage by leveraging the microarchitecture features in a processor [2]. Common examples include the cache-based side channel [11], [15], [17], [25], [32], [34], [38], [45], [56], memory system and DRAM [27], as well as core microarchitecture such as branch prediction [1], [13], [28] and speculative execution [31]. Most prior work on microarchitectural side-channel has focused either on the compute or the memory component in computing; however, another core component of any computing system is the communication or interconnection networks [9]. By definition and design, an interconnection network is fundamentally a *shared* resource between two or more components [9]. Recent work [8], [44] have demonstrated side-channel for network-on-chip (NoC) in multi-core CPUs; however, to the best of our knowledge, side-channel attacks on GPU NoC have not been demonstrated before. In this work, we demonstrate how a timing side-channel attack based on interconnect contention can be carried out on a GPU.

GPUs have been widely utilized because of their high compute throughput and high memory bandwidth [43]. Recently, GPUs have enabled multi-programming where multiple programs (and users) can execute concurrently on the same GPU to efficiently utilize the GPU resources [41]. However, concurrent execution introduces security concerns, especially in terms of side-channel or covert-channel timing attacks. In particular, different GPU microarchitectural-based side-channel and covert-channel attacks have been demonstrated [3], [12], [57]. Prior work [3] identified how network-on-chip within GPUs can be leveraged for high-bandwidth covert channels. The shared on-chip interconnect (and the hierarchical organization) was exploited to enable covert channels; however, prior work did not demonstrate how timing side-channel attacks using network-on-chip interconnect. In addition, while *secure* arbitration was proposed to prevent covert channels, the performance degradation can be significant since arbitration effectively reduces the bandwidth through a time-division multiplexing approach.

In this work, we first show how shared on-chip interconnect in GPUs can be leveraged for side-channel attacks by observing the victim's L2 accesses. In particular, the contention for on-chip bandwidth can be leveraged to identify the amount of L2 (or memory) accesses and be used to leak RSA key information. Unlike other side-channel attacks, the GPU interconnect enables *fine*-grain information to be obtained during the side-channel attack. We also describe how an *interconnect-gram* of the L2 accesses can be recovered – a visual representation of L2 accesses, including intensity and frequency as well as latency – to infer information about the victim. Based on the interconnect-gram, we demonstrate an application fingerprinting attack that allows the attacker to infer the application that is concurrently executing on the same GPU. By using the fingerprinting attack, we show how a highly accurate attack can be carried out across

979-8-3503-5057-9/24/$31.00 ©2024 IEEE
DOI 10.1109/MICRO61859.2024.00086

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

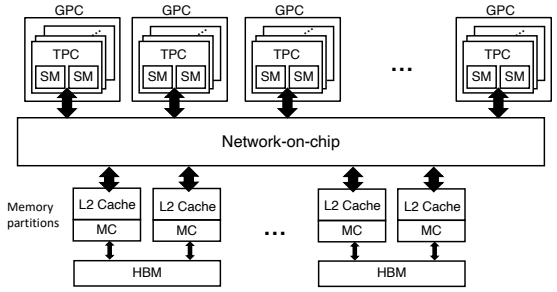


Fig. 1: High-level block diagram of GPU with network-on-chip between the cores and the memory partitions.

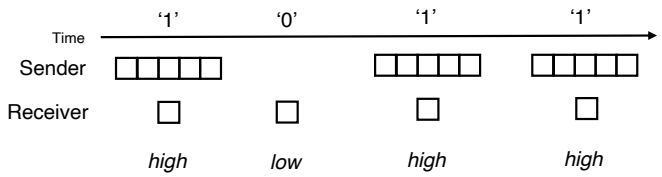


Fig. 2: High-level diagram of interconnect contention-based covert channel where each box represents requests (or packets) that access the interconnect. Contention during '1' communication results in high latency while for a '0', low latency can be observed by the receiver and result in information leakage.

various deep-learning neural networks to identify (or extract) the models.

To avoid such information leakage, we propose a novel *ghost* arbitration that ensures information leakage based on interconnect contention can be minimized. A naïve implementation of ghost arbitration is effectively least-recently-used (LRU) arbitration. While such implementation can avoid information leakage, an attacker can exploit their knowledge of the arbitration algorithm to re-establish side-channel (or covert-channel) with lower bandwidth. To address this challenge and ensure that the secure arbitration can scale to more than two nodes, we propose *weighted* ghost arbitration where the weights (or ghost counters) are varied dynamically to exploit the "malicious" communication patterns. We identify the malicious pattern as when the sender needs to transmit more requests compared to the receiver to ensure contention occurs and when such sequence occurs, the weights are varied dynamically to ensure that the receiver is prioritized to achieve 'low' latency. We provide an analysis on the impact of the ghost counters needed for weighted ghost arbitration and show that given a sufficient size of counters, weighted ghost arbitration prevents any information leakage. In particular, the main contributions of this work include the following.

- We demonstrate how an interconnect contention-based side-channel attack can be carried out on a GPU to leak information about the secret key in RSA.
- We also show how *fine*-grain information can be leveraged to create an *interconnect-gram* of victim L2 accesses and can be used for fingerprinting attacks to extract the DNN models.
- We propose a novel *ghost* arbitration to enable secure arbitration by effectively identifying malicious communication patterns and prevent timing side-channel attacks based on network-on-chip contention.
- We demonstrate that performance loss from ghost arbitration is negligible on normal workloads while exceeding the performance of prior work by over 4×.

II. Background

A. Threat Model

Side-channel attacks consist of a spy and a victim, where the spy extracts essential information from the victim by analyzing unintentional information leaks that reveal the victim program's characteristics [29]. Similar to prior work [35], we assume the spy and the victim are colocated in the same GPU and run as two separate applications. Both applications execute concurrently on the same GPU using Multi-Process Service (MPS) [41]. However, unlike prior work, we assume that the attacker does not have access to the performance counters. Our evaluation of the side-channel attack is conducted on the Amazon EC2 P3 instance, with an NVIDIA V100, and we also include fingerprinting attacks evaluated on the NVIDIA H100. For the side-channel attack, we target an RSA program with a 2048-bit key, where the amount of computation differs based on whether the secret key bit sequence contains a '0' or a '1'. We also evaluate a scenario where the victim executes deep neural network (DNN) or transformer-based inference workloads concurrently with the spy kernel. The spy fingerprints the victim's application and we demonstrate how the spy can identify which model the victim is currently executing.

B. Network-on-Chip (NoC)

Network-on-Chip (NoC) [10] is used to interconnect multiple on-chip components together. While NoC is used within modern GPUs, details of GPU NoC are not publicly available. Real GPUs are known to have hierarchical organization among the cores [30], [39] – thus, it is reasonable to assume the GPU NoC is organized hierarchically based on core or stream multiprocessor (SM) organization. One key difference of GPU NoC is the need to provide high bandwidth (and throughput) as well as interconnect *speedup* [21]. As the large number of cores in a GPU share the on-chip bandwidth to the memory partitions, arbitration is necessary to allocate the interconnect bandwidth among the cores. In this work, we explore the impact of such sharing on side-channel attacks. In modern GPUs, multiple SMs (often two SMs) form a Texture Processing Cluster (TPC), and multiple TPCs further form a Graphics Processing Cluster (GPC) (Figure 1). The channel contention within the TPC (or GPC) provides an opportunity for an attacker to establish a side-channel by monitoring the shared interconnect channel [3].

C. Interconnect Timing-based Covert Channels

Prior work [3] proposed how covert channel can be created with interconnect contention and demonstrated high bandwidth ($>$ 24 Mbps) by exploiting the parallelism available in a GPU.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 RSA algorithm

```
1: for each bit b do                          ▷ in secret key
2:     square();
3:     reduction();
4:     if b == 1 then
5:         multiply();
6:         reduction();
```

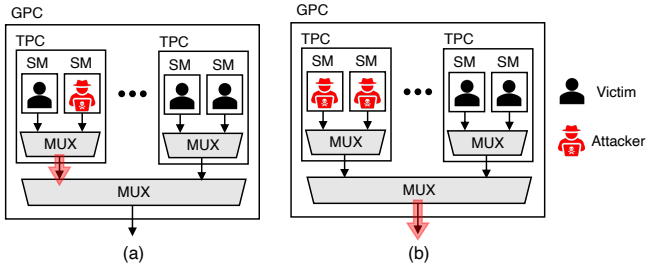


Fig. 3: High-level block diagram of (a) TPC-channel and (b) GPC-channel interconnect-based side-channel attacks with the highlighted physical channel exploited.

A high-level example of an interconnect-based covert channel is shown in Figure 2 with a sender node communicating '0' or '1' to a receiver node. For simplicity, we show the requests from each node as a single box with the sender transmitting multiple requests for a '1' to ensure contention with the receiver. The receiver constantly sends a request to measure its latency. When communicating a '1', the requests from the sender create contention for the interconnect bandwidth, and an increase in L2 latency is observed by the receiver. When communicating a '0', the sender does not send any request and thus, no contention for the interconnect resource occurs – resulting in "low" latency. The fundamental reason for such covert channels is that i) interconnect bandwidth is a shared resource and ii) arbitration commonly used attempt to provide "fairness" that can be exploited for covert channels. Prior work [3] proposed strict round-robin arbitration to prevent covert channels by exploiting time-division multiplexing of interconnect bandwidth; however, the performance degradation can be significant, especially as the number of nodes participating in the arbitration increases. In this work, we first demonstrate how the microarchitectural timing channel can be leveraged for a side-channel attack. We then propose a novel ghost arbitration that minimizes performance degradation while maintaining secure arbitration

III. Side Channel Attack

In this section, we demonstrate how on-chip interconnect contention in GPUs can be leveraged for side-channel attacks by identifying L2 cache access events to enable side-channel attacks. Using RSA as an example, we show how fine-grain information can be obtained to infer the number of 1's in the secret key. We also demonstrate how the interconnect contention characteristics can be exploited to create an *interconnect-gram* in fingerprinting applications.

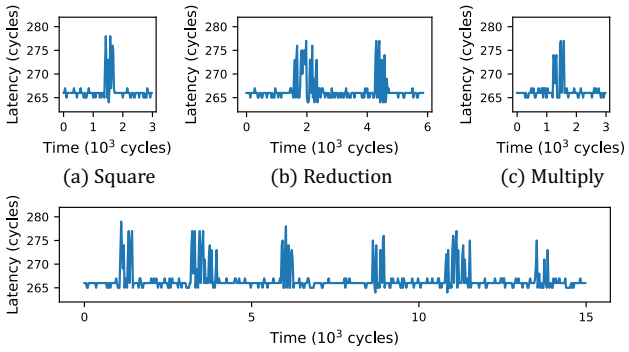


Fig. 4: Latency sample collected by the spy for different kernels within the RSA decryption including (a) `square()`, (b) `reduction()`, (c) `multiply()`, and (d) a sequence of samples for bit '1'.

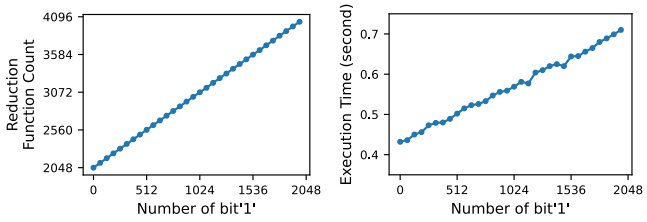
A. Spy-NoC Side-Channel Attack

With the attacker (or the spy) and the victim executing concurrently on the same GPU, interconnect contention can be leveraged for *Spy-NoC* or side-channel attack in modern GPUs. Similar to the interconnect-based covert channel in GPUs [3], the spy can leverage either a *TPC channel* or a *GPC channel* for the side-channel attack within a single GPU, as shown in Figure 3. To utilize TPC-channel, the spy must be co-located within the same TPC as the SM executing the victim workload while for a GPC-channel, the spy must be co-located within the same GPC, on a different TPC.

In our side-channel attack, we use the RSA algorithm as a case study to demonstrate the side-channel attack. A high-level overview of the RSA algorithm is summarized in Algorithm 1. Based on the sequence of bits in each key (whether they are 0's or 1's), the amount of computation varies. For all bits, there is some constant amount of computation performed (e.g., `square()` and `reduction()`) but if the bit sequence contains a '1', additional computation is required (e.g., `multiply()` and `reduction()`) – resulting in the computation time for a bit '1' being approximately twice as long as a bit '0'. The additional computation leads to extra L2 accesses and this work exploits the additional L2 access traffic (or the network-on-chip interconnect contention) in the side-channel attack. While this work uses RSA, other work [8], [44] have demonstrated other side-channel attacks for different algorithms that result in different amount of computation for bit '0' and bit '1'.

The spy begins by allocating device memory using the `cudaMalloc()`. Throughout the spy kernel execution, the program repeatedly generates requests to a cache line of 128B data in the L2 cache and measures the L2 round-trip latency. All threads within the warp generate requests that are contiguous to ensure that the requests are fully coalesced to minimize self-interference (or interference caused by the spy itself). In addition, the spy kernel generates a request (or a new sample) every 300 cycles to ensure that its own request does not create

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.



(a) (b)

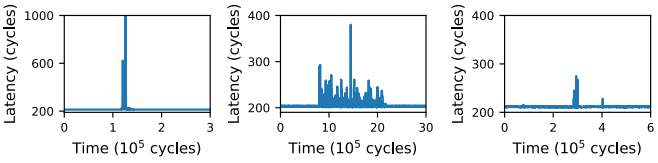
Fig. 5: (a) Number of times `Reduction()` kernel executed, measured through the interconnect side-channel on the V100 GPU, and (b) execution time measured with CPU clock as the number of bit '1' increases in the RSA key.

contention with its previous request. [1] The RSA decryption kernel was executed across 4 SMs and TPC channel side-channel attack was used by the spy kernel and Multi-Process Service (MPS) [41] was used to co-locate the kernels.

The L2 access patterns, observed by the spy kernel when co-located with the RSA kernel, are shown in Figure 4. When there is no contention with the RSA kernel, L2 roundtrip latency of approximately 265 cycles is observed but when there is interconnect contention, the latency observed increases to approximately 275 cycles. The L2 access frequency (or intensity) is similar for the `square()` and `multiply()` kernels (Figure 4(a,c)) with relatively narrow "pulse" but L2 intensity increases for `reduction()` kernel (Figure 4(b)). In our evaluation, `reduction()` pulse consisted of approximately 30 to 40 samples, while the `multiply()` or `square()` kernel pulses only consisted of 15 to 20 samples. The actual width of the pulse can vary based on the RSA implementation and/or the GPU architecture – however, the attack relies on the difference between the two kernels. These differences enable the attacker to distinguish the `reduction()` from the `square()` or `multiply()` kernels by comparing the length of the regions where continuous L2 access occurs. A continuous sequence of data obtained by the spy kernel for a bit '1' during the side-channel attack is shown in Figure 4(d) Among six regions of continuous L2 access identified in the selected sample, two are longer than the others, indicating that there are two reduction functions in total.

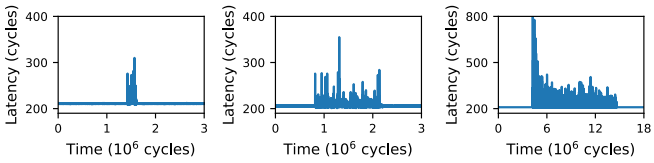
Results in Figure 5(a) show the number of times that `reduction()` kernel was executed determined based on the side-channel attack as the number of 1's in the secret key is varied and the results show a perfectly linear relationship. For comparison purposes, we also measure the execution time of RSA in Figure 5(b). While the linear relationship still holds, there is some noise but more importantly, the slope (or the signal-to-noise ratio) is significantly lower, making it more difficult to accurately determine the number of 1's based on execution time only. Leaking the number of 1's greatly reduces the search space to find the correct secret key [50]; however, the entire actual sequence of the key cannot be leaked since only the amount of 1's is obtained. For example given the sequence

[1] In the GPU that we used for our evaluation, the zero-load L2 latency was approximately 260–270 cycles.



(a) (b) (c)

Fig. 6: Example interconnect-gram of different workloads including (a) matrix-vector multiplication, (b) matrix-matrix multiplication, and (c) MLP.



(a) (b) (c)

Fig. 7: Impact of different input size ((a) 1024×1024, (b) 2048×2048, (c) 4096×4096) on the interconnect-gram for the matrix multiplication workload.

shown earlier in Figure 4(d), the attacker cannot differentiate whether the samples represent two '0's or a single '1.' However, the RSA implementation used in this work is not necessarily an optimized implementation as the amount of computation added for bit '1' is the same amount of computation as all bits. With more optimized implementation [18] of RSA on GPUs, it remains to be seen if the actual sequence of 0's and 1's can be leaked if the amount of computation for 0's and 1's differs. In addition, differences in computation for bit '0' and bit '1' in the sequence of the secret key exists in other fast (yet insecure) cryptographic implementations and similar side-channel attack can be used across other kernels as well.

B. *Interconnect-gram Fingerprinting*

Memorygram [12], [47] has been proposed for fingerprinting applications by collecting information about cache hits/misses across the different cache sets through time. Similar to memorygram, we propose *interconnect-gram* that represents L2 access "events." Interconnect-gram is a visual representation of L2 cache/interconnect activities from the spy kernel repeatedly measuring its own L2 latency over time. The spy kernel's L2 latency includes the interconnect contention latency caused by the victim's L2 accesses but in addition, interconnect-gram also contains the impact from cache contention since if the data for the spy is evicted from the cache, significantly higher latency will be observed.

One of the key differences with fingerprinting attack (compared to the side-channel attack described in the previous section) is that the working set for the workloads being fingerprinted is often bigger than the on-chip cache – thus, resulting in frequent off-chip memory accesses and cache contention between the workload and the spy kernel. As a result, to minimize interference between consecutive requests

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

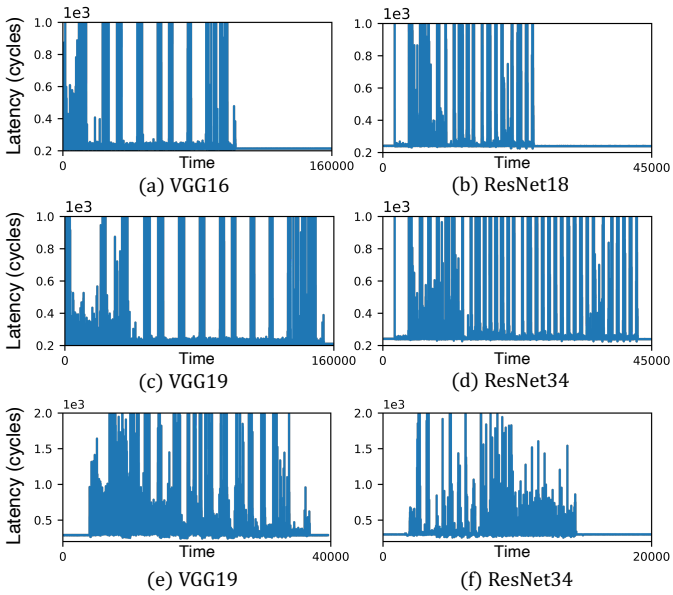


Fig. 8: Interconnect-gram collected from different DNN workloads from (a)-(d) V100 and (e)-(f) H100 GPUs.

from the spy kernel, a barrier was added to ensure that only one request from the spy kernel was in-flight.

The interconnect-gram for several different kernels that are commonly used within different DNN workloads are shown in Figure 6. The interconnect-gram was measured on the V100 by running the spy kernel on one SM and DNN workload concurrently on the remaining SMs using MPS (multi-process service). Because of the difference in their L2 access pattern (and intensity), each kernel has a different interconnect-gram representation. In addition, as the input size of the kernels changes, their interconnect-gram also correspondingly changes; in Figure 7, as the matrix input size is increased, the interconnect-gram for the matrix multiplication kernel also changes. Using the same interconnect-gram, we fingerprint different DNNs as shown in Figure 8. The interconnect-gram allows the attacker to identify not only different DNN models but can also identify variations of the same model – for example, while VGG is different from ResNet, VGG16 and VGG19 share similarities but the interconnect-gram is able to differentiate between the two. An interconnect-gram on a different GPU (i.e., H100) is also shown in Figure 8. For the same workload, the interconnect-gram changes across the different GPU architectures because of the different microarchitecture.

To demonstrate the potential fingerprinting attack, ResNet image classifier was used to identify the different DNN workloads based on the interconnect-gram images. The model was trained across 15 different DNN models from PyTorch and two LLMs from HuggingFace, using 400 training samples (or interconnect-grams) for each DNNs, with each sample consisting of 10,000 data points (or L2 latency samples from the spy kernel). The accuracy of the model was tested using 20 new samples from each of the 17 workloads. The confusion

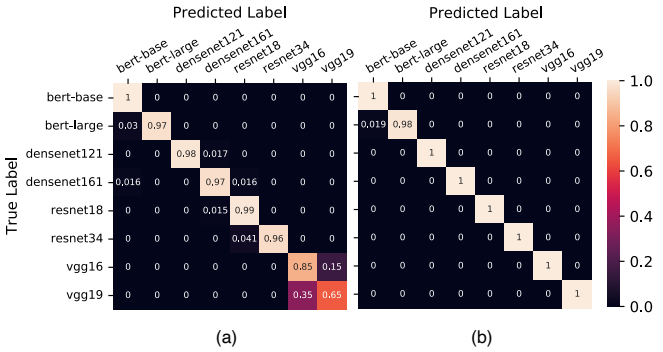


Fig. 9: Confusion matrix of the classifier trained with (a) L2 access information only and (b) L2 latency information.

matrix of the classification results is shown in Figure 9 where two different types of interconnect-grams were used – one using only the L2 access information and another one using the L2 latency. [2] The L2 latency interconnect-gram is similar to the ones shown in Figure 8 while the L2 *access information* interconnect-gram shows only L2 hit events. The L2 access was used to understand the impact L2 *only* information on the potential leakge from the side-channel attack. When using only L2 access information, an accuracy of 93.3% is achieved but when the more detailed L2 latency information is used, an accuracy of 99.8% can be achieved. Thus, while the L2 access information by itself can provide information for fingerprinting, a more detailed L2 latency can be exploited for a more accurate fingerprinting attack. Given the *fine*-grain information that can be obtained from interconnect-gram, our proposed side-channel attack has very high accuracy and we expect to achieve high accuracy across different workloads as well.

IV. Ghost Arbitration

In this section, we describe our proposed ghost arbitration that prevents interconnect-based side-channel attacks through *secure* arbitration. Prior work [3] proposed secure arbitration through a "strict" round-robin arbitration or a time-division multiplexing of the interconnect bandwidth. However, when the number of nodes being arbitrated increases, the performance degradation becomes larger. To address the performance degradation, we propose ghost arbitration – a secure arbitration with minimal impact on performance. The key observation that we exploit in this work is that the ***sender needs to generate* more *requests compared to the receiver to ensure contention and communicate (or leak) information.*** We leverage this observation to propose a class of arbitration referred to as *ghost arbitration*. The baseline ghost arbitration prioritizes the least-recently-used (LRU) nodes based on *ghost* requests. While simple, the arbitration is limited if the attacker is aware of the arbitration. We extend ghost arbitration with a *contention-triggered* ghost arbitration to provide secure arbitration across

[2]For simplicity, only 8 out the 17 workloads are shown but high accuracy was achieved on the other workloads, including AlexNet, EfficientNet, GoogLeNet, InceptionV3, MnasNet, MobileNetV2, ResNet50, ShuffleNet and SqueezeNet.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

different bit sequences. While this approach is effective for 2:1 arbitration, it can be problematic in terms of security for N:1 arbitration when $N > 2$. We then describe a *weighted* arbitration that adjusts the weights such that ghost arbitration is scalable across a large number of nodes.

A. Preliminary

In this work, we classify interconnect arbitration requests into the following three categories.

- **real request**: New requests to the arbiter made by the nodes. This is the traditional definition of requests in interconnection networks [9].
- **ghost request**: If a node does not have a request (i.e., no packet needs access to the channel resource), the node is defined to generate *ghost* requests when one or more other nodes have real requests.
- **zombie request**: When a real request goes through arbitration but is not granted, the request to the arbiter will be made again; however, these requests are defined as *zombie* requests since we do not account for these requests in modifying the counters used in the ghost arbitration.

Based on the definition of requests, we define *real* arbitration as when there are two or more real (or zombie) requests and *ghost* arbitration as when there is only one real request (or one zombie request). When there are no real or zombie requests, no arbitration is necessary and thus, no real or ghost arbitration would occur and there would also be no ghost requests as well. To enable ghost arbitration, a *ghost counter* is maintained per node and is used for arbitration in ghost arbitration – i.e., a node with a higher ghost counter value is prioritized over a node with a lower ghost counter. In the remainder of this section, we propose different implementations of ghost arbitration.

Threat model: We assume that the attacker can reverse engineer the on-chip interconnect arbitration algorithms and then, exploit the knowledge to create an input sequence pattern to create side channel information leakage.

B. Baseline Ghost Arbitration

Interconnect contention can be accurately leveraged when the sender has more requests (or packets) compared to the receiver. As a result, we exploit this behavior to prioritize LRU node or nodes that have used a smaller amount of channel bandwidth. In our proposed baseline ghost arbitration, a ghost counter is maintained per node, which counts the number of ghost requests, as shown in Figure 10. The ghost counter effectively represents the number of times other nodes received "free" grants when the node itself did not have any request – thus, preventing communicating a '1' by *prioritizing* the request from the receiver. [3] The ghost counter for a node is incremented if it has a ghost request and decremented when a node receives a "free" grant or is prioritized by having more ghost counts.

[3]The proposed arbitration effectively counts the number of times arbitration is not required (or free grants) and uses this information to bias the arbitration whenever there is contention. This enables two-node arbitration but does not generalize to n-node arbitration when only a subset of nodes participate in the arbitration to be secure.

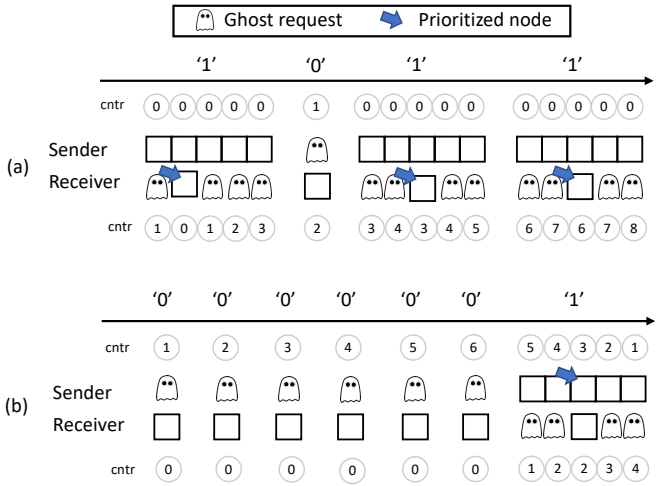


Fig. 10: (a) High-level overview of baseline ghost arbitration with ghost requests and ghost counter (`cntr`) and (b) a bit sequence that the attacker can leverage to attack the baseline ghost arbitration and prioritize the sender. `cntr` is updated *after* the arbitration.

When the ghost counter reaches its maximum value based on the counter size, the ghost counter saturates.

An example of the ghost arbitration for a two node system is shown in Figure 10(a) where the receiver sends a request at a constant rate while the sender transmits a sequence of '1011' and the corresponding ghost counter values of each node are shown. During real arbitration, the node with the higher ghost counter value is prioritized. Because of the ghost requests that are accumulated by the receiver (i.e., higher ghost counter value), when trying to communicate a '1', the receiver is prioritized and does not experience contention – thus, results in "low" latency similar to when a '0' is communicated – and prevent covert channel from being established. If the ghost counter value is identical or all the ghost counter values are 0's, no priority is given to any node and the default arbitration (e.g., round-robin arbitration) is used.

While the ghost arbitration described earlier can protect against cases where there is a mix of 1's and 0's being communicated, it can be problematic if the attacker is aware of the arbitration policy and exploits it. An example of one such exploit is shown in Figure 10(b). If the sender sends a sufficient amount of '0's or ghost requests through the arbitration, the sender can accumulate enough ghost requests through the ghost counter and ensure that the ghost arbitration is no longer secure. For example, with a sequence of '0000001', the sender maintains a higher ghost counter and when a '1' is communicated, the priority remains with the sender. Thus, the receiver request is delayed and 'high' latency can be observed to communicate a '1'. For covert channels, the extra 0's communicated to overcome the ghost arbitration reduces the covert channel bandwidth – however, given the significantly high bandwidth demonstrated for GPU covert channels [3], the loss of bandwidth still results in a high bandwidth covert channel.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

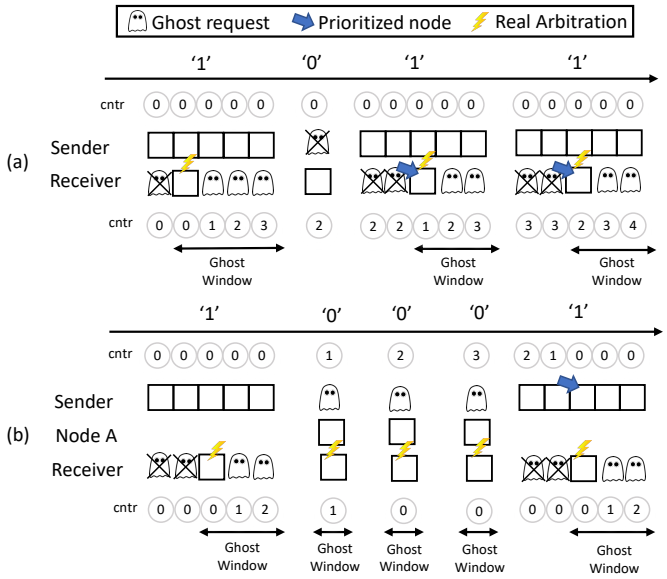


Fig. 11: (a) Illustration of contention-triggered ghost arbitration where a ghost window is used to accumulate ghost requests and (b) an example of scalability challenges when there are more than two nodes.

C. Contention-triggered Ghost Arbitration

To address the limitations of the baseline ghost arbitration, a *contention-triggered* ghost arbitration is proposed where ghost requests are only considered after contention (or real arbitration). The key aspect of contention-triggered ghost arbitration is to prevent the sender from accumulating ghost counter values by introducing a *ghost window after* real arbitration. Ghost window is terminated when there are no more real requests. During the ghost window, ghost requests are counted using ghost counters, similar to the baseline ghost arbitration; however, when the ghost window is terminated, ghost requests are no longer accumulated – i.e., ghost requests outside of the ghost window are ignored. Intuitively, the sender sends more requests than the receiver but this occurs only during the communication of a '1'. As a result, the ghost window ensures that ghost requests are only counted during the communication of a '1' and the ghost window is terminated when there are no more real requests.

An example of contention-triggered ghost arbitration is shown in Figure 11(a). After the first real arbitration [4] a ghost window is established and the receiver node begins to accumulate ghost requests. However, the ghost window disappears before the communication of the next bit, and some of the ghost requests are not counted (or ignored) as shown by the 'X' mark over ghost requests that are not within the ghost window. More importantly, the ghost request for communicating a '0' is not accumulated by the sender and thus, the sender cannot accumulate ghost counter values and prevents any information leakage from the interconnect contention. As a result, the contention-triggered ghost arbitration prevents the attacker from accumulating ghost counters and ensures that *ghost requests are only measured for the receiver*.

However, while contention-triggered ghost arbitration works for two-input arbitration, it is not necessarily scalable when the number of nodes increases beyond two. An example of scalability limitation is shown in Figure 11(b) when there are 3 nodes participating in the arbitration. If the 3rd node (i.e. `Node A`) cooperates with the attacker and injects additional requests to the arbiter to create contention with the receiver, "real" contention will occur. [5] The sender can then begin to accumulate ghost counters and the sender will be able to obtain priority during the next real contention of a '1' – thus, preventing ghost arbitration. In the following section, we describe how *weighted* ghost arbitration can enable secure arbitration where all ghost requests are not created equal but have different weights.

D. Weighted Ghost Arbitration

An ideal, *secure* arbitration with minimal performance impact on "normal" execution is a priority-based arbiter where the receiver is always prioritized. Using a priority arbiter, any information leakage from on-chip interconnect contention can be avoided as constant latency is provided – i.e., regardless of whether a '1' (contention) or a '0' (no contention) is being communicated, the receiver is always prioritized and no 'high' latency is observed. The challenge of such an approach is properly determining which node is the receiver to ensure that the receiver is always prioritized (or conversely, determining which node is the sender and reduce the sender's priority). Unlike other hardware resource contention (e.g., cache) which occurs spatially, interconnect contention needs to occur *temporally* as the request for the interconnect resource between the sender and receiver needs to occur at the same time. However, since the arbitration timing is not guaranteed in terms of contention, the sender needs to send *more* requests compared to the receiver to ensure contention occurs. [6]

In this work, we leverage such "malicious" behavior of the sender generating more requests than the receiver in our proposed weighted ghost arbitration. The key aspect is that all requests are *not* treated equally, as was the case for the ghost arbitration implementations described earlier. In particular, since the sender needs to generate more (consecutive) requests, consecutive ghost requests are weighted exponentially higher than the previous ghost request – thus, ensuring that the sender that is transmitting more requests is effectively "penalized" or the receiver is prioritized in the ghost arbitration.

A high-level overview of the proposed weighted arbitration is shown in Figure 12. Similar to the baseline ghost arbitration,

[4] With contention-triggered ghost arbitration, the communication of the *first* '1' can be communicated since the ghost counters are all 0's and arbitration would default to round-robin arbitration; however, given the long sequence of bits communicated for covert or side-channel attacks, the extra '1' in the beginning is not problematic.

[5] For simplicity, the ghost counter for `node A` is not shown.

[6] Prior work [8], [44] also proposed an interconnect contention-based side-channel attack; however, priority-based arbitration was used in the multi-core system and thus, contention occurred based on "scheduling" and not necessarily based on hardware arbitration.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Weighted Ghost Arbitration Algorithm

Input: $req[\,]$: $[real, ghost, zombie]$ requests
Output: $gnt[\,]$: grants

1: $cnt[\,]$: ghost counter
2: $pre[\,]$: previous request type
3: N : # of nodes
4: **procedure** WEIGHTED GHOST ARBITRATION
5: **for** $i \leftarrow 1$ **to** N **do**
6: $gnt[i] \leftarrow 0$
7: $i_w \leftarrow \texttt{arg max}(cnt)$ ▷ *select winner*
8: $gnt[i_w] \leftarrow 1$
9: **for** $i \leftarrow 1$ **to** N **do** ▷ *update counts*
10: **if** $req[i] = real$ **then**
11: **if** $pre[i] = real$ or $zombie$ **then**
12: $cnt[i] \gg 1$
13: **else**
14: $cnt[i] \leftarrow cnt[i] - 1$
15: **else if** $pre[i] = ghost$ **then**
16: **if** $req[i] = ghost$ **then**
17: $cnt[i] \ll 1$
18: **else**
19: $cnt[i] \leftarrow cnt[i] + 1$

the ghost counters are incremented for ghost requests, as shown for the sender when transmitting a sequence of '0's. In addition, the ghost counters are decremented for a *real* request when the sender transmits a '1'. However, the key difference compared to the baseline ghost arbitration is that the change in the ghost counter differs for consecutive real (and ghost) requests. For the baseline ghost arbitration, the counters were incremented (or decremented) by a value of 1; however, the consecutive requests are weighted higher. For the initial request from the sender when transmitting a '1', the counters are incremented (decremented) for the ghost (real) requests by one. However, the subsequent requests are weighted differently – in this example, the increase (or the decrease) of the counter changes by a factor of 2 for each consecutive requests. In our implementation of the weighted ghost arbitration, we use a shift operation to increment (or decrement) the ghost counters. Thus, the ghost counter for the sender (even if accumulated through a sequence of '0's) will reduce quickly while the ghost counter for the receiver will increase quickly – thus, ensuring that the receiver has a higher ghost counter value and be prioritized.

A block diagram of the weighted ghost arbiter is shown in Figure 13. The main components added in the arbiter are the ghost counters which is proportional to the number of inputs as well as a comparator to determine the winner. A summary of the weighted ghost arbitration is described in Algorithm 2. Similar to any other arbiter, the inputs are the requests from the N nodes and the grants are returned to the N nodes, with only one output asserted. In addition, the request type is also an input – i.e., `real`, `ghost` or `zombie` request. The grant is asserted to the input that has the highest ghost counter value (Line 7) – if there are multiple inputs with



Fig. 12: High-level overview of weighted ghost arbitration.

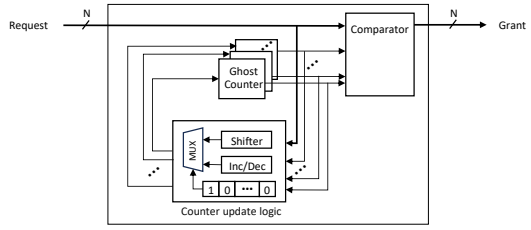


Fig. 13: Block diagram of the weighted ghost arbiter.

the same value, baseline arbitration is used to determine the winner. Afterwards, the ghost counter for each node needs to be updated. For `real` requests (Line 10), the counters are decremented by one if they are granted and if the requests are consecutive, the values are decremented by $2\times$. For the `ghost` requests(Line 15), the ghost counters are incremented by 1 unless the ghost requests are consecutive requests where the ghost counters are incremented by $2\times$.

V. EVALUATION

A. Methodology

Using GPGPU-Sim simulator [26], we model a Volta-like GPU configuration, as summarized in Table II. To evaluate the performance impact of ghost arbitration, the arbitration component of Booksim [37] network simulator within GPGPU-Sim was modified to model ghost arbitration. A hierarchical crossbar was implemented within the Booksim simulator and ghost arbitration was used within each level of the hierarchy. A synthetic workload evaluation was used to determine the impact of the different interconnect arbitration algorithms on the interconnect contention-based side channel. In addition, CUDA workloads from Polybench [14], CUDA-SDK [42], ISPASS2009 [5] and Rodinia 3.0 [6] were selected to evaluate the performance impact of the different arbitration schemes.

We compare the 3 different ghost arbitration proposed in this work (baseline, contention-triggered, and weighted) as well as compare against prior work [3] that proposed secure arbitration through strict round-robin (SRR) arbitration. SRR is a time-division multiplexing approach where the channel bandwidth is partitioned equally among the nodes. We also compare against coarse-grain round-robin (CRR) where round-robin arbitration is done at a *coarse* granularity of warps – thus, each warp or collection of requests within a warp is granted usage of the channel bandwidth at once.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Workloads used in the performance evaluation.

Description	Abbr.	Benchmark
Backpropagation	bp	Rodinia [6]
Breadth First Search	bfs	GPGPU-sim [5]
Neural Network	nn	GPGPU-sim [5]
Matrix Transpose and Vector Multiplication	atax	Polybench [14]
Matrix Vector Product and Transpose	mvt	Polybench [14]
Correlation Computation	corr	Polybench [14]
2D Convolution	conv	Polybench [14]
Mergesort	mrg	CUDA-SDK [42]
Vector Addition	vec	CUDA-SDK [42]
Discrete Cosine Transform	dct	CUDA-SDK [42]
Eigenvalue	eigen	CUDA-SDK [42]
Image Convolution	sp	CUDA-SDK [42]

TABLE II: Simulation configuration parameters.

	Simulation Parameters
Core Features	1200MHz, SIMT width=32, 40 TPCs, 2 SMs per TPC
Caches	128KB L1/Shmem per SM, 48 L2 slices, 96KB per L2 slice
Memory Model	24 MCs, HBM2, $t_{CL} = 12$, $t_{RP} = 12$, $t_{RC} = 40$, $t_{RAS} = 28$, $t_{RCD} = 12$, $t_{RRD} = 3$
Interconnect	$flit_size = 40$, $num_vcs = 1$, $subnet = 2$, 1200MHz, Crossbar

B. Impact on Information Leakage

Evaluation of information leakage when communicating a bit sequence of '010101...' is shown in Figure 14. The evaluations were done using GPGPU-sim with a synthetic kernel where both the sender and the receiver was sending requests to the L2 that resulted in a L2 hit. For round-robin arbitration (Figure 14(a)), there is clear difference in latency between a '0' (or low latency) and '1' (or high-latency). For coarse round-robin arbitration (Figure 14(b)), the receiver is sometimes prioritized but that is not always the case and communication information can be mostly recovered. In comparison, the ghost arbitration is able to prioritize the receiver to ensure constant latency is observed to prevent any information leakage. Results for SRR [3] is not shown but the results of SRR is also able to achieve constant latency, regardless of the bit sequence. However, as we show in the following subsections, there can be significant performance degradation on normal workload execution.

In Figure 15, we compare a different sequence of bits where more '0's are sent for TPC-channel (1 sender and 1 receiver) and GPC-channel (6 sender and 1 receiver) attacks. The baseline ghost arbitration is not able to prevent information leakage as the accumulated zeros results in '1' being communicated with high latency (i.e., the receiver is not properly prioritized). However, contention-triggered ghost arbitration provides secure arbitration for TPC-channel but when there are more than 2 nodes participating, information leakage occurs (i.e., GPC-channel attack). In comparison, the weighted ghost arbitration is able to provide secure arbitration, regardless of the number of nodes participating in the arbitration.

An illustration of how the ghost counter changes is shown in Figure 16 for a GPC-channel attack for the weighted ghost

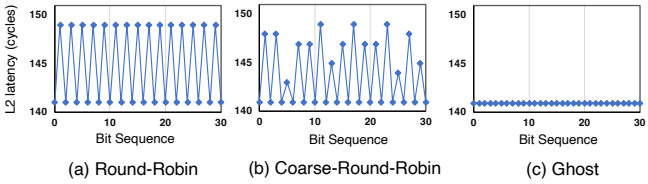


Fig. 14: L2 access latency observed by the receiver when a sender communicates a sequence of '01010..' for (a) round-robin, (b) coarse-grain round-robin, and (b) ghost arbitration. Only one core (SM) is used for the sender and the receiver. All variations of the ghost arbitration achieve the same latency for this sequence.

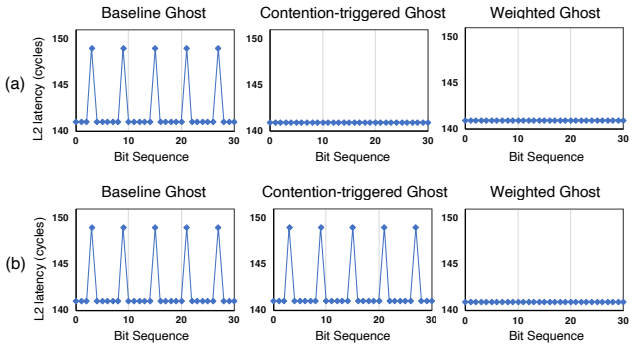


Fig. 15: L2 access latency change according to the number bit. Sender and Receiver communicate '000100010001..' using (a) TPC attack channel and (b) GPC attack channel.

arbitration. We assume a sequence of 0's are transmitted – i.e., the 6 senders are not sending any requests and only the receiver (or the spy) is sending periodic requests. As a result, during the '0' transmission, the sender is able to accumulate ghost counter values. However, during the transmission of a '1', the counter value of the senders will exponentially drop while the ghost counters of the receiver increase – thus, ensuring that the receiver obtains higher priority and achieve near constant latency.

C. Ghost Counter Size

An important aspect of ghost arbitration is the ghost counter size (or the number of counter bits). In Figure 17, we vary the number of bits for the ghost counter and measure the "accuracy" of the baseline and the weighted ghost arbiters. The accuracy is measured as a fraction of requests from the receiver that are prioritized – i.e., if all requests from the receiver are prioritized, then the arbitration is secure.

When the counter size is 0, the ghost counter effectively does not exist and thus, ghost arbitration effectively translates to coarse-grain round-robin arbitration as only approximately 30% of the receiver's requests are prioritized (Figure 17(a)). For smaller counter sizes, the ghost counter value can be accumulated easily but beyond 8, baseline ghost arbitration is secure. However, if the size of the counter increases beyond 16 - the accuracy drops significantly. Given the ratio of requests

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

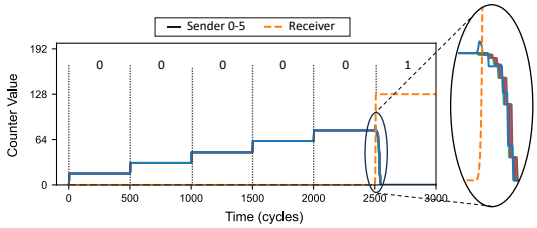


Fig. 16: Changes to the ghost counter values for a GPC channel side-channel attack for weighted ghost arbitration. 7 SMs are sharing the bandwidth and 6 of them are the sender or the victim while the remaining SM is the receiver (or the spy).

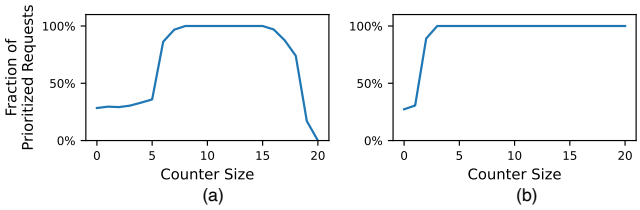


Fig. 17: Fraction of prioritized requests from receiver according to the counter size at (a) baseline ghost arbitration and (b) weighted ghost arbitration.

from the sender to receiver is 16:1, larger counter size is insecure as the sender can accumulate a large amount of ghost counter values – and thus, ensure that the receiver is not prioritized.

In comparison, the weighted ghost arbitration is more secure as shown in Figure 17(b). Similar to the baseline ghost arbitration, the sender can accumulate a large amount of ghost counter values. However, the exponential changing of the ghost counter ensures that the weighted ghost arbitration remains secure for reasonable-size ghost counters. Although not shown, if the counter size does continue to increase, it is feasible that weighted ghost arbitration can lead to lower accuracy – however, the secure arbitration does not necessarily need such larger ghost counters and in addition, the bandwidth of the attacker (or the sender) is reduced by approximately $O(2^n)$ where n is the number of bits in the ghost counter.

D. Workload Evaluations and Overhead

Performance comparison of real workloads using a GPU using different arbitration schemes is shown in Figure 18. Unlike previous analysis which assumed co-location to enable side-channel (or covert-channel) attacks, we evaluate "normal" execution where a single application is utilizing the entire GPU and evaluate the potential performance degradation. The baseline is the conventional round-robin arbitration that is commonly used and we compare against SRR [3] and the proposed weighted ghost arbitration and the results show performance degradation (or slowdown) from the secure arbitration – thus, lower is better. For workloads that do not heavily utilize the L2, the impact of secure arbitration is negligible – however, for L2 intensive workloads, the performance degradation increases

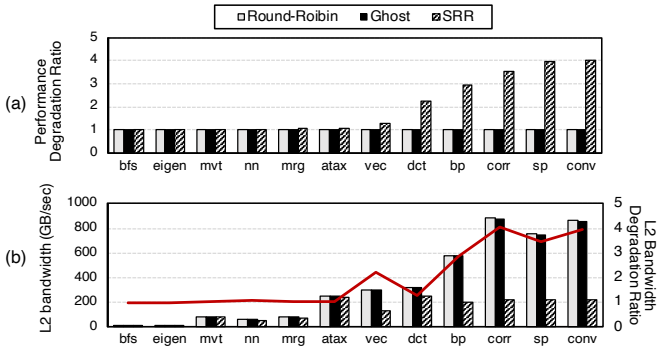


Fig. 18: (a) Performance impact of different *secure* arbitration algorithms and (b) an analysis of L2 bandwidth utilization. The red line shows the L2 performance degradation ratio from SRR, compared to baseline.

by up to 4×. SRR introduces significant performance overhead since it effectively time slices each physical channel – thus, the effective L2 bandwidth decreases significantly as shown in Figure 18(b). Workloads with high L2 bandwidth demands (e.g., `conv`) suffer from relatively higher performance degradation, and the workloads that have low L2 bandwidth demands (e.g., `bfs`) results in minimal performance degradation.

The area and power overhead of the weighted ghost arbiter were estimated using a detailed RTL model using PyOCN [48]. The new arbiter module for ghost arbiter was added and tested using PyOCN. We synthesized the Verilog RTL model of the ghost arbiter using OpenRoad [4] and 7nm ASAP PDK [7]. We synthesized the ghost arbiter with an 8-bit counter and 5 input nodes, and observed 2.1 % (34 μm^2) of area overhead and 5.8 % (5.76 mW) power overhead, compared to the baseline router with round-robin arbitration.

VI. Discussion

In this section, we provide a discussion about the security guarantees of the proposed weighted ghost arbitration. The following parameters are used in the description of the interconnect contention side-channel attack discussion in this section.

m	number of requests sent by the sender for a '1'
k	number of requests sent by the receiver
n	number of bits in the counter
C_s, C_r	ghost counter for sender (receiver)
T_1, T_0	observed latency when transmitting bit '1' ('0')

A. Security Discussion

To ensure that a side-channel cannot be established, the receiver must not be able to distinguish different access patterns of the sender from its latency measurements. Thus, arbitration is *secure* if $T_1 \approx T_0$. For the interconnect contention-based side-channel attack, the following two observations can be made.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

Lemma 1. *The sender needs to transmit more requests than the receiver to generate a reliable contention when communicating a '1'.*

Given the non-determinism in the arbitration and scheduling of requests within a core, the number of requests (or packets) transmitted from the sender must be greater than what the receiver sends – to ensure contention will occur. We provide additional empirical analysis in the subsequent section (Sec VI-B).

Lemma 2. *For ghost arbitration, $T_1 \approx T_0$ if the receiver is always prioritized during contention or when communicating a '1.'*

Ghost arbitration prioritizes the node with a higher ghost count; thus, to ensure ghost arbitration is secure, the ghost counter for the receiver (C_r) needs to be higher than the ghost counter of the sender (C_s). This will ensure that the receiver is prioritized and effectively constant latency is observed by the receiver. In our analysis, without loss of generality, we assume the following in our analysis.

- On average, the contention of the receiver occurs in the middle (or at $m/2$ request).
- The receiver only sends one request ($k = 1$).
- The attacker is aware of the arbitration algorithm – thus, we assume that the attacker (or the sender) has accumulated the ghost counters and the sender currently has a maximum value (i.e. $2^n - 1$)

As discussed earlier, the arbitration that is able to prioritize the receiver can prevent information leakage when transmitting a '1'. Based on the assumptions above, when the '1' is being sent, the sender has the maximum counter value ($C_s = 2^n - 1$) while the receiver has the smallest value ($C_r = 0$). During the transmission of consecutive 1's, the receiver sends $m/2$ ghost requests and increases its ghost counter appropriately while for the sender, its ghost count is decremented. For the baseline ghost arbitration where counters are incremented or decremented by 1, the counter values at contention for the receiver and sender are the following.

$$C_r = m/2$$
$$C_s = 2^n - 1 - m/2$$

To ensure that the receiver is prioritized, the ghost count of the receiver has to be greater than the sender,

$$C_r > C_s$$
$$m/2 > 2^n - 1 - m/2$$
$$n < \log_2(m+1)$$

For the weighted ghost arbitration, the key difference is how the ghost counters are incremented (decremented). Since we assume the ghost values are bit-wise shifted, the counter values at contention for the receiver and sender are the following.

$$C_r = 2^{m/2}$$
$$C_s = 2^{n-m/2} - 1$$

Similarly, to ensure that the receiver is prioritized, the ghost count of the receiver needs to be bigger than the sender.

$$2^{m/2} > 2^{n-m/2} - 1$$
$$n \leq m$$

Thus, to prevent information leakage from interconnect contention-based timing side-channel attacks, the maximum size of the ghost counter size is $\log_2(m+1))$ for the baseline ghost arbitration and m for weighted ghost arbitration. If the counter size exceeds this, then the sender can accumulate sufficient ghost counters to ensure that the receiver does *not* receive priority and leak information. m is a parameter that is a function of the microarchitecture (e.g., in the V100, $m = 16$) and our analysis is consistent with the empirical analysis (as shown in Figure 17) where the weighted ghost arbitration has a much higher bound, compared to the baseline ghost arbitration. Since the bandwidth of the information leakage is reduced by $O(2^n)$, providing a higher upper bound is beneficial for secure arbitration.

In this analysis, we assumed there is only 1 request sent from the receiver (i.e. $k = 1$). If the receiver has multiple requests (i.e., $k > 1$), as long as the first request can be granted, the rest of the request can also be granted with coarse-grain round-robin arbitration (e.g., the grant is asserted at a warp granularity in a GPU). As shown in Figure 17, having a small counter can be problematic since the counters can easily saturate (for both the sender and the receiver).

The analysis above assumed that real contention occurs at $m/2$; however, contention may occur earlier or later. If contention occurs after $m/2$, it is not a problem since the receiver will have sufficient ghost counter compared to the sender but if the contention occurs earlier, it can be problematic since the sufficient ghost counter might not have been accumulated at the receiver. If the contention does occur "early," then it is possible that the receiver will continue to accumulate ghost counters. However, another limitation is that for $k > 1$, While having a larger counter can prevent counter saturation, the sender can exploit it to accumulate ghost counts and enable information leakage. Thus, for a given m, an optimal region of counter size can be determined; however, the attacker can potentially vary m to leak information. As a result, weighted ghost arbitration is more robust as shown earlier in Figure 17(b) since there is a much wider range of counter values for which the arbitration is secure.

B. Error rate

In this subsection, we provide additional evaluations that analyze the impact of different components on the error rate (or the accuracy), including the polling rate of the spy (or the receiver), the difference between the TPC and GPC-channel, and the impact on the ratio of request from the receiver and the sender.

1) Polling rate: One factor that impacts the accuracy (as well as the bandwidth of information that can be leaked) is the polling rate of the spy or the interval at which the spy generates requests. In our initial evaluation, we set the interval

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

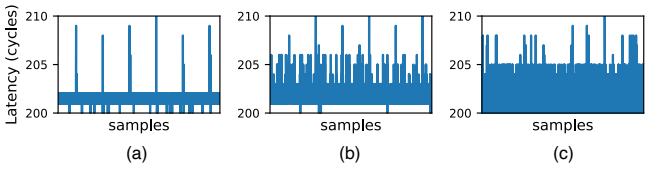


Fig. 19: Information leakage as the receiver request injection interval is varied from (a) 260 (b) 250 and (c) 240 cycles per sample for a sequence of '101010...'

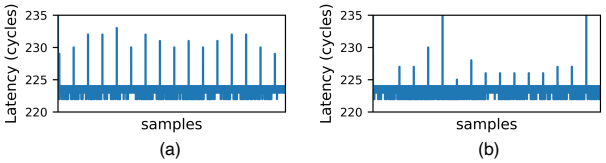


Fig. 20: Comparison of (a) TPC channel and (b) GPC channel covert channel accuracy on V100. For TPC channel, there is one sender (SM) and one receiver while for the GPC channel, there are six senders and one receiver.

at 300 cycles to approximately match the L2 round-trip latency. In Figure 19, we vary the polling rate and reduce the interval at which the requests are sent. The results show that once the polling rate is lowered to 240 cycles, the information leakage becomes very limited because of the *self*-interference from the multiple requests from the spy kernel – thus, although higher polling rate can lead to more contention, it will also cause higher error rate when reduced beyond the L2 round-trip latency.

2) TPC vs GPC channel: The difference between TPC-channel and GPC-channel measurements are shown in Figure 20 for a sequence of '1010...' The TPC-channel has one SM as a sender and the one SM within the same TPC as the receiver while for the GPC-channel, 6 SMs are the sender while 1 SMs are the receiver. The TPC channel attack (Figure 20(a)) achieves a much higher signal-to-noise ratio as the difference between the '1's and the '0's are very clear. In comparison, the GPC-channel results in more noise as shown in Figure 20(b) – likely because of the additional sharing and the interconnect speedup at the GPC hierarchy; however, information can still be leaked and be leveraged to establish side-channel attacks.

3) Sender vs Receiver Ratio: The impact of the number of requests ratio between the sender and the receiver is shown in Figure 21. When the receiver sends more requests, contention can not be reliably observed by the receiver, as shown in Figure 21(a). However, as the number of requests from the sender (compared to the receiver) is increased, the amount of noise is reduced and some information can be leaked (Figure 21(b)) and when the ratio increases, the error rate decreases significantly as shown in Figure 21(c). The overall change in the error rate as the receiver-to-sender request ratio is varied is shown in Figure 21(d). The results were obtained for a V100 GPU and other GPU architectures will have different characteristics; however, to establish a reliable

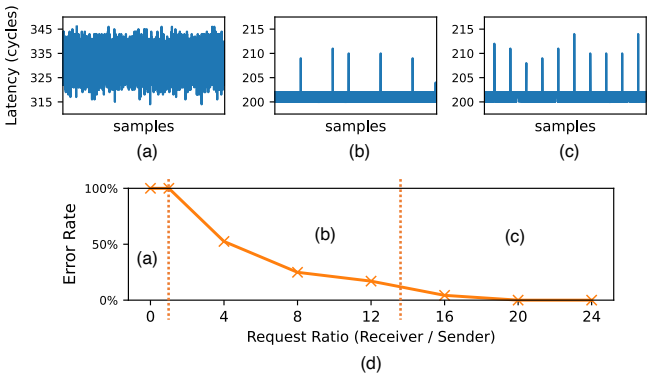


Fig. 21: Change in the information leakage as the ratio of requests (or packets) between the sender and the receiver is varied between (a) 1:4, (b) 4:1, and (c) 16:1. Change in the error rate is shown in (d) and for V100 GPU, the request ratio of approximately 16 is needed to achieve a very low error rate.

side-channel between the sender and the receiver through interconnect contention, the sender needs to transmit more requests compared to the receiver.

C. Related Work

Side-Channel Attacks: Many work have demonstrated different types of side-channel attacks on the GPU. One type of side-channel attack is using timing information obtained from the CPU to infer in information about the secret key for AES [2], [19], [20] and RSA [33] by correlating the execution time with different characteristics of the secret key. Other work have explore the shared memory hierarchy including prime-probe attacks on L2 caches [12] and L3 TLBs [57] for fingerprinting attacks. The shared resources in the memory controller [49] and the memory row-buffer [46] has also been exploited for side-channel attack. Other side-channel attacks have exploited GPU performance counters to infer behaviors of the victim applications, including website fingerprinting [36], model extraction [16], [53], and keystroke detection [55]. However, because of the threat of side-channel attacks, NVIDIA has recently restricted access to some performance counters for the users. [40] To overcome these challenges, some have demonstrated how a device driver downgrade [53] can be done to access performance counters. In comparison, this work does not rely on performance counters but contention on the interconnect bandwidth or the on-chip L2 traffic observed by the spy.

This work, similar to many prior work [16], [36], [53], [55], requires co-location of victim and spy in the same GPU. Other prior work have explored threat models such as co-location in a multi-GPU system [12], multiple GPU instances within a single physical GPU [57], while others have relied on the CPU host for the GPU [19], [20], [33]. It remains to be seen if the the side-channel attack proposed in this work can be extended to other type of interconnection networks such as the multi-GPU system interconnect. Shared interconnect has been exploited for side-channel attacks on different type of systems. Memory

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

bus-based side-channel attack [54] was demonstrated but a "lock" of the shared bus is required and results in relatively low bandwidth attack. Network-on-chip side-channel attack was first shown using a simulation-based study [51] and recent work [8], [44] demonstrated feasibility on a real multi-core CPU. However, the arbitration for the Intel CPU mesh (and ring) topologies are based on priority arbitration such that packets in-flight are prioritized compared to packets that need to be injected into the NoC. The priority or "unfairness" can be leveraged for timing side-channel attacks but in this work, the arbitration is "fair" (i.e., round-robin arbitration) and we demonstrate how side-channel attacks can still be enabled.

Side-Channel Defense: As discussed earlier, one solution to prevent side-channel attacks is to provide priority. Prior work [51] assumed that there are predefined security domains and lower security domains (or the "receiver") at the memory controller were prioritized. However, predefining such security domains is a challenge, especially for GPUs. SurfNoC [52] exploited time-division multiplexing across the entire topology to provide full isolation, similar to the secure arbitration [3]; however, such strict partitioning of the resources can impact performance and resource utilization. With priority-based arbitration in multi-core NoC, careful placement of the attacker and victim applications can also be used to mitigate side-channel attacks [8] but is not necessarily feasible when the arbitration is locally fair.

The side-channel attack in this work focused on the *source*-side of the on-chip interconnect but *destination*-side contention can occur at the input to the L2 slices (or the memory partitions). Remapping of L2 slices [23] was proposed to alleviate such side-channel attack but does not address the source-side side-channel. GPUGuard [21] uses decision tree classifiers to detect attacks but comes at the cost of increased latency; however, their work addresses *intra*-SM contention and cannot be applied for *inter*-SM contention that we address in this work. Randomization has been proposed to prevent correlation-based attacks by introducing randomization to different components in the hardware and scheduling [21], [22], [24]; however, random arbitration does not necessarily mitigate interconnect contention-based side-channels.

VII. Conclusion

In this work, we demonstrated how timing side-channel attacks can be implemented on modern GPUs by exploiting the shared on-chip interconnect bandwidth resources. In particular, we show how *interconnect-gram* can be obtained by the spy kernel to enable fingerprinting attack by using the on-chip interconnect and L2 access patterns. To prevent information leakage, we propose a secure arbitration that we refer to as *ghost arbitration*. Naive implementation of ghost arbitration can still be leveraged by the spy for side-channel attacks but we show how *weighted* ghost arbitration prevents information leakage by exploiting the "malicious" communication pattern needed to enable interconnect contention and adjusting the arbitration priority accordingly.

Acknowledgment

We would like to thank the anonymous reviewers for their insightful comments. This work was supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 20211F1A1064009), IITP-RS202300228255, NRF-2023R1A2C200422912, and IITP-RS202400402898.

References

[1] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting secret keys via branch prediction," in *Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology*. San Francisco, CA: Springer, 2007, pp. 225–242.

[2] J. Ahn, C. Jin, J. Kim, M. Rhu, Y. Fei, D. Kaeli, and J. Kim, "Trident: A Hybrid Correlation-Collision GPU Cache Timing Attack for AES Key Recovery," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 332–344.

[3] J. Ahn, J. Kim, H. Kasan, L. Delshadtehrani, W. Song, A. Joshi, and J. Kim, "Network-on-chip microarchitecture-based covert channel in gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, Virtual Event, Greece, 2021, pp. 565–577.

[4] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, R. G. Dreslinski, M. Fogac¸a, S. M. Hashemi, A. A. Ibrahim, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. I. Pénzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. K. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo, and B. Xu, "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain," *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, pp. 1105–1110, 2019.

[5] A. Bakhoda, G. L. Yuan, W. W. L. 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*, 2009, pp. 163–174.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, USA, 2009, pp. 44–54.

[7] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[8] M. Dai, R. Paccagnella, M. Gomez-Garcia, J. McCalpin, and M. Yan, "Don't mesh around: Side-Channel attacks and mitigations on mesh interconnects," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, Aug. 2022, pp. 2857–2874.

[9] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Elsevier, 2004.

[10] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference*, Las Vegas, NV, USA, 2001, pp. 684–689.

[11] S. Deng, W. Xiong, and J. Szefer, "A Benchmark Suite for Evaluating Caches' Vulnerability to Timing Attacks," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Lausanne, Switzerland, 2020, pp. 683–697.

[12] S. B. Dutta, H. Naghibijouybari, A. Gupta, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23, Orlando, FL, USA, 2023.

[13] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, Mar. 2018.

[14] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–10.

[15] D. Gullasch, E. Bangerter, and S. Krenn, "Cache Games – Bringing Access-Based Cache Attacks on AES to Practice," in *2011 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2011, pp. 490–505.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

[16] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, Lausanne, Switzerland, 2020, pp. 385–399.

[17] R. Hund, C. Willems, and T. Holz, "Practical Timing Side Channel Attacks against Kernel Space ASLR," in *2013 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, 2013, pp. 191–205.

[18] K. Jang, S. Han, S. Han, S. Moon, and K. Park, "SSLShader: Cheap SSL acceleration with commodity processors," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, Boston, MA, Mar. 2011.

[19] 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)*, Barcelona, Spain, 2016, pp. 394–405.

[20] ——, "A novel side-channel timing attack on gpus," in *Proceedings of the Great Lakes Symposium on VLSI 2017*, Banff, Alberta, Canada, 2017, p. 167–172.

[21] Z. Jin, C. Rocca, J. Kim, H. Kasan, M. Rhu, B. Ali, T. M. Aamodt, and J. Kim, "Uncovering Real GPU NoC Characteristics: Implications on Interconnect Architecture," in *MICRO-57: 57th Annual IEEE/ACM International Symposium on Microarchitecture*, Austin, TX, USA, 2024.

[22] 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)*, Vienna, Austria, 2018, pp. 156–167.

[23] A. Kar, X. Liu, Y. Kim, G. Saileshwar, H. Kim, and T. Krishna, "Mitigating Timing-Based NoC Side-Channel Attacks With LLC Remapping," *IEEE Computer Architecture Letters*, vol. 22, no. 1, pp. 53–56, 2023.

[24] E. Karimi, Y. Fei, and D. Kaeli, "Hardware/Software Obfuscation against Timing Side-channel Attack on a GPU," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA, 2020, pp. 122–131.

[25] M. Kayaalp, D. Ponomarev, N. Abu-Ghazaleh, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, 2016, pp. 1–6.

[26] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 473–486.

[27] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, Minneapolis, MN, USA, 2014, pp. 361–372.

[28] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2019, pp. 1–19.

[29] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, 1996, pp. 104–113.

[30] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[31] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, Aug. 2018, pp. 973–990.

[32] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*. San Jose, CA, USA: IEEE, 2015, pp. 605–622.

[33] C. Luo, Y. Fei, and D. Kaeli, "Side-channel Timing Attack of RSA on a GPU," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, Aug. 2019.

[34] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Boston, MA, USA, 2017, pp. 354–366.

[35] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered Insecure: GPU Side Channel Attacks are Practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, Canada, 2018, p. 2139–2153.

[36] ——, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, ser. CCS '18, Toronto, Canada, 2018, pp. 2139–2153.

[37] Nan Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, USA, 2013, pp. 86–96.

[38] M. Neve and J.-P. Seifert, "Advances on access-driven cache attacks on AES," in *Proceedings of the 13th International Conference on Selected Areas in Cryptography*, ser. SAC'06, Montreal, Canada, 2006, p. 147–162.

[39] NVIDIA, "NVIDIA TESLA V100 GPU ARCHITECTURE," 2017, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[40] ——, "NVIDIA Response to "Rendered Insecure: GPU Side Channel Attacks are Practical"," NVIDIA Corporation, Tech. Rep., 2018, https://nvidia.custhelp.com/app/answers/detail/a_id/4738/~/security-notice%3A-nvidia-response-to-.

[41] NVIDIA, "Multi-Process Service," https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf, 2020.

[42] NVIDIA Corporation, "NVIDIA CUDA SDK code Sample," http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html.

[43] ——, "NVIDIA A100 Tensor Core GPU Architecture," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020.

[44] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 645–662.

[45] C. Percival, "Cache missing for fun and profit," 08 2009.

[46] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for Cross-CPU attacks," in *25th USENIX security symposium (USENIX security 16)*, Austin, TX, USA, 2016, pp. 565–581.

[47] A. Shusterman, Z. Avraham, E. Croitoru, Y. Haskal, L. Kang, D. Levi, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Website fingerprinting through the cache occupancy channel and its real world practicality," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2042–2060, 2021.

[48] C. Tan, Y. Ou, S. Jiang, P. Pan, C. Torng, S. Agwa, and C. Batten, "PyOCN: A Unified Framework for Modeling, Testing, and Evaluating On-Chip Networks," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, Abu Dhabi, United Arab Emirates, 2019, pp. 437–445.

[49] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Orlando, FL, USA, 2014, pp. 225–236.

[50] Y. Wang and G. E. Suh, "Efficient Timing Channel Protection for On-Chip Networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 142–151.

[51] Y. Wang and G. E. Suh, "Efficient Timing Channel Protection for On-Chip Networks," in *2012 6th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, ser. NOCS '12, Lyngby, Denmark, 2012, pp. 142–151.

[52] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, Tel-Aviv, Israel, 2013, p. 583–594.

[53] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Valencia, Spain, 2020, pp. 125–137.

[54] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: high-speed covert channel attacks in the cloud," in *Proceedings of the 21st USENIX Conference on Security Symposium*, 2012, p. 9.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.

[55] B. Yang, R. Chen, K. Huang, J. Yang, and W. Gao, "Eavesdropping user credentials via GPU side channels on smartphones," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Lausanne, Switzerland, 2022, pp. 285–299.

[56] Y. Yarom and K. Falkner, "FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14, San Diego, CA, Aug. 2014, p. 719–732.

[57] Z. Zhang, T. Allen, F. Yao, X. Gao, and R. Ge, "TunneLs for Bootlegging: Fully Reverse-Engineering GPU TLBs for Challenging Isolation Guarantees of NVIDIA MIG," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23, Copenhagen, Denmark, 2023, p. 960–974.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:59:07 UTC from IEEE Xplore. Restrictions apply.