Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU System

Sankha Baran Dutta
Pacific Northwest National Laboratory
Richland,WA, USA
sankha.b.dutta@pnnl.gov

Hoda Naghibijouybari
Department of Computer Science
Binghamton University
Binghamton, NY, USA
lhnaghibi@binghamton.edu

Arjun Gupta
Department of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM, USA
arjun@unm.edu

Nael Abu-Ghazaleh
CSE and ECE Departments
University of California, Riverside
Riverside,CA, USA
nael@cs.ucr.edu

Andres Marquez
Pacific Northwest National Laboratory
Richland,WA, USA
Andres.Marquez@pnnl.gov

Kevin Barker
Pacific Northwest National Laboratory
Richland, WA, USA
Kevin.Barker@pnnl.gov

ABSTRACT

The deep learning revolution has been enabled in large part by GPUs, and more recently accelerators, which make it possible to carry out computationally demanding training and inference in acceptable times. As the size of machine learning networks and workloads continues to increase, multi-GPU machines have emerged as an important platform offered on High Performance Computing and cloud data centers. Since these machines are shared among multiple users, it becomes increasingly important to protect applications against potential attacks. In this paper, we explore the vulnerability of Nvidia's DGX multi-GPU machines to covert and side channel attacks. These machines consist of a number of discrete GPUs that are interconnected through a combination of custom interconnect (NVLink) and PCIe connections. We reverse engineer the interconnected cache hierarchy and show that it is possible for an attacker on one GPU to cause contention on the L2 cache of another GPU. We use this observation to first develop a covert channel attack across two GPUs, achieving the best bandwidth of around 4 MB/s. We also develop a prime and probe attack on a remote GPU allowing an attacker to recover the cache access pattern of another workload. This access pattern can be used in any number of side channel attacks: we demonstrate a proof of concept attack that fingerprints the application running on the remote GPU, with high accuracy. We also develop a proof of concept attack to extract hyperparameters of a machine learning workload. Our work establishes for the first time the vulnerability of these machines to microarchitectural attacks and can guide future research to improve their security.

ACM Reference Format:
Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. 2023. Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU System. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589080

1 INTRODUCTION

GPUs have been an important computational platform enabling a variety of data intensive workloads such as deep neural networks, scientific kernels, cryptocurrency mining, and many others. The size of these workloads continues to increase: for example, training large deep networks often requires both computational and memory resources that far exceed those of a single GPU. In response to these trends, Multi-GPU platforms have emerged that offer tightly integrated GPUs, enabling applications that span multiple-GPU with unified memory accesses supported by fast communication fabrics. For example, the Nvidia DGX series [34] offers a number of server-class GPUs that are interconnected through a combination of proprietary high bandwidth interconnect (NVLink) and PCIe.

In this paper, we explore whether multi-GPU machines are vulnerable to both covert and side channel attacks. Given the importance of workloads that run on these machines, it is important to understand their security properties. On multi-GPU machines, multiple applications may concurrently execute to more effectively use the available resources. Applications generally belong to different mutually untrusting users. In our threat model, an application either covertly communicates with another (covert channel) or attempts to spy on them (side-channel). Covert and side channel attacks have been demonstrated on a variety of CPU microarchitectural structures [23, 27, 43, 51]. More recently, attacks have been demonstrated on GPUs as well [16, 17, 29, 30, 32].

This work demonstrates for the first time that microarchitectural covert and side channel attacks are also dangerous in the context of multi-GPU systems. Specifically, we first reverse engineer the caches in the context of multi-GPU systems, and discover that they are shared in a Non-Uniform Memory Access (NUMA) configuration: the L2 cache on each GPU caches the data for any memory pages mapped to that GPU's physical memory (even from a remote GPU). We also evaluate the impact of the NVlink topology on the remote access delays. These observations enable us to create contention on remote caches enabling the attacks. We build a covert channel attack, where a sender (trojan) process is located on one GPU transferring secret information to a receiver (spy) which is located on another GPU. We are able to obtain a high bandwidth, prime-and-probe covert channel, achieving a bandwidth of 3.95 MBps, with a low error rate of 1.3%. Using additional parallelism (e.g., involving additional GPUs) can further improve bandwidth, but we did not explore this in this paper. We believe that this type of channel applies more generally to other multi-accelerator systems that allow memory sharing across accelerators.

We also develop side channel attacks where an attacker process spies on another application executing on the same multi-GPU system. We demonstrate an application/kernel fingerprinting attack where the attacker tries to infer which application is running on a remote GPU. This attack will be useful as a first step in any other attack to determine where the victim kernels are running. We also demonstrate a simple model extraction attack that recovers the number of neurons in a hidden layer of a machine learning model [12, 30, 48].

Cross-GPU attacks offer the attacker a number of advantages compared to prior attacks targeting GPUs. First, they relieve the attacker from the issue of manipulating the scheduler on a single-GPU to establish co-location of the attacker kernels with the victim (e.g., on the same Streaming Multiprocessor (SM)) [30]. Also, in a cross-GPU attack, any noise generated by the attack code itself is isolated to the attacking GPU, which results in a higher-quality channel. In addition, isolation-based mechanisms such as Nvidia MIG [37] and partitioning-based [50] defense mechanisms that can be enabled for processes running within a single GPU can not protect against cross-GPU attacks (discussed in detail in Section 7). The attacks are conducted entirely from the user level without any special access (e.g. huge pages or flush instruction). As a result, we believe this attack model challenges assumptions from prior GPU-based attacks and significantly expands our understanding of the threat model in Multi-GPU servers.

In summary, the contributions of the paper are as follows:

- We reverse engineer the cache hierarchy and timing properties of the distributed NVlink-connected shared L2 cache in a multi-GPU environment from the user level.
- We demonstrate the first cross-GPU covert channel attack between a sender and a receiver on two different GPUs.
- We demonstrate two side channel attacks: (1) fingerprinting applications on the victim GPU; and (2) identifying the number of neurons in a hidden layer of a machine learning model.

2 BACKGROUND AND THREAT MODEL

In this section, we overview the organization of our attack target, the DGX-1 multi-GPU system from Nvidia. We also present the threat model, defining the assumptions we make about the attacker's access and capabilities.

2.1 Multi-GPU Systems

As neural networks grow deeper and training data sets become larger, the computational demands to train substantially exceed the capacity of a single GPU. Multi-GPU systems have emerged as an important platform for these and other workloads; they overcome the memory limitation of a single GPU and offer significant parallelism and interconnect and memory bandwidth. For large language models, often training takes hundreds of days on clusters with thousands of GPUs [31]. Nvidia's DGX series are being deployed in HPC clusters (e.g., [21]), as well as on cloud systems [26, 35, 44]. Other GPU manufacturers are also starting to offer similar products; for example, AMD's crossfire allows the building of relatively inexpensive multi-GPU configurations [4], and AMD's CDNA 2 architecture [5] is targeted towards HPC clusters [20]. It is likely that such systems will continue to grow in terms of the performance of the components (GPUs, interconnect, and memory) as well as in the number of GPUs that can be supported on each machine.

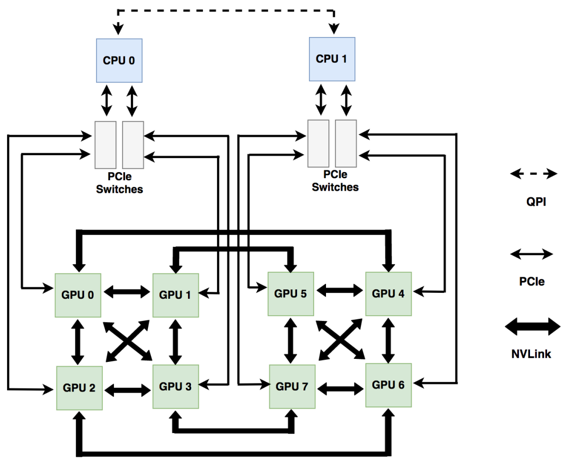


Figure 1: Nvidia DGX-1 topology

We develop the attacks in this paper on Nvidia's Pascal-based DGX-1 system [34]. Figure 1 shows the organization and network topology of this system. DGX-1 box consists of eight Tesla P100 GPUs, arranged in a hypercube fashion. DGX-1 also includes two CPUs (connected through QuickPath Interconnect (QPI)) for boot, storage management, and application coordination. The PCIe links between the GPUs and CPUs enable access to the CPU's bulk DRAM memory to enable data streaming to and from the GPUs. The GPUs are connected in a hybrid cube-mesh network topology, using Nvidia's proprietary NVLink interconnect. NVLink is an energy-efficient, high-bandwidth interconnect that enables Nvidia GPUs to connect to peer GPUs or other devices within a node at a bidirectional bandwidth of 160 GB/s per GPU: roughly five times that of current PCIe interconnections. The GPUs that are connected by NVlink can access each other's memories by using Nvidia provided CUDA APIs.

GPUs in DGX-1 box are Nvidia's Tesla P100 based on Pascal architecture (as shown in Figure 2). It consists of 56 SMs with a total of 3584 single-precision and 1792 double-precision units. Each GPU comes with 16 GB of High Bandwidth Memory (HBM2) stacked memory with 732 GB/s of bandwidth. There is a private 64KB shared memory per SM and a 4MB L2 cache shared across all SMs.

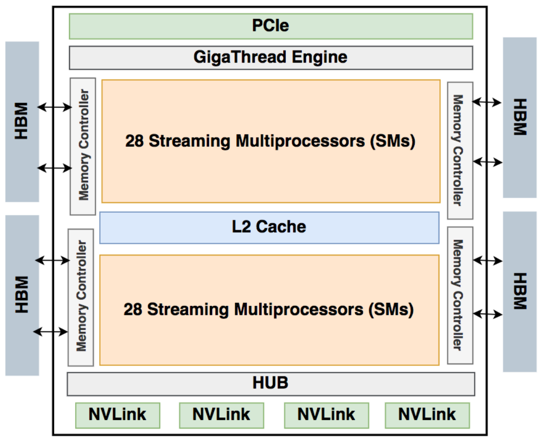


Figure 2: Pascal P100 GPU Architecture

2.2 Threat Model

In this paper, we develop Prime+Probe based microarchitectural covert and side channel attacks across multiple GPUs on Nvidia's modern GPU servers. Previous microarchitectural attacks were demonstrated on CPUs or on a single GPU. However, in our multi-GPU threat model, our attacks span across multiple GPUs that are connected via NVLink-V1(as shown in Figure 1). The sender or the victim process is located on a GPU (e.g. GPU 0) and the receiver process is located on another GPU (e.g. GPU 1). The involved GPUs are required to be connected by NVLink allowing peer-to-peer GPU memory access. Note that if both processes are located on the same GPU, then prior covert/side channel attacks on GPUs may be used [30]; however, in our threat model, the attacker does not need to co-locate on the same GPU with the victim and does not need to run a process on the victim GPU. Note that establishing the co-location on a single GPU is more challenging, as it requires reverse-engineering the scheduling processes, managing the noise, etc.

The attacker does not have access to any specialized system support or superuser privileges. They use experiments to reverse engineer the cache (one time, offline) and to find conflict sets, groups of addresses that hash to the same physical cache set, as a preliminary step of the attack. This step is necessary because caches are physically indexed, and sometimes use index hashing, making it difficult to determine the eventual set a virtual address will hash to.

3 REVERSE ENGINEERING CACHE ORGANIZATION

In multi-GPU system, a GPU can allocate and access directly the memory of a remote GPU that is connected via NVLink using Nvidia provided APIs. Our attacks are cache based timing attacks. However, the cache hierarchy and its properties are not well documented. For this reason, we reverse engineer the cache hierarchy, architecture, and its timing characteristics in this section. We conducted our timing experiments on two different DGX boxes, DGX-1 and DGX-2. Although both DGX boxes have the same topology as shown in Fig. 1, they are equipped with different GPU architectures. DGX-1 is equipped with Pascal P100 GPUs and DGX-2 with Volta V100. Our reverse engineering experiments demonstrate similar characteristics across the different architectures and DGX machines.

3.1 Caching organization and timing properties

In the first set of experiments, our goal is to understand the overall cache architectural details as well as the timing properties of different access types (hits vs. misses, local and remote). The DGX boxes offer a uniform address space, and virtual pages can be allocated to physical pages that belong to any of the GPU HBM DRAM memory (i.e., a NUMA organization). Both Pascal and Volta GPUs have two levels of data cache, L1 and L2. L1 cache is private to SM and L2 is shared among all the SMs. CUDA PTX [38] provides several loading primitives each with different caching properties.Loading primitives like *__ldca* allows the data to get cached in both L1 and L2 cache levels. Whereas, using loading primitives like *__ldcs* caches data with evict first policy and *__ldcv* loads data without caching and re-fetches the cache line on each new load. We have to use a loading primitive that caches the data at the L2 level only. A programmer can bypass L1 data caching by using specific data loading primitive (specifically, *__ldcg()*). However, L2 data caching cannot be bypassed and all data and instructions get cached in L2.

We discover that that memory locations are cached only in the cache of the GPU where the memory is allocated as we show later in this section. This is unlike, say multi-core systems, where data is cached at the core that accesses it. We believe that this choice is made to simplify cache coherence. However, as we show in the next section, this enables a dangerous remote prime-and-probe attack because the remote cache is shared. The attacker only needs to allocate memory on the remote GPU without running on that remote GPU. Remote memory allocation and remote memory accesses are supported in all multi-GPU systems to support Unified Virtual Memory, enabling large scale applications to share the available memory across the connected GPUs. As an example, the spy running on GPU 1 uses *cudaSetDevice* API to select GPU 0 as the location where memory is allocated (using *cudaMalloc* API). In the second step, another *cudaSetDevice* is called to change the current device to GPU 1, and in the last step *cudaDeviceEnablePeerAccess* is called to enable remote access from GPU 1 to 0. All these APIs are called from the spy process running on GPU 1 without special permissions.

To establish the timing properties, in the first experiment, we allocate a buffer in the memory of the local GPU to measure local access time. To measure remote access time, we allocate a buffer in the memory of a remote GPU, and we use *cudaDeviceEnablePeerAccess* to access the remote GPU's memory. Note that remote buffer allocation and accessing it does not create any context on the remote GPU.

To find both the remote and local access time, we first populate the L2 cache with the data from a buffer in DRAM with a stride of 128 bytes which is the L2 cache line size for our Pascal 100 GPU architecture. We use the *__ldcg()* load primitive to load the data which allows the data to get cached in the L2 cache only. Each data access is followed by a dummy operation to make sure the access is not optimized out by the compiler. The access time is measured using the clock() function and is recorded in a buffer in the shared (local) memory of SM to avoid any contention in the L2 cache as the access path of the shared (local) memory is separate from the main memory access path. This first cold access shows the DRAM access time. We access the buffer again and measure the access time which represents the L2 cache access time. Depending on the location of the allocated buffer (local or remote) with respect to the process launched, our reverse engineering process reveals the local or remote L2 hit and miss time.

The local and remote GPU L2 and DRAM access latency of Pascal P100 GPUs in the DGX-1 machine is shown in the histogram in Figure 3. We have made 48 accesses in each loop to measure both local and remote memory accesses. The X-axis specifies the access delay of the data and the Y axis specifies the number of bins in the histogram. As we can see in the figure, there are four clusters of accesses with respect to the timing, varying from just over 200 cycles to over 850 cycles. When examining the accesses, the fastest accesses (green in the figure) occur to cached accesses of the memory allocated and measurement kernel launched on the same GPU. The next group of accesses corresponds to local cache misses: DRAM accesses to the local HBM. The next two clusters correspond to cache hits and misses of the memory that is allocated on the remote GPU. It also provides us with timing thresholds to distinguish between cache hits and misses, to both local and remote GPU caches for different GPU architectures.

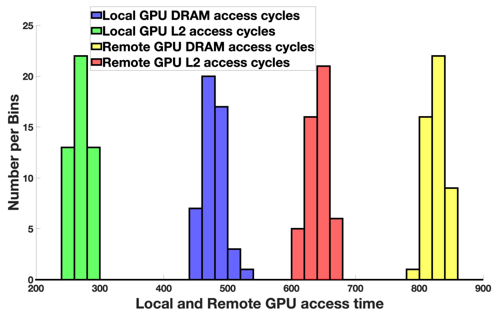
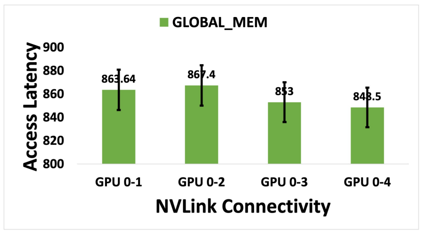


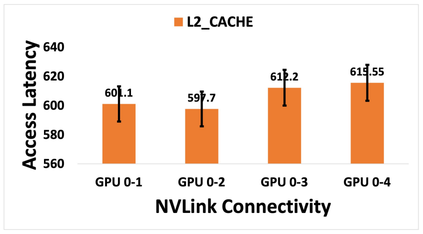
Figure 3: Local and remote GPU access time for DGX-1

We repeated the latency experiments from GPU 0 to the other GPUs to which it has a direct NVLink connection (as shown Figure 1) and we observed similar timing characteristics (Figure 4). Note that NVidia runtime API (*cudaDeviceEnablePeerAccess*) throws an error if the GPUs are not connected via NVLink.

According to the best of our knowledge, we are first to demonstrate the caching mechanism when the buffer and the kernel launched are on two different GPUs connected via NVLink in a multi-GPU setup. Thus, we understand the memory access pathways and caching as shown in Figure 5. When DRAM pages are allocated in the local GPU memory, the data access path is straightforward: the first access is serviced from the local HBM DRAM and subsequent accesses hit the L2 cache on the same device. On the other hand, when the data is allocated on one GPU and accessed from another, the request is routed through the NVLink connection and the requested cache line is also sent back through NVLink. Our experiment shows that this data accessed on the remote GPU



(a) DRAM Latency from GPU 0



(b) L2 Latency from GPU 0

Figure 4: Access Latency to different GPUs from GPU 0

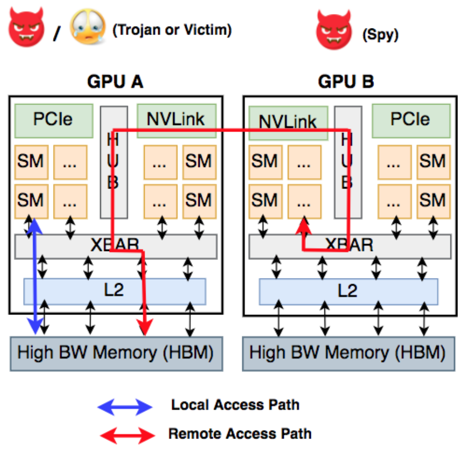


Figure 5: Accessing a remote GPU's memory through NVLink

is cached on the remote GPU, rather than on the local L2 GPU. Of course, caching the data locally would introduce cache coherence issues since copies of the same data could exist in multiple L2 caches. **In summary, our reverse engineering results demonstrate that an access to the memory of a remote GPU through NVLink is cached on the L2 cache of remote GPU, but not the L2 cache of the local GPU.** We use this shared remote L2 cache in GPU-to-GPU communication to build microarchitectural covert and side channel attacks. Figure 5 shows our threat model and the data path to access a remote GPU's memory through NVLink.

3.2 Determining Cache Eviction Sets

To conduct a successful Prime+Probe attack, an attacker needs to find a set of addresses that index into the same cache set. The number of addresses in this set should at least match the associativity of the cache, such that access to the set replaces current entries in that cache set; such a set is called an *eviction set*.

Although finding conflict sets is a standard component of prime and probe attacks, deriving these sets in our context is somewhat different. Many (but not all) CPU attacks benefit from additional features such as huge pages which substantially simplify the conflict set derivation. There are previous studies that explore the L2 cache architecture in the recent GPU architectures. In general, their assumptions are not compatible with our scenario where the attacker is attempting to find the eviction sets on the fly. Specifically, Mei *et al.* [28] explored different levels of memory hierarchy in GPUs. However, we could not use their attack directly because their reverse engineering process requires storing all the timer values in shared memory which significantly limits the number of samples we are able to take. Jia *et al.* [15] explored different memory levels of Volta and Pascal-based architectures. However, they did not provide detailed information about the reverse engineering of L2 architecture (and none for the multi-GPU scenario). Jain *et al.* [14] provided detailed information about the L2 reverse engineering as well as the architectural details. However, they modified the driver virtual to physical address translation to force consecutive allocation in the physical address space. Of course, this property does not hold under our threat model since modifying the driver requires privileged access. We use a pointer chasing experiment similar to traditional prime and probe attacks, but customized to the GPU. Moreover, since our attack is remote, we are able to substantially accelerate the attack and reduce the noise. Specifically, all memory used to store measurement values are on the attacker GPU, and therefore they do not generate noise that interferes with the target/remote victim cache. This enabled us to be more aggressive in deriving the conflict sets.

We observed that the derived eviction sets remain valid over application runs, saving us the cost of deriving them on the fly for every run. Instead, we check if the mapping holds, and if not rederive conflict sets on the fly which occurred rarely. However, this observation is surprising given that the cache is physically indexed and the virtual to physical memory mapping is likely to change across application runs. We conjecture that this may have to do with the way the physical memory allocator works on these systems. Specifically, the memory we allocate for the buffers to construct the conflict set are large: if consecutive physical memory pages get allocated, then the conflict sets in the virtual address space may remain the same but hash into different sets in the cache, keeping the conflict sets valid. Another effect that favors consecutive allocation is the use of coalescing TLBs [41]. Specifically, since the TLBs on modern GPUs are known to use coalescing to increase their reach, the allocator favors allocating aligned virtual addresses mapped to a consecutive runs of physical addresses to support coalescing and minimize the TLB pressure giving us the equivalent of huge pages, without having to explicitly use them. We verified experimentally that accesses within a single 32MByte buffer require only a single TLB miss, confirming the large range of the coalesced TLB entry.

The cache line size is 128B and from our eviction set determination experiment, we also learn the associativity of the cache (16). We observe that the target address is evicted after every 16th address reliably. This implies that there are 16 cache lines in the cache set.

Table 1: L2 cache architecture

Cache Attribute	Values
L2 cache size	4MB
Number of Sets	2048
Cache line size	128B
Cache lines per set	16
Replacement Policy	LRU

Also, the eviction pattern shows that the replacement policy is LRU (or pseudo-LRU) without randomization since the target address is evicted consistently after the 16th address. However, we have observed different caching patterns with different loading primitives. Through our reverse engineering experiment, we identified the loading primitive (*__ldcg()*) that allows the GPU L2 cache in the desired fashion. Table 1 summarizes L2 cache parameters and architecture derived from our reverse engineering experiments.

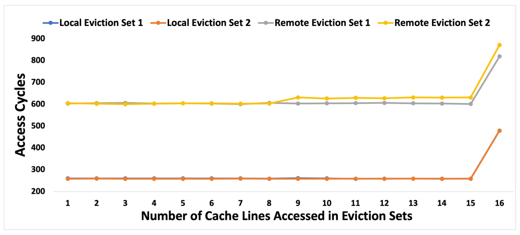


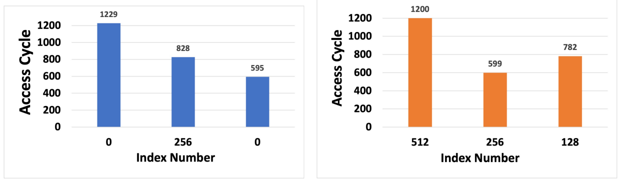
Figure 6: Validating the eviction set determination

Figure 6 shows an experiment we conduct for eviction set validation for two derived eviction sets on both the local and remote GPUs. The X-axis is the number of cache lines from the conflict set that have been accessed and the Y-axis is the access time in cycles. We observed that there is an eviction (increase in access time) after every 16th access. This behavior confirms the LRU-based replacement policy with a deterministic replacement for the eviction set access pattern.

The GPU L2 cache is physically indexed and the attacker does not have the knowledge of data placement in the cache. As a result, once we discover an eviction set, we are unsure whether it indexes into a new cache set or a previously discovered one. If we do not ensure that the eviction sets correspond to unique physical sets, this aliasing will result in noise during the attack. Specifically, there are two eviction sets determined by the malicious process that happen to index to the same physical cache set, due to the lack of knowledge of the address placement. If there are aliased cache sets within the same process, then during the actual attack phase, the eviction sets would cause interference due to self-eviction leading to the detection of a cache miss and misinterpreting as victim's access even when there wasn't one. Thus, it is important to test each discovered new eviction set against already discovered ones. If we notice misses when we combine more than 16 addresses from the two sets, we conclude that the two sets correspond to the same physical set and eliminate the newly discovered eviction set from consideration.

At the conclusion of this process, each process has discovered a collection of unique eviction sets ideally to cover the full cache. The reverse engineering results also provide the attacker with timing thresholds to distinguish between cache hits and misses, both on the local GPU as well as the remote GPU. With this information, we are ready to develop the end-to-end covert channel attack in the next section.

3.3 Other reverse engineering experiments



(a) Access latency from GPU 1 to 0 (b) Access latency from GPU 2 to 0

Figure 7: Reverse engineering locality of TLB

We also conduct another experiment to understand whether the TLB is also vulnerable to similar attacks. We discover that the page table entries, unlike data, are cached in the local TLB for each GPU. We allocate a buffer on GPU 0 (Figure 1) and launch first a kernel from GPU 1 and then another kernel from 2 sequentially. The kernel on GPU 1 accesses three memory addresses within the buffer: offset 0, offset 256 (a different cache line within the same page), and offset 0 again. We see in Figure7a the measured time for each access. The first access experiences a TLB miss and a cache miss. The second access experiences a TLB hit and a cache miss, while the final access experiences both a TLB and a cache hit. The subsequent kernel launched from GPU 2 accesses offsets 512, 256 and then 128. If the TLBs are cached remotely at GPU 0 (as with the data), then the first access would be TLB hit and a cache miss. However, the results in Figure 7b show a similar time to the first access from GPU 1 corresponding to a TLB miss and a cache miss. The second access to address 256, results in both a TLB hit and a cache hit since the data is cached at GPU 0 after it was accessed by GPU 1.

Finally, to show that the attack generalizes beyond the DGX-1 machine, we repeated the reverse engineering experiment on the DGX-2 with Volta V100 GPUs. The local and remote access time have similar timing characteristics with different threshold values, and therefore the attack is possible in this environment as well.

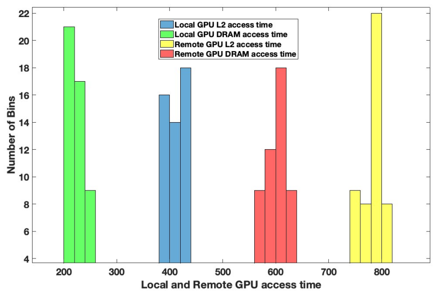


Figure 8: Local and remote GPU access time for DGX-2

4 COVERT CHANNEL ATTACK AND CHALLENGES

Having established the caching organization and timing characteristics, in this section, we develop a covert channel attack across two GPUs. Previous GPU-based microarchitectural attacks were demonstrated within a single GPU, and the majority use aggregate measures of contention such as performance counters. Besides establishing this new threat model, the attack has advantages over a single GPU attack: it bypasses defenses focused on a single GPU, it reduces the noise, and it avoids having to work around the scheduler to co-locate the two kernels within the GPU so that they can establish contention (e.g., on the same SM [29]). The attack is conducted from user level and does not require any system level features such as huge pages or flush instructions that are necessary for many attacks.

As demonstrated in Fig. 5, the the sender of the covert channel is located on a local GPU, GPU A, and the receiver is located on the remote GPU, GPU B, and accesses the memory of the GPU A to synchronize and receive information sent by sender process on GPU A(the opposite is also possible). These two processes communicate covertly over the shared L2 cache of GPU A. First, the receiver primes a cache set. To communicate "1", the sender would access its own data, evicts the receiver's data, and fills up the cache set, and to communicate "0", the sender process does nothing. The receiver process keeps probing the same cache set and records the access time. A high access time indicates a miss and interpreted as "1" and a low access time indicates a hit and is interpreted as "0". Although the overall attack process is similar to traditional Prime+Probe attacks, there are several unique challenges that arise due to the platform. We describe these challenges and our approach to overcoming them next.

4.1 Aligning the cache sets

At this stage, the two processes are able to derive exclusive eviction sets covering the L2 cache . However, all we are able to determine is that each set hashes to the same physical cache set, but not to which set. To be able to communicate, the processes have to use eviction set pairs, one in each process, that hash to the same physical cache set. We develop a protocol to enable the processes to discover and agree on the sets to use for the signaling and communication as shown in Figure 9.

Assume again that the sender process is located on GPU A and the receiver process has been launched on GPU B and they are connected via NVLink, shown in Fig. 5. Both processes allocated their buffer on GPU A and share the L2 cache on that GPU. In this scenario, the sender is a local process and the receiver is remote. In a single run of the malicious applications, one sender eviction set is checked with another eviction set of the receiver process. In Fig 9, we can see a local sender process eviction set TE_A launched on GPU A and the remote receiver process launched on GPU B have three eviction sets SE_A, SE_B, and SE_C. The eviction set of the local sender process is checked against three eviction sets of the receiver process that could be located in the same physical cache set X. The set matching experiment reveals that the sender eviction set TE_A is not mapped to the receiver eviction set SE_A and SE_B shown by

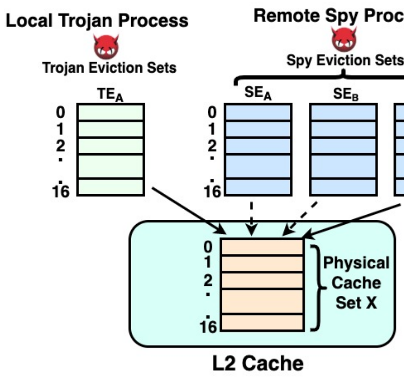


Figure 9: Eviction set alignment for multiple processes

Algorithm 1: Eviction set alignment across processes

```
for $i = 0;\ i < numMainLoop;\ i = i + 1$ do
    idxTemp = startIdx;
    timer1 = 0;
    dummy1 = 0;
    for $i = 0;\ i < numOfCacheLines;\ i = i + 1$ do
        dataPtr = &mainBuff[idxTemp];
        start = clock();
        idxTemp = __ldcg(dataPtr);
        dummy1+=idxTemp;
        end = clock();
        timer1+=(end-start);
    end
    timer2+=(timer1/numOfCacheLines);
    dummy operation
end
timeBuffMain = (timer2/(numMainLoop));
```

dotted arrows. But the sender eviction set TE_A is mapped to the receiver eviction set SE_C.

The eviction set mapping kernel is shown in Algorithm 1. The eviction set is accessed from lines 5 - 12 and the number of access is equivalent to the number of cache lines specified by *numOfCacheLines* (which is 16 in our case). A single eviction set is accessed for *numMainLoop* number of times in 1. The actual access of the data takes place in line 8 and the first index is specified in line 2 and gets initialized every time within the outer loop. The access takes place in a pointer chase fashion within the inner loop. The access cycles are measured and kept in a register variable *timer1* which accumulates the single access of the eviction set. Another register variable *timer2* in line 13 accumulates the average access time of a single access of the eviction set. Finally, all the accesses over the outer loop are averaged in line 16. The kernel algorithm is the same for both the sender and receiver processes. The only difference between them is the number of outer loops that decides how many times a cache set would be probed. The sender process has faster access compared to the receiver process as the memory is local to the sender process. So the value of *numMainLoop* is much higher for the sender process compared to the receiver process. For our work, we have selected a value of 400000 and 150000 for the local sender and remote receiver process respectively. However, these probing values can be reduced to optimize the execution time of the set mapping process. The main target is to create a visible contention in the L2 cache set and the loop boundary controls that contention.

Note that this particular challenge is required in the covert channel only to communicate between two malicious processes. For side channel, only finding the unique cache sets satisfies the purpose.

4.2 Putting it together: Covert channel

The sender process (launched on GPU A) allocates the data buffer on the same GPU in step 1 and the receiver process gets launched on another GPU (B), but allocates the buffer on the remote GPU A, where the sender process is launched. The first access of both the sender and the receiver process get the data from the off-chip GPU DRAM and get cached in the L2 cache. The subsequent memory accesses will be serviced from the L2 cache of GPU A.

The overall flow of the covert channel attack is shown in Figure 10. The sender is located on GPU A and the receiver is located on GPU B. Steps 1 and 2 of the attack represent the determination of the eviction sets of both processes. These are followed by the alignment step (Step 3 in the figure) to map the sets on each side to the same physical cache sets, which now enables them to communicate by creating or withholding contention on these sets.

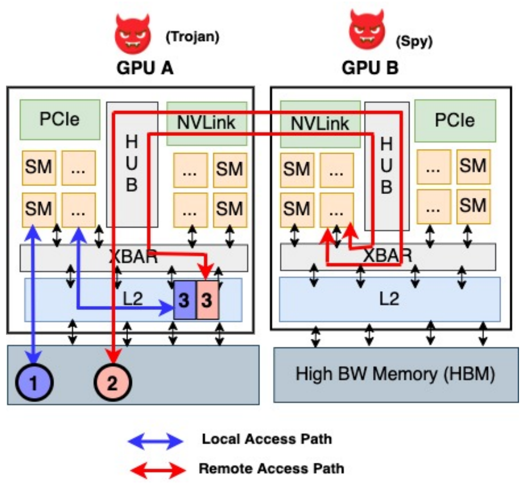


Figure 10: Cross GPU covert channel attack

From the previous step of cache set alignment, we have been able to determine the cache sets that are mapped among the malicious processes. This allows us to select the cache sets that would be used during the covert channel communication process. For each cache set, we have allocated a thread block that would be launched to an SM in the GPUs. Hence, when the communication takes place over a single cache set, a single thread block on both sender and receiver sides would access their own eviction set whose mapping was determined from the set aligning step. We leveraged the GPU parallelism by increasing the number of thread blocks. Each thread block, in both sender and receiver,would access different eviction sets that are already mapped to have a faster communication. The sender thread block consists of a single warp of threads (32 on our machines). All 32 threads in a thread block of the sender process are involved in probing the cache set. The 16 addresses referring to the 16 cache lines in the eviction set are accessed through pointer chasing similar to the eviction set determination technique.The receiver process essentially also has 32 threads that are active in the attack; however, we use a significantly higher number of threads (1024) and use the additional threads to help efficiently save the recorded times from the buffer in shared memory to global memory when it fills. Storing the access cycles temporarily on the shared buffer and then copying to the main buffer reduces memory pressure as well as increases the parallelism during the data copy. To send a "1" the sender process accesses the cache set, replacing the data placed there by the receiver,and does nothing to send a "0". We have used controlling parameters that control the priming of the cache set while sending a "1", and use computationally heavy dummy instructions (e.g. trigonometric instructions) to wait during transmitting "0" to the spy process. The spy process, however, continuously probes the cache set to receive the data from the sender process.

4.3 Covert Channel Evaluation

In this section, we evaluate the multi-GPU covert channel attack. All experiments use CUDA 10.0 with Nvidia driver version 410.79. For the covert channel evaluation, we send a long message across the GPUs using L2 cache sets. We send a message of size 1Mb across the covert channel. We vary the number of cache sets we use in the attack. Fig. 12 shows a demonstration of the transmission of the first part of a message. Specifically, the X-axis of the figure is the time progression and the Y axis is the access cycles. The message shows the first line in the text,(*"Hello! How are you? "*) in the long message that has been transferred covertly. The y-axis shows the timed access cycles measured from the remote receiver as it accesses the cache set. We observe that the number of cycles is 630 while sending '0' and 950 cycles while communicating '1'. To synchronize the communication, as the sender and the receiver processes are located on different GPUs, we tune parameters on the sender side that controls the cache access frequency to communicate the covert message successfully to the spy side.

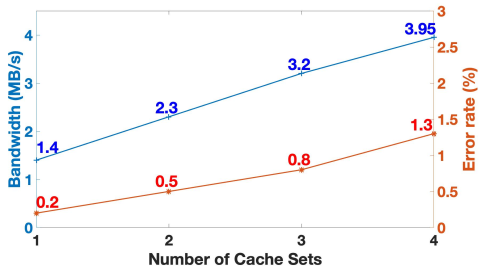


Figure 11: Bandwidth and Error rate in covert channel

The bandwidth and the error rate are shown in Fig. 11. We have measured the bandwidth and error rate as we increase the number of sets used for communication on the x-axis of the figure. The left y-axis of the figure is the bandwidth corresponding to the blue line in the figure which is displayed in MB/s. Similarly, the right y-axis shows the error rate in percentage corresponding to the red line. We measured the bandwidth and the error rate measured over 1000 runs of sending the message from sender to receiver.The bandwidth increases as the number of cache sets increases since we are able to communicate over multiple cache sets in parallel. However, as the number of cache sets increases, the contention increases among resources such as ports, introducing more variability in the timing, and increasing the error rate increases as well. **The highest bandwidth is 3.95 MB/s when using 4 cache sets, with an average error rate of 1.3% measured over 1000 runs. Using additional sets improves bandwidth but results in higher error rates.** Table 2 shows an error and bandwidth comparison between previously demonstrated covert channels over various hardware resources to show our channel's significance. In fairness, we note that it is difficult to compare these numbers directly since they were obtained on different architectures (CPUs vs. GPUs, as well as generation of each).

Table 2: Bandwidth of prior covert channels

	Shared Resource	Error %	Bandwidth
Wu et al. [49]	CPU memory Bus	N/A	38 Kbps
DRAMA [40]	DRAM Row Buffer	4.1%	411 Kbps
Liu et al. [23]	CPU Last-level cache	22%	1.2 Mbps
Gruss et al. [11]	CPU Shared Memory	0.84 %	3.9 Mbps
Naghibi et al. [29]	GPU L1 cache	0%	285 kbps
Ahn et al. [2]	GPU NoC	0%	1 Mbps
Leaky buddies [9]	CPU Last level cache	2%	120 Kbps
This work	multi-GPU L2 cache	1.3%	3.95 MBps

5 SIDE CHANNEL ATTACK

In this section, we demonstrate proof-of-concept side channel attacks. The attacks start by probing a remote cache constructing a *memorygram* of the accesses to the cache, which is a collection of cache hits and misses for the different cache sets over time [45]. This access pattern correlates with the activity on the remote GPU, allowing us to infer information about the applications running on this GPU.

Attack 1: Application Fingerprinting: This attack identifies a running application based on its memorygram. We envision this attack to serve as a first step of future attacks where we identify a target application running on a GPU and then infer information about it. In our proof-of-concept attack, we used six different applications from the NVidia toolkit[36] as our victim applications. The applications include common HPC workloads such as vectoradd, histogram, blackscholes, matrix multiplication, quasirandom and welshtransform. The memorygram results are shown in Fig. 13 which monitors 256 L2 cache sets over time priming and probing each: the x-axis represents the time progression in probing iterations and the y-axis is the cache set number. A yellow dot represent a cache miss on the corresponding L2 cache set at the corresponding time, indicating a likely victim application's access. From the images it is evident that each victim application leaves a unique memory footprint. The memory footprint pattern remains fairly similar across runs despite potential changes of virtual to physical mapping, which enables the classifier to accurately detect the application.

We train an image classifier to identify the different applications based on input memorygram images (other approaches are possible). Specifically, we run the attack many times against the different applications to collect 1500 samples for each application. We split

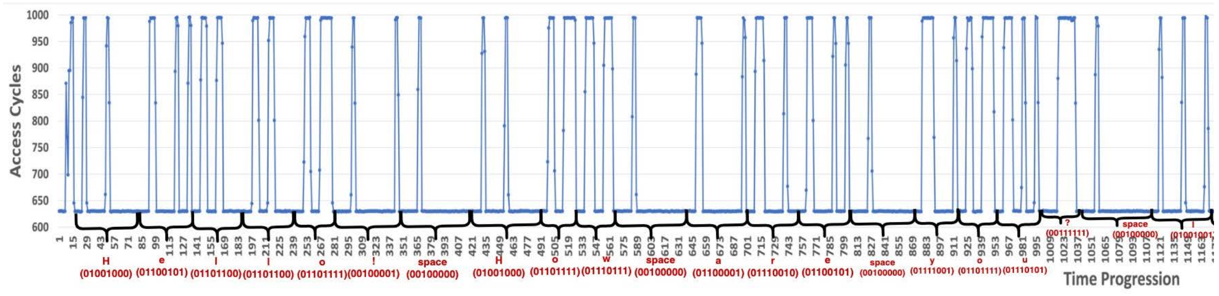


Figure 12: Cross GPU covert message received by spy process

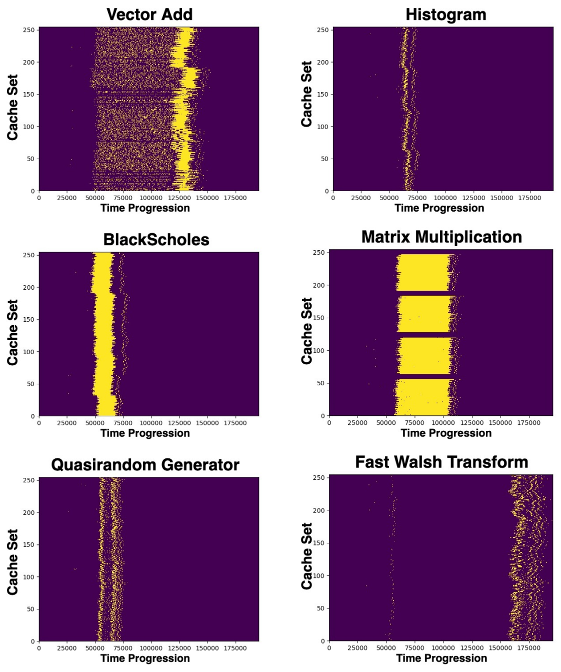


Figure 13: Memorygram of 6 applications

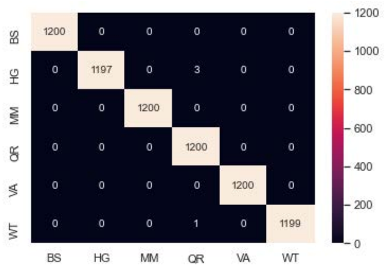


Figure 14: Confusion Matrix. BS (Black Scholes), HG (Histogram), MM (Matrix Multiplication), QR (Quasi Random), VA (Vector Addition), WT (Walsh Transform)

the data into training and validation sets of 150 samples each and isolate 1200 samples as the test or control set. Since there is no class imbalance in the data set, keeping a sufficiently large test set ensures that we evaluate the generalization capabilities with good confidence.

Table 3: Average misses over all cache sets

Number of Neurons	Average Number of Misses
64	5653
128	6846
256	8744
512	10197

The classifier achieved an overall accuracy of 99.91% on the test set of 7200 samples spanning six classes. Black Schole, Matrix Multiplication, Quasi Random Generator, and Vector Addition were classified with a perfect accuracy score of 100% while Histogram and Welsh Transform scored 99.75% and 99.91% respectively. The confusion matrix depicting the classification results is shown in Figure 14. We believe the formulation can be readily extended to classify a larger number of applications, and eventually extended to identify specific kernels within an application. This will enable us to use this attack as a first step to locate the kernels of a long running application and then carry out side channel attacks targeting them individually.

Attack 2: Attacking a Deep Learning Application: Machine learning training and inference is perhaps the primary application envisioned for multi-GPU machines. We demonstrate a preliminary side channel targeting extraction of model information from a machine learning model as it executes.

Our victim application is a Multi-layer Perceptron (MLP) model with 1 hidden layer built using PyTorch [39]. The application trains the MLP using the MNIST digit recognition data set[7]. We used four different network configurations varying the number of neurons in the hidden layers; the attack's goal is to identify this number of neurons. We monitored 1024 unique L2 cache sets in the remote GPU. We chose this number to balance sampling coverage and the speed of the attack (how often we can sample each set). A histogram of the number of misses for each of the monitored cache sets is shown in Fig. 15. Visually, we can see that the intensity of misses increases as the size of the hidden layer increases, reflecting the additional computations during training.

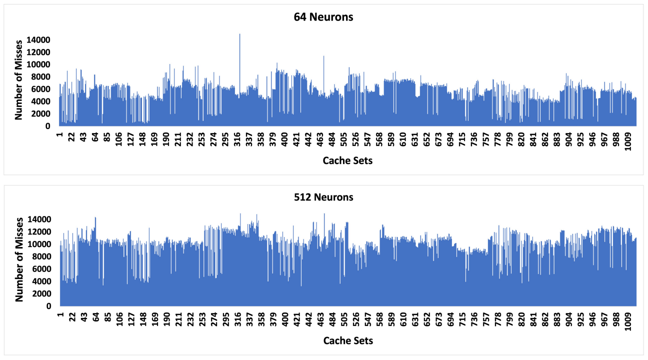
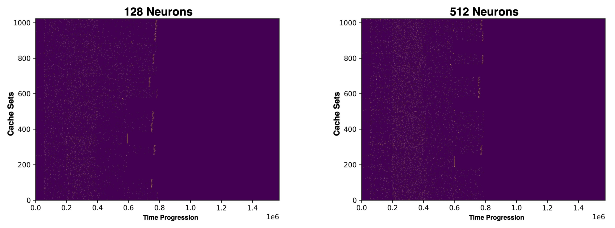


Figure 15: Cache misses per set



(a) 128 neurons (b) 512 neurons

Figure 16: Memorygram of the MLP application

Table 3 shows the average number of cache set misses; we see separation which allows us to infer the configuration. Figures 16a and 16b show the memorygram with 128 and 512 neurons respectively. Memorygram data is richer, showing the pattern of misses over time. For example, the model was configured to run two epochs which we are able to infer visually in Figure 17. We also verified that the spy is able to obtain the memorygram for an application launched through OpenCL; although we did not construct an end-to-end attack on OpenCL, we believe extracting sensitive information from the memorygram is possible as we have demonstrated for CUDA workloads so far.

6 NOISE MITIGATION

We developed our attacks in a quiet environment. However, in real scenarios, there will potentially be other concurrent applications running on GPUs, accessing the L2 cache, and as a result, adding noise to the covert or side channel attacks.

For mitigating noise, we propose to leverage concurrency limitations of GPUs using similar approaches as prior work [29] to force exclusive execution of receiver or sender on GPUs. Based on the leftover policy for GPU multiprogramming, thread blocks of the first process are assigned to different SMs and if there are leftover intra-SM resources for other applications, they can get launched on the same SM concurrently. These resources include shared memory, register, and the maximum number of thread blocks per SM. For example, in covert channel attacks, if we control the resource demand of our sender on GPU A and receiver on GPU B to saturate the intra-SM resources, no other concurrent application can be assigned to those SMs on two GPUs during the covert communication. Of course, this approach is more difficult for side channels, but it is likely that we would be able to customize a kernel to block out additional noise from the GPU with knowledge of the resources needed by the target victim application. The attack uses one thread

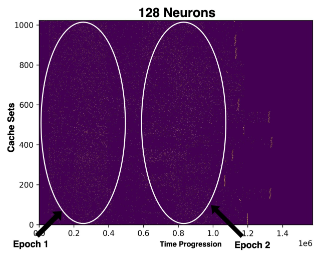


Figure 17: Memorygram for a two-epoch experiment

block per SM. However, each thread block can only allocate 32Kb of shared memory on Pascal, which is half the size of the available shared memory per SM. To consume the shared memory and block other applications, we launch idle thread blocks to use the leftover shared memory without interfering with the attack (they do not access the global memory during the communication). Therefore, we can ensure the exclusive execution of receiver (or sender) on GPU reducing noise.

7 POTENTIAL MITIGATIONS

A variety of microarchitectural covert and side channel defense mechanisms have been studied on CPU-based systems [8, 19, 22, 47] and recently within a single GPU [50]. A possible defense to prevent such attacks is the isolation of concurrent processes into different security domains. The security domains can be defined by partitioning of resources which eliminates the interference of applications on shared hardware and mitigates these attacks.

As an example, Nvidia designed Multi-Instance GPU (MIG) Technology [37] in the latest generation GPUs (Ampere). In this design, GPU can be securely partitioned into separate GPU instances for multiple users with an isolated path through the entire memory system. Although MIG was primarily introduced for performance improvement, it can potentially be used for process isolation against microarchitectural attacks. We explored whether MIG can provide protection against our application fingerprinting attack in multi-GPU systems. MIG isolates the contexts/processes that are executing within a single GPU into several hardware instances and supports several predefined options for slicing the compute and memory resources of the device. In its current design, MIG does not support Multi-GPU systems [37] (that is, it cannot even be enabled) and even if it could, it is not able to separate remote memory accesses originating from the contexts running on other GPUs through direct interconnect such as NVLink. In addition, even if it can be used, MIG is a static partitioning approach that can significantly degrade the performance of multi-GPU systems.

Xu et al. [50] propose GPUGuard, a contention detection and dynamic partitioning framework to defend against covert and side channels in discrete GPUs. GPUGuard enables dynamic isolation of contending processes while ensuring fine-grained GPU sharing and maximum utilization, providing a nice trade-off between performance and security. However, similar to MIG, GPUGuard focuses on applications running on a single GPU and is not compatible with multi-GPU systems.

Current partitioning-based designs could potentially be extended to support the efficient isolation of remote memory accesses of other GPUs. Direct cache access in current multi-GPU systems is offered to significantly increase the efficiency of GPU-to-GPU communication for distributed and cooperative workloads. An ideal partitioning-based defense for multi-GPU systems must have minimal impact on the local and remote memory accesses of cooperating processes, and at the same time eliminate the interference of other contexts. This can limit or prevent interference between mistrusting applications sharing the system, but a practical design raises a number of issues which we hope to explore in future work.

A potential mitigation is to consider limiting remote memory allocation. However, sharing memory across GPUs is essential to the workloads enabled by multi-GPU systems where multiple GPUs work together on a large problem such as machine learning or inference. Allocating and accessing remote GPU memory is achieved by Nvidia provided APIs at the user level, and are essential to unified virtual memory which is used in most large scale applications. For example, Nvidia is executing machine learning benchmarks like MLPerf [42] in the DGX A100 SuperPods [33] to harness the multi-GPU capability and accelerate both training and inference. Limiting memory to be used just on the GPU where it is allocated breaks all but embarassingly parallel multi-GPU applications. Researchers are also using multi-GPU servers to study and accelerate graph anaytics [18, 52] applications. In addition, new system optimizations are emerging to improve the performance of multi-GPU systems that rely on remote GPU memory access. For example, Choi et. al. [6] propose reclaiming memory available in multi-GPU systems to allocate to applications that need them. Thus, we consider such a mitigation to be impractical, although it is perhaps interesting to explore whether policies can be developed to detect and limit unusual allocation patterns.

8 RELATED WORK

With the increasing support of multiprogramming on GPUs in recent years, several works have studied microarchitectural covert and side channel attacks on a single GPU. Naghibijouybari et al. [29] characterize contention and construct covert channels on a variety of resources on GPUs, including constant caches, different types of functional units, and memory. Nayak et al. [32] develop a similar microarchitectural covert channel on GPU's shared last level translation lookaside buffer(TLB). Ahn et al. [3] implement covert channel attacks on shared on-chip interconnect on GPUs. In a completely different environment, Dutta et al. [9] developed covert channel attacks between CPU and GPU through shared LLC and ring bus in integrated CPU-GPU systems.

Another line of research times a GPU kernel from the CPU. GPU kernels experience timing variations due to data-dependent memory coalescing [1, 16] or shared memory bank conflicts [17]. These variations, measured through timing the kernel or by collecting hardware performance counters [46], Electromagnetic traces [10, 12] or power consumption traces [25], can be used to infer secrets about the GPU kernel (typically an encryption key). GPU side side channel attacks that use the performance counters were also demonstrated to do, website fingerprinting, inter-keystroke timing attack [30], workload fingerprinting [24, 53], or Neural Network model extraction attacks [30, 48].

All of these attacks on discrete GPUs exploit the aggregate measures of contention on GPUs. The attacks that we develop in this paper, are the first Prime+Probe based timing attacks on L2 cache on GPUs, which focus on a single set of cache, providing high-resolution attacks by fine-grained access time measurements. Our attacks also span multiple GPUs in multi-GPU systems, bypassing possible partitioning based mechanisms within a single GPU [37, 50].

Irazoquie et al. [13] explore cross core side channel leakage in multi-core CPUs. Like our attacks, these span across different processing units. The attack is substantially different than ours in that it requires shared memory between the attack and the victim and relies on the cache coherence protocol. Our paper may be viewed to expand our understanding of cross processor attacks to the case where the processors do not share a coherence domain.

9 CONCLUDING REMARKS

In this paper, we demonstrate for the first time a microarchitectural attack on Multi-GPU systems. These systems are emerging and increasingly important computational platforms, critical to continuing to scale the performance of important applications such as deep learning. They are already offered as cloud instances offering opportunities for an attacker to spy on a co-located victim. We reverse engineer the cache organization and sharing on an Nvidia DGX-1 machine, showing that remote caches can be shared when the attacker allocated memory on the memory banks of the remote GPU. We reverse engineer the timing properties of both local and remote accesses, as well as the cache replacement policy. We develop both prime-and-probe based covert channel and side channel attacks across different GPUs.

We believe that other attacks may be possible on multi-GPU systems. Coarse-grain cache occupancy attacks [45] are possible since even finer grained attacks are possible. We showed that TLBs are not remotely cached and therefore are not vulnerable to our attack model. We also conjecture that attacks that detect contention on NVlink may be possible, perhaps assisted with access to performance counters. We hope to study these attacks, as well as mitigations in our future work.

10 ACKNOWLEDGEMENTS

This work was supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under awards 66150: "CENATE - Center for Advanced Architecture Evaluation" and 76125: “AMAIS - Advanced Memory to support Artificial Intelligence for Science” . The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830. The work was also partially supported by US National Science Foundation grants CNS-1955650 and CNS-2053383.

REFERENCES

[1] 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. https://doi.org/10.1109/HPCA51647.2021.00036

[2] 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-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 565–577. https://doi.org/10.1145/3466752.3480093

[3] 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-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 565–577. https://doi.org/10.1145/3466752.3480093

[4] AMD. 2017. AMD CrossFire guide for Direct3D® 11 applications.

[5] AMD. 2022. Introducing AMD CDNA™ 2 Architecture. https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf.

[6] Sangjin Choi, Taeksoo Kim, Jinwoo Jeong, Rachata Ausavarungnirun, Myeongjae Jeon, Youngjin Kwon, and Jeongseob Ahn. 2022. Memory Harvesting in Multi-GPU Systems with Hierarchical Unified Virtual Memory. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 625–638. https://www.usenix.org/conference/atc22/presentation/choi-sangjin

[7] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[8] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 1–21.

[9] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, 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. https://doi.org/10.1109/ISCA52012.2021.00080

[10] Yiwen Gao, Hailong Zhang, Wei Cheng, Yongbin Zhou, and Yuchen Cao. 2018. Electro-Magnetic Analysis of GPU-Based AES Implementation. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 121, 6 pages. https://doi.org/10.1145/3195970.3196042

[11] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 279–299.

[12] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. 2020. 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* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 385–399. https://doi.org/10.1145/3373376.3378460

[13] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cross processor cache attacks. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*. 353–364.

[14] Saksham Jain, Iljoo Baek, Shige Wang, and Ragunathan Rajkumar. 2019. Fractional GPUs: Software-based compute and memory bandwidth reservation for GPUs. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 29–41.

[15] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. 2018. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826* (2018).

[16] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2016. A complete key recovery timing attack on a GPU. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. IEEE, Barcelona Spain, 394–405. https://doi.org/10.1109/HPCA.2016.7446081

[17] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2017. A Novel Side-Channel Timing Attack on GPUs. In *Proceedings of the on Great Lakes Symposium on VLSI (VLSI'17)*. 167–172. https://doi.org/10.1145/3060403.3060462

[18] Seunghwa Kang, Alex Fender, Joe Eaton, and Brad Rees. 2020. Computing PageRank Scores of Web Crawl Data Using DGX A100 Clusters. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–4. https://doi.org/10.1109/HPEC43674.2020.9286216

[19] Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, and Huiyang Zhou. 2009. Hardware-Software Integrated Approaches to Defend Against Software Cache-based Side Channel Attacks. In *Proceedings of the International Symposium on High Performance Comp. Architecture (HPCA)*.

[20] Oak Ridge National Laboratory. 2022. Systems. https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html.

[21] Pacific Northwest National Laboratory. 2016. Nvidia DGX-1 housed in PNNL's campus. https://www.pnnl.gov/science/highlights/highlight.asp?id=4431.

[22] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby Lee. 2016. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.

[23] 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*. IEEE, 605–622.

[24] S. Liu, Y. Wei, J. Chi, F. H. Shezan, and Y. Tian. 2019. Side Channel Attacks in Computation Offloading Systems with GPU Virtualization. In *2019 IEEE Security and Privacy Workshops (SPW)*. 156–161.

[25] Chao Luo, Yunsi Fei, Pei Luo, Saoni Mukherjee, and David Kaeli. 2015. Side-Channel Power Analysis of a GPU AES Implementation. In *33rd IEEE International Conference on Computer Design (ICCD'15)*. https://doi.org/10.1109/ICCD.2015.7357115

[26] Tobias Mann. 2020. Amazon Amps Up Cloud With Nvidia A100s. https://www.sdxcentral.com/articles/news/amazon-amps-up-cloud-with-nvidia-a100s/2020/11/.

[27] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2015. C5: cross-cores cache covert channel. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 46–64.

[28] Xinxin Mei and Xiaowen Chu. 2016. Dissecting GPU memory hierarchy through microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (2016), 72–86.

[29] 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.

[30] 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. https://doi.org/10.1145/3243734.3243831

[31] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters. *CoRR* abs/2104.04473 (2021). https://arxiv.org/abs/2104.04473

[32] Ajay Nayak, Pratheek B., Vinod Ganapathy, and Arkaprava Basu. 2021. (Mis)Managed: A Novel TLB-Based Covert Channel on GPUs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (Virtual Event, Hong Kong) *(ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 872–885. https://doi.org/10.1145/3433210.3453077

[33] Nvidia. [n. d.]. https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks/.

[34] Nvidia. 2017. NVIDIA DGX-1 System Architecture White Paper.

[35] Nvidia. 2020. GPU-Accelerated Google Clouds, Google Cloud Anthos on NVIDIA DGX A100. https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/google-cloud-platform/#:~:text=NVIDIA%20DGX%20A100%20is%20the,NVIDIA%20GPUs%20within%20Google%20Cloud.

[36] Nvidia. 2021. Nvidia cuda samples. https://docs.nvidia.com/cuda/cuda-samples/index.html.

[37] Nvidia. 2021. Nvidia Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[38] Nvidia. 2021. Parallel Thread Execution ISA. https://docs.nvidia.com/cuda/pdf/ptx_isa_7.6.pdf.

[39] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[40] 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)*. USENIX Association, Austin, TX, 565–581. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl

[41] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. 2012. Colt: Coalesced large-reach tlbs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 258–269.

[42] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee,

Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2019. MLPerf Inference Benchmark. arXiv:1911.02549 [cs.LG]

[43] Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. 2021. Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 1077–1090.

[44] Cirrascale Cloud Services. 2022. Bringing NVIDIA DGX A100 to the Cloud. https://cirrascale.com/platforms-nvidiadgx-a100.php.

[45] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2019. Robust website fingerprinting through the cache occupancy channel. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 639–656.

[46] Xin Wang and Wei Zhang. 2020. An Efficient Profiling-Based Side-Channel Attack on Graphics Processing Units. In *National Cyber Summit (NCS) Research Track*, Kim-Kwang Raymond Choo, Thomas H. Morris, and Gilbert L. Peterson (Eds.). Springer International Publishing, Cham, 126–139.

[47] Zhenghong Wang and Ruby B. Lee. 2007. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.

[48] Junyi Wei, Yicheng Zhangy, Zhe Zhou, Zhou Liy, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing Deep-learning Model Secret with GPU Context-switching Side-channel. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (Valencia, Spain).

[49] 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. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/wu

[50] 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)*. ACM, New York, NY, USA, 497–509. https://doi.org/10.1145/3330345.3330389

[51] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 719–732.

[52] Heng Zhang, Lingda Li, Donglin Zhuang, Rui Liu, Shuang Song, Dingwen Tao, Yanjun Wu, and Shuaiwen Leon Song. 2021. An Efficient Uncertain Graph Processing Framework for Heterogeneous Architectures *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 477–479. https://doi.org/10.1145/3437801.3441584

[53] P. Zou, A. Li, K. Barker, and R. Ge. 2019. Fingerprinting Anomalous Computation with RNN for GPU-accelerated HPC Machines. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. 253–256.