# Network-on-Chip Microarchitecture-based Covert Channel in GPUs

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

Leila Delshadtehrani
Boston University
Boston, MA, USA
delshad@bu.edu

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

Ajay Joshi
Boston University
Boston, MA, USA
joshi@bu.edu

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

## ABSTRACT

As GPUs are becoming widely deployed in the cloud infrastructure to support different application domains, the security concerns of GPUs are becoming increasingly important. In particular, the support for multiprogramming in modern GPUs has led to new vulnerabilities since multiple kernels in a GPU can be executed at the same time. In this work, we propose a new microarchitectural timing covert channel for GPUs that can be established based on the shared, on-chip interconnect channels. We first reverse-engineer the organization of the on-chip networks in modern GPUs to understand the core placements throughout the GPU. The hierarchical organization of the GPU results in the sharing of interconnect bandwidth between neighboring cores. Based on this understanding, we identify how contention for the interconnect bandwidth can be exploited for a novel covert channel attack. We propose two types of interconnect-based covert channels that exploit the on-chip network hierarchy. Unlike cache-based covert channels, no states of the on-chip network need to be modified for communication in our interconnect-based covert channel and the impact of contention is very predictable. By exploiting the parallelism of GPUs, our proposed covert channel results in very high bandwidth – achieving approximately 24 Mbps of bandwidth on NVIDIA Volta GPUs and results in one of the highest known microarchitectural covert channel bandwidth.

## CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

## KEYWORDS

GPU, Covert Channel, Network-on-Chip

**ACM Reference Format:**
Jaeguk Ahn, Jiho Kim, Hans Kasan, Leila Delshadtehrani, Wonjun Song, Ajay Joshi, and John Kim. 2021. Network-on-Chip Microarchitecture-based Covert Channel in GPUs. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3466752.3480093

## 1 INTRODUCTION

The compute and memory capacity of GPUs are continuing to grow and enable more applications to leverage GPUs [13]. The large number of compute cores and their corresponding need for large memory bandwidth had made it critical to design an efficient on-chip network. One key characteristic of any on-chip network is that the resources, in particular, interconnect channel bandwidth, are *shared* resources [63, 68]. In this work, we focus on the on-chip network and explore the opportunity to exploit the shared, high bandwidth on-chip network of GPUs to create a novel, very high-bandwidth interconnect microarchitectural covert channel.

One feature of recent GPUs is the ability to support multiprogramming – i.e., have multiple kernels executing on a GPU at the same time [55]. While this can improve the utilization of the GPUs and the overall system performance [2, 66, 69], it opens up potential security vulnerabilities in the system because multiple kernels share GPU resources concurrently. Anytime there is a shared resource such as on-chip cache, memory bus, or functional units [21], it opens up opportunities for malicious actors to establish covert channels by encoding a bit as either contention or no-contention in the shared resource. A covert channel can be dangerous as it allows a malicious program to intentionally leak information covertly [54].

The key challenges in establishing covert channels are to reliably detect the occurrence of contention and minimize synchronization overhead. Wu et al. [68] outlined the challenges of uncertainty in cache-based covert channels including the scheduling uncertainty between the sender and the receiver. Other prior works have demonstrated microarchitectural timing covert channels [37, 40, 56]. Recently, a very high-bandwidth, 4 Mbps covert channel between two concurrent kernels was demonstrated on GPUs [42] – one of the highest known covert channel bandwidth. In our work, we propose a novel covert channel that achieves approximately 24 Mbps by exploiting the on-chip shared GPU interconnect. Wu et al. [68] also used the interconnect (or the memory bus) to create a covert channel; however, the bandwidth achieved is significantly lower since it relies on "locking" the bus to create a contention (or delay) in access for the receiver. In contrast, our work exploits the *direct* contention for the physical channel that is shared and greatly reduces the overhead in creating a covert channel – resulting in significant bandwidth.

In our work, we exploit the contention of interconnect channel for covert channel. In particular, we explore the on-chip interconnect in GPUs that is shared by the cores and provides a more *predictable* or a *direct* impact on the performance from contention to enable a high-bandwidth covert channel. We first reverse-engineer the on-chip network of GPUs to identify the organization and the core placement. Based on the reverse-engineering, we show how on-chip communication (or the hierarchical communication organization) is shared between the streaming multiprocessors (SM) or the cores. We then propose two different types of covert channels – TPC (Texture Processing Cluster) channel and GPC (Graphics Processing Cluster) channel – that exploit the contention for on-chip interconnect across the GPU architecture. In particular, we demonstrate up to 24 Mbps of bandwidth for covert channel with a negligible error by exploiting GPU characteristics including hierarchical core organization with limited sharing of the interconnect and parallel threads executed concurrently to increase the number of requests and increase contention. A single TPC channel (between two SM cores) can create a covert channel of $\approx 1$ Mbps, and by exploiting the large number of parallel cores a covert channel with bandwidth up to 24 Mbps can be achieved with a negligible amount of error.

One challenge in maintaining a reliable covert channel is synchronization between the sender and the receiver. Alignment and handshaking are necessary to synchronize communication between the sender and the receiver. However, both these operations can limit the overall bandwidth of the covert channel. We identified how clock registers available in GPUs can be leveraged for synchronization *without* requiring any explicit communication or alignment between the sender and receiver. We determined that the clocks in neighboring cores of the GPU have very low clock skew such that they can be leveraged directly with local synchronization. A combination of a low overhead synchronization and a *local* resource (i.e. physical interconnect channel) that is *directly* shared between the cores enables one to establish and maintain a reliable covert channel that achieves significantly higher bandwidth compared to other known covert channels. To subvert the interconnect-based covert channel identified in our work, we propose to modify the interconnect arbitration, which effectively disables the covert channel with minimal impact on overall system performance. In addition to the reverse-engineering and demonstration of a novel high-bandwidth covert channel, the main contribution of this work is that ***interconnect design (in addition to cache, memory, core pipeline, etc.) needs to consider the security implications of the performance-centric design choices.*** In particular, the contributions of this work include the following.

- We reverse-engineer the on-chip interconnect of modern GPUs, identify how the cores are mapped across the GPU hierarchy, and the bandwidth sharing that occurs between co-located cores in the GPU.
- We propose a novel microarchitectural interconnect covert channel that leverages on-chip interconnect contention to create two types of interconnect covert channels, TPC channel and GPC channel.
- We demonstrate interconnect covert channel on a modern GPU. By exploiting the parallelism in GPU, we achieve covert channel bandwidth of 24 Mbps with negligible error.
- We propose a countermeasure through *secure* arbitration – in particular, we propose strict round-robin arbitration that provides temporal partitioning to thwart the proposed covert channel attack.

## 2 BACKGROUND

### 2.1 GPU Architecture

A modern GPU consists of several components such as SM, L2 cache, and Memory. Multiple SMs are connected to multiple L2 cache slices through the on-chip network (e.g, cross-bar) and each L2 slice is coupled with different Memory Controllers (MCs) and memories. A kernel is divided into multiple thread blocks and the thread blocks can be assigned to each SM in an arbitrary manner [7, 11, 29, 34]. After dispatching the thread blocks to SMs, each thread block is further split into a group of threads called a *warp* or a *wavefront*. The warp scheduler issues one of the ready warps among multiple warps based on a warp-scheduling algorithm [33, 57, 58, 60]. When a warp executes a memory request, *memory coalescing* can reduce the memory bandwidth consumption. If multiple threads in a warp access a contiguous memory block or a same cache line, their requests can be coalesced into one contiguous, aligned memory access. After memory accesses are coalesced, the memory requests from a warp access the on-chip memory hierarchy (i.e. L1 cache, L2 cache, and the main memory).

Recent GPUs have support for multiprogramming to enable multiple kernels to execute on the same GPU concurrently [55]. Multiprogramming on NVIDIA GPUs can be done with multiple streams within the same process or leverage MPS (Multi-Process Service) [49] that allow concurrent kernels from different processes. Similar to prior work [42], we utilized streams (cudaStream) for multiprogramming on GPU for the proposed covert channel attack but this work is also applicable to MPS. The results in this work are with cudaStream but we confirmed that the same covert channel can be carried out with MPS and achieve the same high-bandwidth with negligible impact on the error rate. The only difference was the one-time synchronization overhead required to properly synchronize the launch of the multiple processes from the CPU.

### 2.2 Threat Model

Covert channels can be referred to as "insider threats" where two processes establish an illegitimate communication channel through timing modulation to leak information [10, 41] and are vulnerable in cloud computing as well [68]. As a result, covert channel is an *intentional* channel that transmits data between processes that are not permitted to communicate with each other according to the system security policy [61]. Because of imperfections in isolation usage of shared resources, malicious processes can exploit these shared resources to set up a communication channel. The covert channel consists of a sender, referred to as the trojan, and a receiver, often referred to as the spy. With covert channel, the trojan can send sensitive information or leak data to the spy.

In this work, we target computing systems with GPU that are shared by multiple tenants. We assume that users are not allowed physical access to the target system and only remote access is possible. Our threat model consists of sender (or trojan) and receiver (or spy) kernels that co-exist concurrently on the same GPU. We assume both the sender and receiver processes are user-level processes without any additional privileges. We also assume the sender possesses secret or sensitive information and can only communicate with other trusted components, such as an encryption service, through defined channels. However, the sender intentionally leaks the secret through the covert channel and the receiver reads the secret leaked by the sender. In this work, our goal is to establish a covert channel between the trojan (sender) and the spy (receiver) kernels. Our covert channel attack and the analysis were conducted on NVIDIA Volta V100 GPU with CUDA 10.0.

## 2.3 NoC Concentration and Arbitration

One critical aspect of on-chip networks [14, 22] or any interconnection networks is sharing of resource – in particular, sharing of the channel bandwidth between different endpoints. A simple approach to sharing bandwidth is a multiplexer (mux) – often referred to as network *concentration* [15]. For an $N$ input mux and a single output with the same bandwidth, the output bandwidth is oversubscribed by a factor of $N$. This can cause a performance bottleneck and *speedup* [15] can be provided where *excess* bandwidth is provided at the output channel, compared to the input channel, to improve overall performance. Another important component of the on-chip interconnect is the arbitration policy. In this work, we exploit how concentration is exploited across the GPU hierarchy and the on-chip interconnect to establish a covert channel. In addition, we identify how the *secure* arbitration policy can be used to provide a countermeasure against the proposed covert channel attack.

In this work, we explore the concentration that occurs within the GPU hierarchy – especially between the SMs within a TPC to create a covert channel that we refer to as *TPC channel*. Example placement of the receiver (or the spy) and the sender (or the trojan) is shown on the left in Figure 1, where the sender is shown as *TPC sender*. The different TPCs within the GPC can also be leveraged to create a different type of covert channel that we refer to as *GPC channel*. Figure 1 also illustrates GPC channel with the same receiver as the TPC channel but the sender processes are distributed across multiple TPCs as shown with *GPC sender*. To increase the bandwidth of TPC channels, multiple TPCs can be exploited while the GPC channel bandwidth can also be increased by using a large number of GPCs. [1]

## 3 REVERSE ENGINEERING GPU ON-CHIP NETWORK

In this section, we reverse engineer the organization of a GPU's on-chip network as well as how the various SMs are connected to each other via the on-chip networks and leverage the SM connectivity information for our proposed interconnect covert channel attack. Note that our use of TPC (or GPC) channels refers to logical channels between the SMs (or TPCs) to illustrate the covert channel.

[1]Note that recent NVIDIA GPUs provide isolation between the GPCs through MIG [50]; however, the isolation is provided *between* the GPCs and MPS [49] is still supported within an instance or within the GPC. Thus, the proposed covert channel is still applicable to recent GPUs with MIG.

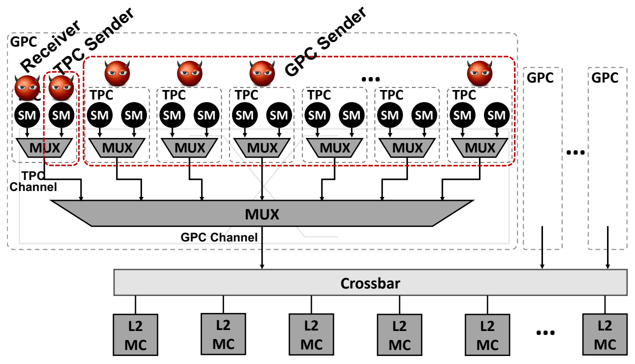

Figure 1: GPU hierarchical organization block diagram.

## 3.1 GPU Hierarchical Organization

It is well-known that GPUs are designed hierarchically [3, 12, 18, 28]. In NVIDIA GPUs, the basic building block of the GPUs are the Streaming Multiprocessors (SMs).[2] Two SMs are grouped together to form a Texture Processing Cluster (TPC) and multiple TPCs are grouped together to form a Graphics Processing Cluster (GPC) [35, 45, 46, 48]. As an example, an NVIDIA Volta GPU can consist of 6 GPCs, with each GPC consisting of 7 TPCs and each TPC consisting of 2 SMs [46]. In this hierarchical organization, all the individual SMs have a private L1 cache that communicates with the L2 cache. The L2 cache is banked with multiple L2 cache banks sharing a memory controller. While it is well-known that a high-throughput on-chip network is necessary to interconnect the SMs and memory, to the best of our knowledge, all the details of the NVIDIA on-chip network are not publicly available. Publicly available block diagrams [45] show a crossbar in the middle of the GPUs – however, we suspect that the crossbar is used to interconnect the GPCs to the partitioned L2 cache. A high-level logical block diagram of GPU hierarchical organization is shown in Figure 1. In our work, we first reverse-engineer the on-chip network organization in an NVIDIA Volta GPU and determine how the SMs are connected to each other via this on-chip network. The logical organization (or hierarchical organization) of the SMs in NVIDIA GPUs is well-known [35, 45, 46, 48]; however, the ***physical implications of the hierarchical organization is not known*** – e.g., while the SMs are hierarchically organized, the interconnect can be "flat" or organized differently. We determine the relative connectivity and the hierarchical interconnect organization of the SMs across the TPCs and the GPCs, and then leverage this information to implement a covert channel.

## 3.2 TPC Channel

Since it is known that two SMs are placed together within a TPC [35, 45, 46, 48], we first attempt to reverse engineer the organization of the SMs to identify which SMs are co-located within a given TPC. Our hypothesis is that ***SMs that are co-located within a TPC share the TPC channel (Figure 1) and this can degrade the performance of the two SMs***. To validate our hypothesis, we run the synthetic benchmark shown in Algorithm 1 on the GPU.

[2]GPUs from AMD are also organized hierarchically – e.g., compute units, shader array, and shader engine [3].

**Algorithm 1:** Reverse Engineering TPC Organization

```
input: $config\_smid$   // SM id to be activated
Procedure Memory Write Test($config\_smid$)
    $sm\_id$ = get_smid()
    $Amount = array\_size / thread\_block\_size$
    $base = Amount \cdot thread\_id$
    if $sm\_id == 0$ then
        for $i \leftarrow 0$ to $Amount$ do
            $arr\_A[base + i] = thread\_id$
        end
    else if $sm\_id == config\_smid$ then
        for $i \leftarrow 0$ to $Amount$ do
            $arr\_B[base + i] = thread\_id$
        end
    return
```

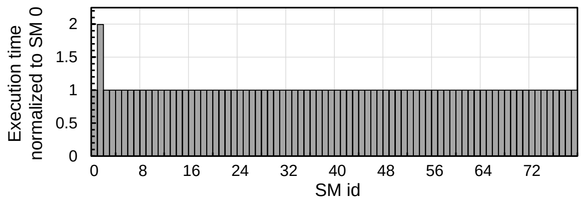

**Figure 2: Execution time of the synthetic benchmark (Algorithm 1) when run on SM0 and one other SM. The $x$-axis represents the ID of the other SMs.**

The goal here is to evaluate a memory-intensive program that continuously accesses the L2 cache (and bypass L1 cache such that the interconnect is accessed). The code does sequential memory write access and ensures that all memory partitions (and corresponding L2 cache) are accessed by the SM. We execute this synthetic code concurrently on SM0 and one other SM in the GPU, i.e., only two SMs are active. The placement of code (or thread block) on the SMs can be identified by using the *smid* register that provides SM ID.

The execution time of the synthetic code when running on SM0 and one other SM is shown in Figure 2. In the figure, the $x$-axis indicates the ID of the SMs. When SM1 is executed concurrently with SM0, the performance degrades by a factor of 2. This factor of 2 degradations comes from the sharing of the TPC channel between the two SMs of a TPC. In comparison, when the code is executed on any other SM, there is no performance degradation, compared to the performance of SM0. Our evaluation across a different combination of SMs confirms that when two SMs within a TPC have consecutive numbers – i.e., SM$i$ and SM$i+1$, where $i$ is an even number, they share the same TPC channel. The key observation is not that two SMs are clustered together but the interconnect bandwidth of the two SMs are shared and the contention for the interconnect bandwidth can be leveraged in the covert channel attack.

## 3.3 GPC Channel

To identify which SMs are co-located within each GPC, we perform an evaluation similar to what was done earlier in Section 3.2. However, instead of activating only 2 SMs, we activate one SM in each of the 7 TPCs [3], i.e., 7 SMs in total. Using a similar approach as before, we always activate TPC0 and then vary the TPC that is selected to run concurrently with TPC0. We use only one SM from each TPC. In addition to these two TPCs, 5 other TPCs are *randomly* selected or made active and we run the evaluation 200 times. Unlike the TPC channel evaluation where we only selected 2 SMs, 7 SMs are needed for this evaluation because of the bandwidth *speedup* [15] that exists within the GPC organization. As shown in Figure 1, the SMs within a TPC share a mux and connect to the TPC channel. Similarly, the TPCs within a GPC share a mux to connect to the GPC channel. However, 7-to-1 mux or an oversubscription by a factor of 7 will result in significant performance degradation for memory-intensive workloads; hence a speedup is commonly implemented [15].

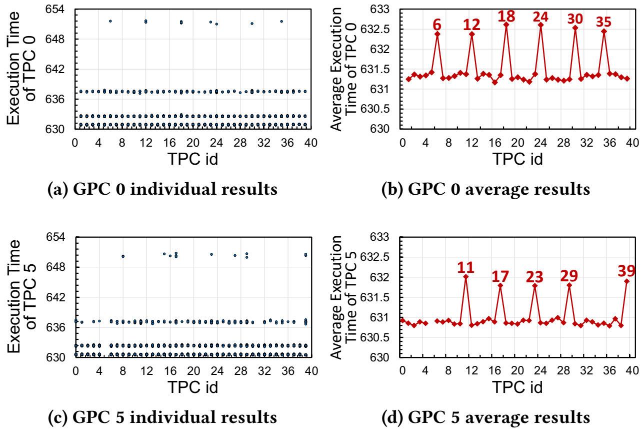

**Figure 3: Performance measurements to identify SM/TPC placement across the GPCs in the Volta GPU.**

The results of our evaluation to determine the SM/TPC organization across the GPCs are shown in Figure 3. Figure 3(a,b) and Figure 3(c,d) show the TPC0 (TPC5) execution time, respectively, where TPC0 (TPC5) are selected as the first TPC. The $x$-axis contains the ID of the other TPC that is selected and the $y$-axis is the execution time of TPC0 (TPC5) for the 200 evaluations with 5 other randomly selected TPCs. There are no explicit TPC IDs on the GPUs and we simply label TPC0 as the TPC that contains SM0 and SM1, TPC1 contains SM2 and SM3, etc. Since there are 40 TPCs in total, $x$-axis value ranges from 0 to 39 and the different data points for a given $x$ value are the performance measurements across 200 evaluations.

The performance of TPC0 varies depending on whether the other randomly selected TPC belongs to the same GPC (see Figure 3(a)). For example, for certain TPC selections (e.g., TPC6, TPC12, etc.), there are a few evaluations that result in higher latency values around 650. We average the 200 measurements of the TPC execution time and demonstrate the results in Figure 3(b). When TPC0 is selected first and the other selected TPC is TPC6, TPC12, etc., the probability of having contention for on-chip network bandwidth becomes much higher, which results in higher execution time. As shown in Figure 3(c,d), we observe a similar pattern in the scenario where TPC5 is selected first. However, the key difference is that the GPC containing TPC5 is only shared with 5 other TPCs, i.e., the GPC

[3]It is known that there are up to 7 TPCs in a GPC of the Volta GPU [46].

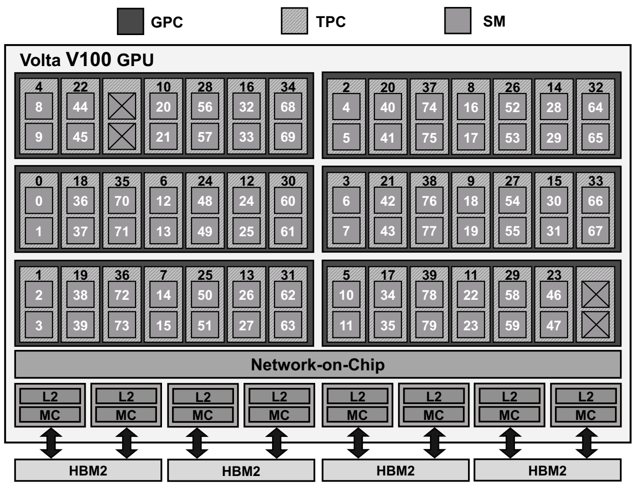

Figure 4: NVIDIA Volta V100 GPU's Logical to Physical Core Mapping.

contains only 6 TPCs and not 7. Given that there are 40 TPCs and 6 GPCs, 4 of the 6 GPCs have 7 TPCs while the remaining 2 GPCs have only 6 TPCs as it appears as if one TPC in each of these two GPCs is disabled. [4] This analysis is used to determine which TPCs are co-located within the same GPC. We repeated the analysis described above across all TPCs, and accordingly determined the mapping of SMs across all GPCs in a single GPU, as shown in Figure 4. Overall, we found that TPCs are mostly interleaved across GPCs. However, considering that some GPCs only have 6 TPCs while other GPCs have 7 TPCs, some of the mappings are not perfectly interleaved. For example, GPC5 contains TPC5, 11, 17, 23, and 29 but instead of TPC35, it includes TPC39.

## 3.4 GPC/TPC Contention Characteristics

The TPC and the GPC hierarchical organization described in the earlier sections can be leveraged to establish a covert channel. When the two SMs within a TPC are used, we refer to this as a *TPC channel*, and when multiple TPCs are leveraged, we refer to this as a *GPC channel*. Both the TPC and the GPC channels are based on the contention for resources during memory requests. However, the type of memory access (i.e., read or write) can have an impact on performance degradation, i.e., the quality of the covert channel. The impact of memory access type on TPC and GPC channels is shown in Figure 5. For the TPC channel, read memory access contention has a minimal impact on performance while write memory access contention results in 2× increase in the execution time, similar to what was shown earlier in Figure 2. However, for the GPC channel, increasing the number of active TPCs (from 1 to 7) within a GPC results in a small performance impact for write requests (degrading performance by only ~15%). We speculate that the TPC bandwidth for write requests is effectively "throttled" due to the TPC output channel contention before contending for the GPC output channel. Hence, the overall performance degradation from increasing the

[4]The Volta GPU that we evaluated (V100) has 40 TPCs but GV100 from NVIDIA has 42 TPCs [46]; thus, we deduce that two TPCs were disabled in the V100 GPU that we evaluated.

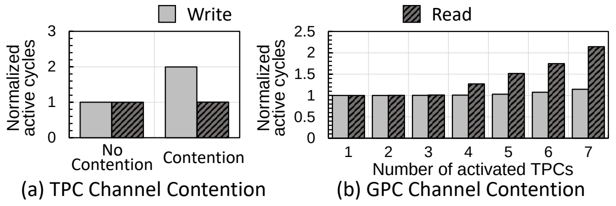

Figure 5: Performance impact of read and write memory accesses on (a) TPC channel and (b) GPC channel.

number of activated TPCs is relatively small for write requests. However, for read requests, TPCs do not limit the requests and thus, requests from 7 TPCs (14 SMs) contend for the GPC output channel and likely creates contention for the shared L2 resources as well as potentially contention during the reply path from the L2 back to the cores [8]. For up to 3 activated TPCs, there is a minimal impact from read requests on performance; however, we observe performance degradation for read requests with 4 or more activated TPCs. Specifically, when all 7 TPCs are activated, the performance degrades by 2.14×. In the proposed covert channel attack, we leverage the contention for memory write accesses in the TPC channel (or covert channel based on SMs within a TPC), while we exploit memory read accesses for the GPC channel (or covert channel based on TPCs within a GPC).

## 4 INTERCONNECT COVERT CHANNEL

In this section, we describe how we can exploit the understanding of GPU's on-chip network for interconnect covert channel. The proposed interconnect covert channel exploits local shared resources (i.e., interconnect channel) that are directly connected to the SMs. The physical interconnect channel contention itself is used to establish covert channel by measuring the impact of the contention on the memory access latency. We first describe how synchronization is done between the sender and the receiver in the proposed interconnect covert channel based on the GPU hardware clock register. We then describe the steps required to establish the two different types of covert channels – TPC channel and GPC channel. We also describe the impact of memory coalescing as well as how potentially multi-bit information can be communicated.

### 4.1 Synchronization

One challenge in any covert channel is the overhead for synchronization between the sender and the receiver, and its impact on overall throughput [21]. In this work, we identify how the `clock()` function available in NVIDIA GPUs can be exploited for synchronization as it returns the value of a 32-bit clock register that exists within each SM in NVIDIA GPUs [47]. To evaluate whether the clock register can be reliably used for synchronization, we ran a simple CUDA kernel to read the clock register from each SM in the Volta GPU – i.e., launch a thread block across all SMs that simply returns the value of the clock register from each SM using `clock()`. The returned clock value from the 80 SMs for a single run of the CUDA kernel is shown in Figure 6. The clock values from SMs within a TPC are nearly identical, shown by similar values of neighboring SMs. There can be significant variance in the values obtained from different TPCs (e.g., 4× difference between TPC0 and

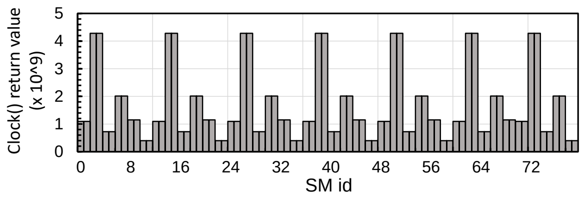

**Figure 6: Distribution of `clock()` return values across the different SMs in the Volta GPU.**

TPC1). [5] However, the TPCs within the *same* GPC have very similar values – e.g., all the SMs with a clock value of approximately $4 \times 10^9$ belong to the same GPC, SMs with a clock value of $2 \times 10^9$ belong to the same GPC, etc.

To understand the impact of system variance, we re-ran this kernel 100 times and the average difference between the clocks of the two SMs within the same TPC was under 5 clock cycles and the average difference between the clocks of two SMs within the same GPC was under 15 clock cycles. Although there is no guarantee that the clock registers of all SMs were read simultaneously in this experiment, we observe that the difference among the clocks within the same TPC (and the same GPC) is negligible compared to L2 access latency (~200-250 clock cycles). Hence, the raw values of the clock registers can be leveraged for synchronization.

## 4.2 Interconnect Channel Leakage

In Figure 7, we show the high-level block diagram of how '0' and '1' are communicated between two SMs for the covert channel. When the sender SM wants to communicate a '0', the sender does not inject any request into the on-chip network and the receiver SM is able to send its memory requests to L2 cache without contention, i.e., with lower latency. On the contrary, if the sender SM wants to transmit a '1', the sender SM injects a request into the network and contention occurs for the NoC injection channel. To avoid the impact of L1 hit/miss, L1 caching is disabled [47] [6] and all memory requests access data that is loaded into the L2 cache. The impact of the contention (or the lack of contention) is shown in Figure 7(b) and the receiver's requests injected into the network continuously increase. Thus, based on the change in the latency (and its impact on bandwidth usage), it enables covert channel to be established between the two cores and allows communication between the receiver SM and the sender SM.

The information leakage through this shared multiplexer (mux) or *external* concentration [31] is based on the two cores sharing the mux (or the output channel bandwidth). In addition, another assumption is that the arbitration for the mux is assumed to be *locally* fair – i.e., round-robin arbitration. We show the impact of the contention in the shared mux on the execution time by running a synthetic memory access workload that accesses L2 across two SMs under the following two scenarios – 1) SM0 and SM1 that are located in the same TPC and share a mux, and 2) SM0 and SM12 that are located in a different TPC and do not share a mux in an NVIDIA

[5] Note that TPC0 and TPC1 are not part of the same GPC.

[6] L1 caching can be disabled with with the *-dlcm=cg* compile option for *nvcc*. Disabling L1 caching is not necessary to create a covert channel but is used to improve the covert channel bandwidth and lower the error rate. In our evaluation, disabling L1 caching resulted in approximately 20% higher covert channel bandwidth.

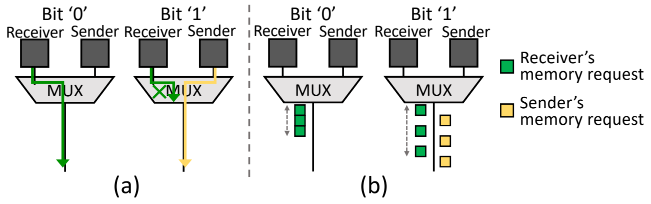

**Figure 7: High-level block diagram of communicating '0' or '1' through interconnect channel contention. (a) Contention in the interconnect bandwidth is shown when communicating '1' and (b) the impact on the latency is shown. The receiver continues to send request and measures latency.**

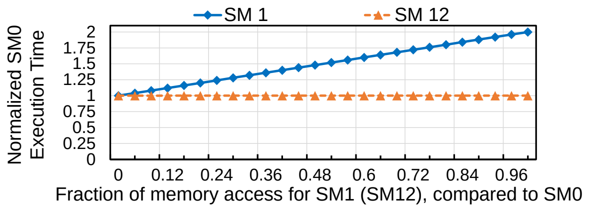

**Figure 8: Impact on SM0 execution time when executing it concurrently with SM1 (or SM12) on NVIDIA Volta GPU. The amount of memory access from SM1 (SM12) is varied ($x$-axis) and the normalized performance of SM0 is shown ($y$-axis).**

Volta GPU (see Figure 8). Here, we first measure the performance (or execution time) of SM0, which does multiple sequential writes to the memory that always results in an L2 hit (while bypassing L1). SM1 and SM12 also do sequential writes but the amount of writes issued to the memory is varied, as a fraction of the amount of memory access made by SM0. When the two SMs do not share the same mux (i.e., SM0 and SM12), the execution time of SM0 is relatively constant, regardless of the amount of memory access from SM12. However, when the two SMs share the same TPC (i.e. SM0 and SM1), the execution time for SM0 linearly increases as the amount of memory requests generated by SM1 increases - thus, the degree of contention for the interconnect channel can be directly observed by SM0.

## 4.3 Establishing Co-location on GPUs

To establish co-location for the proposed interconnect covert channel, the thread-block scheduling needs to be reverse-engineered. Prior work [42] has reverse-engineered thread block scheduling – however, their attack did not require knowledge of *exact* placement of SM across the TPC or the GPC hierarchy in the GPU. Since the interconnect covert channel relies on the exact understanding of the placement, we explore a more detailed thread-block scheduling understanding. In particular, our analysis shows that thread blocks are allocated by interleaving the thread blocks across the GPCs – however, within the GPC, the thread blocks are interleaved across the TPC first – i.e., before two thread blocks are allocated to the same TPC, all of the TPCs are first allocated a block. As a result, for the multi-TPC channel attack, allocating 40 thread blocks for the sender first and then, 40 thread blocks for the receivers results in the 40 threads blocks of the sender (or the trojan) allocated across all TPCs (occupying only 1 SM within each TPC) and the receiver (or the spy) is allocated to the other SM within the TPC.

**Algorithm 2:** Interconnect-based Covert Channel

```
$D_{send}[N]$, $D_{receive}[N]$: $N$ bit data to transmit and receive, respectively.
$T_{slot}$: Size of timing slot shared between sender and receiver.

Procedure Sender operations()
    Synchronization()
    for $i \leftarrow 0$ to $N$ do
        if $D_{send}[i] == 1$ then
            Access L2 cache    // Invoke channel-contention
        else
            Do nothing    // No channel-contention
        end

        busy waiting for remaining $T_{slot}$
        if $i \% Sync\_period == 0$ then
            Synchronization()
        end
    end
    return

Procedure Receiver operations()
    Synchronization()
    for $i \leftarrow 0$ to $N$ do
        $AccessTime[i]$ = Measured L2 access latency
        if $AccessTime[i] > Threshold$ then
            $D_{receive}[i] = 1$    // Channel is under contention
        else
            $D_{receive}[i] = 0$    // Channel is contention free
        end
        busy waiting for remaining $T_{slot}$
        if $i \% Sync\_period == 0$ then
            Synchronization()
        end
    end
    return
```

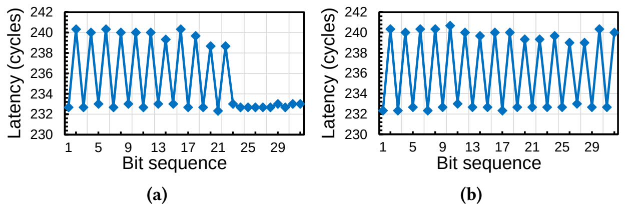

**Figure 9: Timing information measured through covert channel when '010101...' bit sequence is being commentated (a) with timing slot-only and (b) timing slot with local synchronization.**

## 4.4 TPC (Covert) Channel

For the covert channel, we leverage the clock register to achieve synchronization without any explicit handshaking or synchronization communication. A common challenge with using different clocks for synchronization is the clock skew – however, as shown earlier in Section 4.1, the clock skew between SMs in a TPC (or GPC) is relatively small. The high-level overview of the TPC covert channel is described in Algorithm 2. The proposed interconnect covert channel communication is based on a *timing slot* where a bit is sent every $T$ cycles and $T$ is agreed upon between the sender and the receiver. *Threshold* value is empirically determined based on the L2 latency of the GPU architecture. The timing slot of $T$ cycles is counted individually within each SM. During each time slot, the receiver sends a memory request (with a single warp) to L2 cache; the sender does nothing when communicating a '0' and sends write requests to L2 cache when communicating a '1.' To increase the impact of contention, we activated 5 warps for the sender. An example of communicating a bit sequence of '010101...' is shown in Figure 9 with a high latency value communicating a '1' and the low latency (no contention) communicating a '0.' For large values of $T$, no additional synchronization would be needed but this comes at the cost of reduced covert channel bandwidth. [7]

[7] The value of $T$ for the GPC channel described in the following section is set to a higher value since more communication needs to be done across more number of SMs.

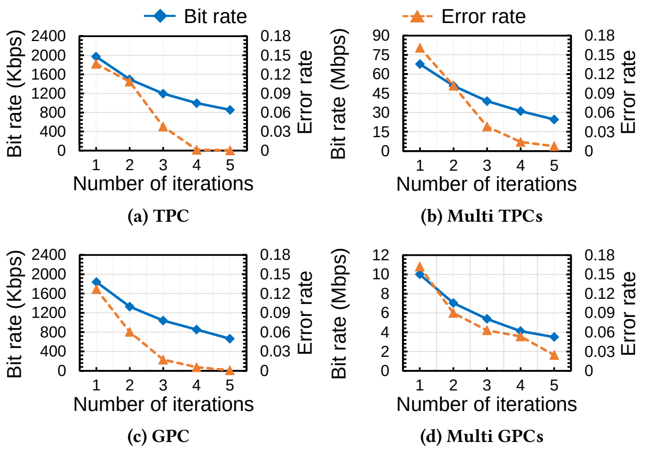

**Figure 10: Interconnect covert channel measurement on the Volta GPU with (a) single TPC channel and (b) multi-TPC channel on all 40 TPCs, (c) single GPC channel and (b) multi-GPC channel using all 6 GPCs. Number of iterations are the num. of memory operations used to communicate 1 bit.**

Thus, we use a value of $T$ that is slightly larger than the value of L2 access round-trip latency. However, errors can accumulate and result in effectively no-contention even when communicating a '1', as shown in Figure 9(a).

As a result, we complement timing slots with a synchronization after every $N$-bits. For the timing slot, both the sender and the receiver effectively waited for $T$-cycles before sending the next bit. However, after a synchronization period, we enforce the *Synchronization()* – where the lower $n$ bits of the clock registers are compared against a fixed value. Given the low clock skew, this effectively provides a "coarse" synchronization. With the added synchronization, any accumulated error will be "reset" and the covert channel can be established as shown in Figure 9(b).

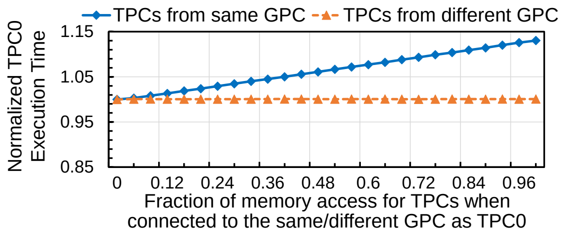

**Figure 11: GPC channel information leakage based on GPU-output interconnect channel contention.**

The results from TPC covert channel are shown in Figure 10(a,b) as the number of iterations is increased. In this work, we define the number of iterations as the number of memory operations used to create contention (or communicate one bit) or the number of memory accesses per warp – thus, higher number of iterations increases the probability of contention, resulting in lower error rate at the cost of lower bandwidth. With a single TPC channel, up to 1 Mbps covert channel bandwidth can be achieved with a near-zero error rate, using 4 iterations. With multi-TPCs, 5 iterations are needed to achieve negligible error while achieving ≈ 24 Mbps.

## 4.5 GPC (Covert) Channel

While TPC channel can provide very high bandwidth for covert channel, it requires the sender and the receiver to be co-located within the same TPC. In our evaluation using *cudaStream*, we were able to co-locate the sender and the receiver within the same TPC – and establish a covert channel based on TPC channel. However, if the sender and receiver cannot be co-located within the same TPC (e.g., by using a different thread block scheduler than the existing one), interconnect covert channel can still be established by exploiting the *GPC channel* as long as the sender and receiver are co-located within the same GPC. Similar to the TPC channel, we demonstrate a covert channel with GPC channel by using one TPC within a GPC as a receiver and the remaining TPCs as the sender in the GPC – i.e., one TPC is the receiver while the remaining TPCs send '1' (or '0') to the receiving TPC. Similar to SMs in TPC, clock register values of SMs within GPC have similar values (Sec 4.1), thus synchronization of receiver and multiple senders in GPC can be achieved similarly through `clock()` function. However, in the GPC channel, the sender sends read requests when transmits bit '1' while for the TPC channel, the sender utilizes write requests (Section 3.4).

The results from GPC covert channel are shown in Figure 10(c,d). With a single GPC, the bandwidth of approximately 800 kbps can be achieved with 4 iterations. The bandwidth that can be achieved with the GPC channel is lower than the TPC channel. We believe this is likely caused by the speedup (Section 2.3) in the GPC hierarchy. To overcome the bandwidth speedup, we made the sender in the GPC channel use more warps than the TPC channel – i.e., 8 warps were used for the GPC channel, and therefore, it had a higher *T* value. The information leakage from the amount of contention in the GPC channel is shown in Figure 11 – similar to TPC channel, the latency increases as more memory accesses are generated by the "sender." However, the change in latency (or the slope of the line) is much smaller than the TPC channel – suggesting that speedup

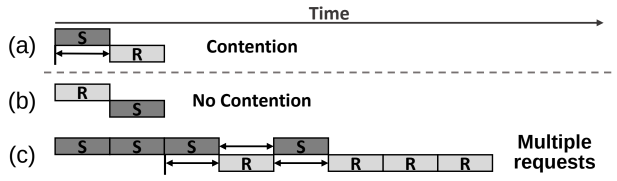

**Figure 12: Impact of multiple requests (or uncoalesced accesses) in creating contention. 'S' ('R') are the requests sent by the sender (receiver). (a) If sender and receiver are aligned, contention will occur but (b) if misaligned, the contention will not be detected. (c) Sending multiple requests ensures that the chance of contention is higher.**

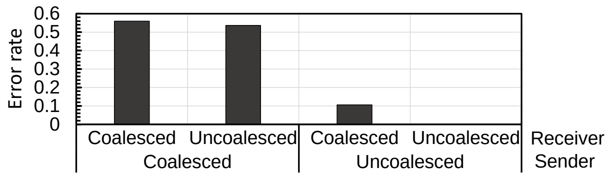

**Figure 13: Impact of memory coalescing on the overall error rate. If sender sends fully-coalesced memory accesses, then covert channel cannot be effectively established.**

reduces the impact of interconnect contention. Since there are only 6 GPCs in the GPU, multi-GPC results are lower than multi-TPCs, achieving slightly less than 4 Mbps and a more noticeable error rate of approximately 3%.

## 5 DISCUSSION

**Memory Coalescing:** While memory coalescing is commonly leveraged to improve memory bandwidth in GPUs, we ensure memory coalescing does *not* occur to reduce the error rate. If memory coalescing is used, the number of memory requests generated is significantly reduced and thus, the probability of contention is also greatly reduced. To avoid this and reduce the error rate, we ensure memory requests are *not* coalesced (i.e., multiple requests generated per warp) to ensure contention for interconnect occurs. An example of the impact of coalescing is shown in Figure 12. If there is only a single request from sender and receiver, contention can occur (Figure 12(a)) but if the receiver request is shifted or delayed, no contention would occur (Figure 12(b)) and '1' could not be properly communicated. However, if there are multiple requests from a warp (e.g., 4 requests as shown in Figure 12(c)), then even if the requests from the sender and receiver do not align, contention can still be observed. Thus, if all memory requests in a warp are uncoalesced, then 32 memory requests can be generated. Since the memory latency of a warp is determined by the latency of the *last* memory request of the warp that returns from the memory system, the covert channel is more error-tolerant.

In Figure 13, the error rate of the TPC channel is compared based on whether memory coalescing is used or not. Coalescing results in just 1 request per warp while uncoalescing creates 32 requests per warp. When the sender enables coalescing of the requests, the error rate exceeds 50% and it is clear that a covert channel cannot be established – regardless of whether the received enabled coalescing or not. On the contrary, when the sender's request is uncoalesced,

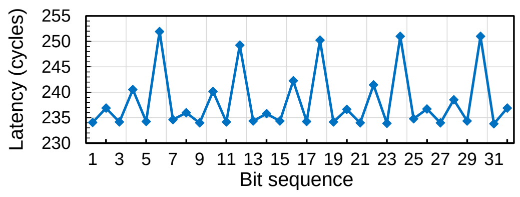

**Figure 14: Receiver latency when a sequence of '0102030102030...' is communicated.**

the error rate drops significantly. With the coalesced receiver, the error rate drops to approximately 10% but when both the receiver and the sender memory access are uncoalesced, the error rate is negligible at around 0.1%.

**Multi-level Channel Communication:** Similar to prior work on microarchitectural timing covert channels, a single bit was communicated at a time in this work – '0' for no-contention and '1' for contention. However, given the interconnect contention characteristics and its ability to *directly* measure contention, the ***degree of contention*** can be altered to enable a multi-bit covert channel by adjusting the amount of memory coalescing within a single warp. As described earlier, we assumed '0' is sent when there were no requests and '1' is sent with 32 requests (i.e., all requests within a warp are uncoalesced). However, the amount of coalescing can be varied to generate a multi-level covert channel – e.g., 2 bits of information can be communicated by generating memory requests with 0, 8, 16, or 32 unique memory requests from a single warp. In Figure 14, we show a measurement of latency when a sequence of '01020301..' is transmitted between the sender and the receiver – i.e., 2 bits of information are communicated with a single contention. The 4 possible values (for 2 bits) are communicated by creating contention with 0%, 25%, 50%, and 100% memory accesses from the sender. Our evaluations show that multi-level communication can result in approximately 1.6× increase in covert channel bandwidth – further increasing the amount of covert channel. However, higher bandwidth came at the cost of a higher error rate as it also increased proportionally.

**Other GPU Architectures:** Our evaluations in this work were demonstrated on NVIDIA Volta GPU and exploited the hierarchical organization (SM, TPC, GPC). We ran similar experiments on other NVIDIA GPU architectures, including Kepler and Pascal architectures as well as the more recent Turing architectures, and the same covert channels were confirmed on these architectures. All of the GPU architectures had a hierarchical network organization that shares interconnect bandwidth through concentration. The main difference (or challenge) across the different GPU architectures was reverse-engineering the GPU hierarchy (e.g., # of TPCs, # of SMs, etc.) and the thread block scheduling algorithm as they varied slightly. MIG (Multi-Instance GPU) [50] in Ampere A100 provides isolation between different users (instances) and each instance is allocated a separate GPC with a dedicated memory partition. However, MPS can still be executed within MIG and thus, the proposed covert channel is problematic with MIG since covert channels can be carried out if the spy and the trojan share the same instance. [8] We also ran similar experiments on a cloud GPU (i.e., AWS Volta GPU instance) and we were able to replicate the findings of this work on the cloud GPU. Collection of Compute Unit (CU) from AMD GPUs is also organized hierarchically [3, 18] but OpenCL does not provide the same interface as CUDA to reverse engineer AMD GPU NoC (e.g., determining core placement, `clock()` functionality, [9] etc.) – thus, we were not able to perform similar covert channels on an AMD system.

**Impact of Noise:** Noise is a concern in covert channel attack since it can reduce the covert channel bandwidth or increase the error rate. The biggest source of potential noise in our covert channel is the L2 cache (and the main memory). In our evaluation when we exploit all GPCs, we do introduce some noise because there is contention among L2 accesses from the different GPCs. This is shown in our evaluation as the error rate slightly increases by approximately 3% when using 6 GPCs compared to 1 GPC. Additional noise can be introduced if a third kernel (in addition to the spy and the trojan kernel) is co-located. Our covert channel is based on L2 accesses, and so if a third kernel shares the L2 capacity and causes the covert channel kernels to access the main memory, the noise from main memory accesses will become dominant and make the covert channel infeasible. However, in the covert channel, one goal of the attacker is to make the environment more "favorable" – thus, if all cores (or GPCs) are leveraged for the covert channel, not only can a higher covert channel bandwidth be achieved but any potential noise from the third kernel is eliminated. The attacker can easily manipulate the resource usage (e.g., local shared memory, register file, etc.) to ensure that co-location does not occur within SM to minimize the impact of any noise [42]. Interestingly, MIG [50] which has been proposed for QoS in Ampere GPUs can actually *help* to potentially minimize any noise from the system. Since each instance (or the different GPCs) are fully isolated with a dedicated path to its own memory partition, there is no opportunity for noise to be introduced by another instance.

**Side Channel Attack:** In this work, we focused on the covert channel caused by the GPU on-chip interconnect. However, since covert channel establishes leakage, it can potentially lead to other dangerous side-channel attacks [52]. An example of a simple side channel attack based on the leakage described in this work is using the NoC channel contention to measure the "amount of L1 miss" since there is a linear correlation between the NoC channel contention and the amount of L2 accesses (or L1 miss).

## 6 SECURE ARBITRATION

In this section, we discuss countermeasures to defend against the proposed interconnect covert channels. One potential solution is an alternative thread block scheduling to ensure that TPC (and GPC) channels are not shared. Temporal partitioning, similar to scheduling proposed in GPUGuard [70], can be used to ensure sender

[8] As described earlier, covert channels are effectively an "insider" threat and thus, sharing the same instance is a reasonable threat model in the cloud with MIG. We also confirmed that in AWS, multiple users can share the same instance as well [6].

[9] In recent AMD GPUs, `clock()` can also be accessed [4]. However, per-CU `clock()` are not synchronized and thus, cannot be used for synchronization. There is a globally synchronized clock available across the CUs but that clock is much slower (10-100MHz). Thus, this will cause significant degradation of covert channel bandwidth.

and receiver do not share the same TPC (or GPC). This prevents resource sharing and removes the possibility of the proposed interconnect covert channel as the TPC/GPC channels are no longer shared. However, this approach can lower the utilization of the SMs by reducing the amount of concurrent SMs that can be exploited during multi-kernel execution. Clock fuzzing [39] or introducing clock offset can be used to reduce the accuracy/precision of the clock and make synchronization using the clock more difficult; however, clock fuzzing does not necessarily remove the covert channel as alternative synchronization approaches can be explored. For example, handshaking on the interconnect channel (similar to prime+probe on L1 cache [42]) can be potentially leveraged for synchronization. SurfNoC [67] is a non-interference NoC [9] that can prevent covert channels but was proposed for a 2D mesh topology and is not applicable to hierarchical GPU NoC.

In this work, we exploit how interconnect arbitration can be leveraged to remove the proposed covert channel. As discussed earlier, one cause of the covert channel is the contention for the interconnect channel bandwidth and the locally-fair arbitration [15]. Instead of a locally-fair arbitration, a globally fair arbitration (e.g., age-based arbitration [1]) provides global fairness; however, such fairness does not mitigate the interconnect covert channels since requests that contend can be generated at a similar time or have a similar 'age'. We discuss alternative arbitration that can be potentially used to mitigate the proposed interconnect covert channel.

**Round-Robin (RR):** Baseline arbitration where the grant is given to the requester in an alternating, round-robin manner. If there is a request from one input but no request from the other inputs, the grant is given to the lone requester. However, this allows 'no-contention' to be observed by the receiver when the sender does not inject any packet.

**Coarse-grain Round-Robin (CRR):** Instead of arbitrating per packet, the arbitration can be done at coarse granularity or group requests from a warp together and perform *per-warp* arbitration. Each warp generates multiple requests which translate into multiple request packets into the on-chip network. As discussed earlier, enabling memory coalescing reduces contention and prevents covert channel from being established. Coarse-grain arbitration is effectively *network-coalescing* to minimize the amount of arbitration done — however, as shown in Figure 15, CRR by itself does not prevent the covert channel since communication of '1' results in contention and higher latency.

**Strict Round-Robin (SRR):** One potential mitigation is *strict* round-robin (SRR) arbitration or effectively providing *temporal partitioning* [63] through *secure* arbitration. Prior temporal network partitioning [63] statically partitioned on-chip resources such as virtual channels but in this work, we propose to statically partition interconnect bandwidth through arbitration. In the SRR arbitration, in contrast with RR arbitration, each node is granted access to the interconnect bandwidth, even if there is no request – effectively time-division multiplexing interconnect bandwidth. Thus, when 'no-contention' occurs, the arbitration would prevent the receiver from using the unused bandwidth and prevent covert channel from being established.

**Table 1: 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 | 1200MHz, Crossbar, $flit\_size$ = 40, $num\_vcs$ = 1, $subnet$ = 2 |

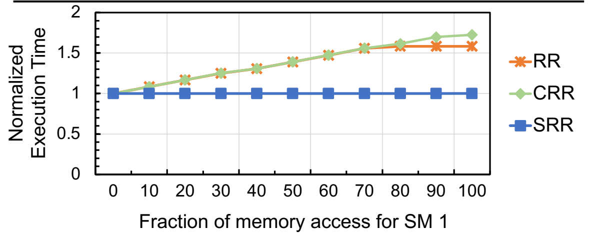

**Figure 15: Simulation comparison of arbitration algorithms.**

Using GPGPU-Sim simulator [30], we model a Volta-like GPU configuration, as summarized in Table 1. We modify the network arbitration within the BookSim [44] network simulator of GPGPU-Sim to evaluate the performance impact of the different arbitrations and the results are shown in Figure 15. Similar to the analysis from Section 4.2, we activate two SMs – (SM0 and SM1) and each SM has 2 warps allocated that continually make memory requests. The amount of memory requests from SM1 is varied and we measure the performance (execution time) of SM0. With the baseline RR arbitration, the performance of SM0 increases linearly (similar to what was shown earlier in Figure 8). We observe similar behavior for CRR as well – even though the number of arbitration is reduced, the total length of the data (or interconnect channel usage) does not increase (unlike memory coalescing) and contention still occurs. In comparison, the SRR approach completely mitigates the covert channel as the performance of SM0 is constant, regardless of the amount of memory accesses from SM1. While this can prevent covert channel, the performance degradation can be significant for other memory-intensive workloads – e.g., for the TPC channel, there can be up to 2× reduction in memory bandwidth from the SRR. For compute-intensive workloads, the performance degradation is negligible across all arbitration algorithms but for memory-intensive workloads, SRR results in significant performance degradation – up to 60% loss in performance. Thus, there is effectively some performance trade-off from the secure arbitration policy.

## 7 RELATED WORK

A high-level qualitative comparison of the different covert channels is summarized in Table 2. While there are many covert channels that have been previously proposed, we highlight a few representative covert channels, including high-bandwidth CPU covert channel that have targeted different shared hardware resources. [10] We categorize each covert channel based on the following characteristics to summarize the benefits from the proposed covert channel:

[10]Bandwidth comparison across different prior work is not necessarily fair since they leverage different architectures, different technologies, etc. For example, covert channel bandwidth of [42] can be higher with recent GPUs that have more parallelism.

**Table 2: Qualitative comparison with prior work on covert channels.**

| | Shared HW | Type of Covert Channel | | | Description | Synchronization | Error rate | Bandwidth |
|---|---|---|---|---|---|---|---|---|
| | | Parallel /Serial | Local /Global | Direct /Indirect | | | | |
| Wu et al. [68] | CPU Memory Bus | Parallel | Global | Direct | Cross-core covert channel through bus contention by utilizing atomic memory operation | Self-clocking coding (differential Manchester coding) | N/A | 38 Kbps |
| DRAMA [53] | DRAM Row Buffer | Parallel | Global | Direct | Cross-cpu covert channel by reverse engineering DRAM addressing schemes | Wall clock (for native runs); Transmit a clock signal (for VM runs) | 4.1% | 411 Kbps |
| Liu et al. [37] | CPU Last-Level Cache | Serial | Global | Indirect | Practical cross-core covert channel in virtualized environment | Asynchronous | <1% | 600 Kbps |
| | | | | | | | 22% | 1.2 Mbps |
| Gruss et al. [19] | CPU Shared Memory | Serial | Global | Indirect | Cross-core covert channel exploiting Flush+Flush technique | N/A | 0.84% | 3.9 Mbps |
| Sullivan et al. [62] | Memory order buffer | Parallel | Global | Indirect | Cross-thread covert channel with 4K-aliasing in GCE (Google Computer Engine) | N/A | <8.7% | 1.49 Mbps |
| Naghibijouybari et al. [42] | GPU L1 Cache | Serial | Local | Indirect | Prime+Probe type covert channel using L1 cache | Hand-shake based synchronization protocol | 0% | 285 Kbps (4.25 Mbps) |
| | GPU Functional Unit | Parallel | Local | Indirect | Funtional unit based covert channel exploiting SFU unit | N/A | N/A | 28 Kbps (1.3 Mbps) |
| | GPU Global Memory | Parallel | Global | Indirect | Global memory channel leveraging L2-level atomic operation | N/A | N/A | 41 Kbps |
| This work | GPU TPC Channel | Parallel | Local | Direct | Covert channel exploiting channel contention of interconnection network | Hardware clock register | ~0% | 1 Mbps (24 Mbps) |
| | GPU GPC Channel | Parallel | Local | Direct | | | ~0% (<3%) | 800 Kbps (4 Mbps) |

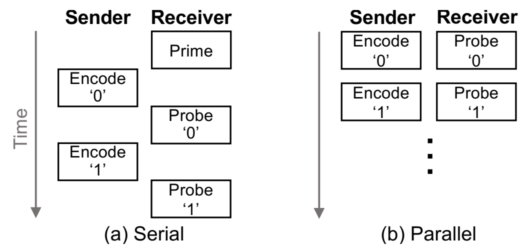

**Figure 16: (a) Serial and (b) parallel approach to establishing a covert channel.**

- **serial or parallel:** Many covert channels are often based on a serial approach (Figure 16(a)) (e.g., Prime+Probe approach); however, when the communication can be done in parallel (Figure 16(b)), higher bandwidth can be achieved.
- **local or global:** The shared resource used for the covert channel can be classified as either local or global. Global resources are often shared by multiple cores, have longer latency, and can be more susceptible to noise. In comparison, local resources are less susceptible to noise and in some cases, lower latency.
- **direct or indirect:** We classify if the shared resource contention behavior can be *directly* controlled or *indirectly* controlled. For example, the interconnect for TPC channel in this work is directly controlled by the core and only used for memory requests to L2 cache (and main memory). In comparison, the function units [42] in GPUs are local resource but are *indirect* since the contention for the functional units are impacted by pipelining and the scheduler [42].

The proposed covert channel in this work is able to achieve high bandwidth as it exploits *parallel* communication while leveraging a *local* shared resource that is *directly* controlled by the cores. No microarchitectural state needs to be modified in our attack and leveraging local `clock()` minimizes the synchronization overhead.

**Microarchitectural Covert Channels:** After first discussed by Lampson [32], many prior work have explored different types of covert channels. Microarchitectural covert channel attacks based on shared hardware resources including cache [19, 20, 37, 40, 56, 64, 71], hard drives [36, 56], branch predictors [17], and memory controllers [59] have been proposed. In particular, cache has been regarded as an attractive shared resource that is easy to exploit in the covert channel because its operation speed is very fast compared to other shared resources. Liu et al. [37] studied covert/side channel attacks under the cross-VM environment using LLC-based covert/side channel through prime+probe attack. Covert channels in the virtualized environment were demonstrated [68] by memory bus covert channel exploiting the atomic memory operations. To prevent covert channel attacks, different defenses mechanisms have been proposed. Yan et al. proposed ReplayConfusion [72] framework that detects unnatural contention through replay with program execution records. Liu et al. proposed Newcache that defends against cache-based attacks through randomized mapping of the cache [65]. There have also been recent covert channels based on on-chip interconnect. Paccagnella et al. [51] proposed cross-core side/covert channels that exploit the vulnerability of ring interconnect in Intel CPUs. Dutta et al. [16] demonstrated two types of covert channels exploiting contention of shared LLC cache and ring interconnect in a heterogeneous Intel’s integrated CPU-GPU systems. Compared to prior work, this work exploits the hierarchical on-chip interconnect in GPUs to achieve a high-bandwidth covert channel.

**GPU Covert/Side Channels:** Covert channel attack in GPUs was recently proposed [42], including intra-SM covert channel exploiting L1 constant cache and functional units, and inter-SM covert channel exploiting the contention in the L2 constant cache and the global memory. To prevent contention-based side-channel and covert channel attacks in GPUs, GPUGuard was proposed [70] that detects malicious behavior based on shared resource contention using a decision tree classifier. Countermeasures for intra-SM resources are not applicable for the covert channel proposed in this work since SM sharing is not required. As discussed earlier, thread block scheduling based on temporal partitioning can be leveraged as a countermeasure but comes at the cost of limiting the amount of concurrent kernels that can be executed.

Jiang et al. [23] identified *SIMT vulnerability* in which a positive correlation between the number of unique cache line requests and execution time can be exploited for AES key recovery. Karimi et al. explored hardware/software-based countermeasures to obfuscate the SIMT vulnerability [27] while RCoal [25] and BCoal [26] proposed alternative memory coalescing using randomized and bucketing-based memory coalescing to prevent SIMT leakage. Trident [5] identified how differences in modern GPUs with sectored-cache created *negative* correlation. Bank conflict in GPU shared memory has also been shown to enable AES key recovery [24]. Luo et al. [38] analyzed the difference in power consumption according to key information and succeeded in power analysis attack through power traces during AES encryption. Naghibijouybari et al. [43] showed that aggregating measures of contention through available resource tracking APIs can be used as attack surfaces on GPUs.

## 8 CONCLUSION

In this work, we proposed and demonstrated a novel microarchitectural timing covert channel in GPUs that exploits on-chip interconnect microarchitecture. We showed how the on-chip network of GPUs can be reverse-engineered and based on the understanding of the core placement, the shared interconnect and contention for network channels were exploited to create a covert channel. We demonstrated how it achieved significantly higher bandwidth compared to previously proposed covert channels and this is one of the first works that exploits on-chip interconnect microarchitecture to establish a covert channel with very high bandwidth.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and the shepherd for their invaluable comments. This work was supported in part by KAIST HRHRP program, NRF-2020R1A2B5B0100168711 and NRF-2021R1F1A1064009.

## REFERENCES

[1] Dennis Abts and Deborah Weisser. 2007. Age-Based Packet Arbitration in Large-Radix k-ary n-cubes. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. 1–11.

[2] Jacob T. Adriaens, Katherine Compton, Nam Sung Kim, and Michael J. Schulte. 2012. The Case for GPGPU Spatial Multitasking. In *2012 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1–12.

[3] Advanced Micro Devices, Inc. 2019. *Introducing RDNA Architecture The all new Radeon gaming architecture powering "Navi"*.

[4] Advanced Micro Devices, Inc. 2020. *"AMD Instinct MI100" Instruction Set Architecture Reference Guide.*

[5] Jaeguk Ahn, Cheolgyu Jin, Jiho Kim, Minsoo Rhu, Yunsi Fei, David Kaeli, and John Kim. 2021. Trident: A Hybrid Correlation-Collision GPU Cache Timing Attack for AES Key Recovery. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 332–344.

[6] Amazon Web Services, Inc. 2021. *Amazon Elastic Compute Cloud: User Guide for Linux Instances.*

[7] Mihir Awatramani, Joseph Zambreno, and Diane Rover. 2013. Increasing GPU Throughput using Kernel Interleaved Thread Block Scheduling. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 503–506.

[8] Ali Bakhoda, John Kim, and Tor M. Aamodt. 2010. Throughput-Effective On-Chip Networks for Manycore Accelerators. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, USA, 421–432.

[9] Travis H. Boraten and Avinash K. Kodi. 2018. Securing NoCs Against Timing Attacks with Non-Interference Based Adaptive Routing. In *2018 12th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. 1–8.

[10] Jie Chen and Guru Venkataramani. 2014. CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 216–228.

[11] Li-Jhan Chen, Hsiang-Yun Cheng, Po-Han Wang, and Chia-Lin Yang. 2017. Improving GPGPU Performance via Cache Locality Aware Thread Block Scheduling. *IEEE Computer Architecture Letters* 16, 2 (2017), 127–131.

[12] Jack Choquette. 2017. VOLTA: Programmability and Performance. https://www.old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.21-Monday-Pub/HC29.21.10-GPU-Gaming-Pub/HC29.21.132-Volta-Choquette-NVIDIA-Final3.pdf.

[13] Angelo Corana. 2015. Architectural Evolution of NVIDIA GPUs for High-Performance Computing. https://doi.org/10.13140/RG.2.1.1496.1042

[14] William J. Dally and Brian Towles. 2001. Route Packets, Not Wires: On-Chip Inteconnection Networks. In *Proceedings of the 38th Annual Design Automation Conference (DAC)*. 684–689.

[15] William J. Dally and Brian Towles. 2004. *Principles and Practices of Interconnection Networks*. Elsevier.

[16] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Márquez, and Kevin Barker. 2021. Leaky Buddies: Cross-Component Covert Channels on Integrated CPU-GPU Systems. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 972–984.

[17] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2015. Covert Channels through Branch Predictors: A Feasibility Study. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy* (Portland, Oregon) *(HASP '15)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages.

[18] Radeon Technologies group. 2017. AMD's Redeon Next Generation GPU Architecture. https://www.old.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.21-Monday-Pub/HC29.21.10-GPU-Gaming-Pub/HC29.21.120-Radeon-Vega10-Mantor-AMD-f1.pdf.

[19] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721* (San Sebastián, Spain) *(DIMVA 2016)*. Springer-Verlag, Berlin, Heidelberg, 279–299.

[20] Youngkwang Han and John Kim. 2019. A Novel Covert Channel Attack Using Memory Encryption Engine Cache. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[21] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding Contention-Based Channels and Using Them for Defense. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 639–650.

[22] Natalie Enright Jerger and Li-Shiuan Peh. 2009. On-Chip Networks, Synthesis Lectures on Computer Architecture. *Morgan & cLaypool publishers* (2009).

[23] Zhen H. Jiang, Yunsi Fei, and David Kaeli. 2016. A Complete Key Recovery Timing Attack on a GPU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 394–405.

[24] Zhen H. Jiang, Yunsi Fei, and David Kaeli. 2019. Exploiting Bank Conflict-Based Side-Channel Timing Leakage of GPUs. *ACM Trans. Archit. Code Optim.* 16, 4, Article 42 (Nov. 2019), 24 pages.

[25] Gurunath Kadam, Danfeng Zhang, and Adwait Jog. 2018. RCoal: Mitigating GPU Timing Attack via Subwarp-Based Randomized Coalescing Techniques. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 156–167.

[26] Gurunath Kadam, Danfeng Zhang, and Adwait Jog. 2020. BCoal: Bucketing-Based Memory Coalescing for Efficient and Secure GPUs. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 570–581.

[27] Elmira Karimi, Yunsi Fei, and David Kaeli. 2020. Hardware/Software Obfuscation against Timing Side-channel Attack on a GPU. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 122–131.

[28] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. 2011. GPUs and the Future of Parallel Computing. *IEEE Micro* 31, 5 (2011), 7–17.

[29] Mahmoud Khairy, Vadim Nikiforov, David Nellans, and Timothy G. Rogers. 2020. Locality-Centric Data and Threadblock Management for Massive GPUs. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1022–1036.

[30] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In

*2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486.

[31] Prabhat Kumar, Yan Pan, John Kim, Gokhan Memik, and Alok Choudhary. 2009. Exploring Concentration and Channel Slicing in On-chip Network Router. In *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. 276–285.

[32] Butler W. Lampson. 1973. A Note on the Confinement Problem. *Commun. ACM* 16, 10 (Oct. 1973), 613–615.

[33] Minseok Lee, Gwangsun Kim, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. 2016. iPAWS: Instruction-Issue Pattern-Based Adaptive Warp Scheduling for GPGPUs. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 370–381.

[34] Minseok Lee, Seokwoo Song, Joosik Moon, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. 2014. Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 260–271.

[35] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro* 28, 2 (2008), 39–55.

[36] Bartosz Lipinski, Wojciech Mazurczyk, and Krzysztof Szczypiorski. 2014. Improving Hard Disk Contention-Based Covert Channel in Cloud Computing. In *2014 IEEE Security and Privacy Workshops*. 100–107.

[37] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*. 605–622.

[38] Chao Luo, Yunsi Fei, Pei Luo, Saoni Mukherjee, and David Kaeli. 2015. Side-Channel Power Analysis of a GPU AES Implementation. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*. 281–288.

[39] Robert Martin, John Demme, and Simha Sethumadhavan. 2012. TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-Channel Attacks. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. 118–129.

[40] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2015. C5: Cross-Cores Cache Covert Channel. In *Proceedings of the 12nd International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2015)*. 46–64.

[41] David N. Muchene, Klevis Luli, and Craig A. Shue. 2013. Reporting Insider Threats via Covert Channels. In *2013 IEEE Security and Privacy Workshops*. 68–71.

[42] Hoda Naghibijouybari, Khaled N. Khasawneh, and Nael Abu-Ghazaleh. 2017. Constructing and Characterizing Covert Channels on GPGPUs. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 354–366.

[43] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered Insecure: GPU Side Channel Attacks Are Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2139–2153.

[44] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, and William J. Dally. 2013. A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 86–96.

[45] NVIDIA Corporation. 2016. *Nvidia Tesla P100 The Most Advanced Datacenter Accelerator Ever Built Featuring pascal GP100, the World's Fastest GPU.*

[46] NVIDIA Corporation. 2017. *Nvidia Tesla V100 GPU Architecture, The World's Most Advanced Data Center GPU.*

[47] NVIDIA Corporation. 2018. *CUDA C Programming Guide.*

[48] NVIDIA Corporation. 2018. *Nvidia Turing GPU Architecture, Graphics Reinvented.*

[49] NVIDIA Corporation. 2020. *Multi-Process Service.*

[50] NVIDIA Corporation. 2020. *NVIDIA Multi-Instance GPU User Guide.*

[51] Riccardo Paccagnella, Licheng Luo, and Christopher W. Fletcher. 2021. 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, 645–662.

[52] Colin Percival. 2005. Cache Missing for Fun and Profit.

[53] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *25th USENIX Security Symposium (USENIX Security 16)*. 565–581.

[54] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung, and Han Ratul Mahajan. 1985. Trusted Computer System Evaluation Criteria. In *National Computer Security Center*. Citeseer.

[55] Steve Rennich. 2011. CUDA C/C++ Streams and Concurrency. In *GPU Technology Conference.*

[56] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS '09)*. 199–212.

[57] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2012. Cache-Conscious Wavefront Scheduling. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 72–83.

[58] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2013. Divergence-Aware Warp Scheduling. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 99–110.

[59] Benjamin Semal, Konstantinos Markantonakis, Raja Naeem Akram, and Jan Kalbantner. 2020. Leaky Controller: Cross-VM Memory Controller Covert Channel on Multi-Core Systems. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. 3–16.

[60] Ankit Sethia, Davoud A. Jamshidi, and Scott Mahlke. 2015. Mascar: Speeding up GPU Warps by Reducing Memory Pitstops. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 174–185.

[61] William Stallings. 2006. *Cryptography and Network Security, 4/E*. Pearson Education India.

[62] Dean Sullivan, Orlando Arias, Travis Meade, and Yier Jin. 2018. Microarchitectural Minefields: 4K-Aliasing Covert Channel and Multi-Tenant Detection in Iaas Clouds.. In *NDSS*.

[63] Yao Wang and G. Edward Suh. 2012. Efficient Timing Channel Protection for On-Chip Networks. In *2012 6th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. 142–151.

[64] Zhenghong Wang and Ruby B. Lee. 2006. Covert and Side Channels Due to Processor Architecture. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. 473–482.

[65] Zhenghong Wang and Ruby B. Lee. 2008. A Novel Cache Architecture with Enhanced Performance and Security. In *2008 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 83–93.

[66] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous Multikernel GPU: Multi-tasking Throughput Processors via Fine-Grained Sharing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 358–369.

[67] Hassan MG Wassel, Ying Gao, Jason K Oberg, Ted Huffmire, Ryan Kastner, Frederic T Chong, and Timothy Sherwood. 2013. SurfNoC: A Low Latency and Provably Non-Interfering Approach to Secure Networks-On-Chip. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 583–594.

[68] Zhenyu Wu, Zhang Xu, and Haining Wang. 2012. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, Bellevue, WA, 159–173.

[69] Qiumin Xu, Hyeran Jeon, Keunsoo Kim, Won W. Ro, and Murali Annavaram. 2016. Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 230–242.

[70] Qiumin Xu, Hoda Naghibijouybari, Shibo Wang, Nael Abu-Ghazaleh, and Murali Annavaram. 2019. GPUGuard: Mitigating Contention Based Side and Covert Channel Attacks on GPUs. In *Proceedings of the ACM International Conference on Supercomputing* (Phoenix, Arizona) *(ICS '19)*. Association for Computing Machinery, New York, NY, USA, 497–509.

[71] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. 2011. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop* (Chicago, Illinois, USA) *(CCSW '11)*. Association for Computing Machinery, New York, NY, USA, 29–40.

[72] Mengjia Yan, Yasser Shalabi, and Josep Torrellas. 2016. ReplayConfusion: Detecting Cache-Based Covert Channel Attacks Using Record and Replay. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14.