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

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

Hoda Naghibijouybari
Department of Computer Science
Binghamton University
Binghamton, New York, USA
hnaghibi@binghamton.edu

Arjun Gupta
Independent Contributor
arjun.gupta@gmail.com

Nael Abu-Ghazaleh
CSE and ECE Departments
University of California, Riverside
Riverside, California, 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. As these machines are shared between 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 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 3.95 MB/s. We also develop a prime and probe attack on a remote GPU allowing an attacker to recover the cache hit and miss behavior of another workload. This basic capability 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. Our work establishes for the first time the vulnerability of these machines to microarchitectural attacks, and we hope that it guides future research to improve their security.

I. 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 continue to increase: for example, training of 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 fabric. For example, the Nvidia DGX series [27] offers a number of server class GPUs that are interconnected through a combination

of proprietary high bandwidth interconnect (NVLink) and PCIe. Other GPU manufacturers are also starting to offer similar products; for example, AMD's crossfire allows building relatively inexpensive multi-GPU configurations [4]. 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.

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 [19], [22], [31], [37]. More recently, attacks have been demonstrated on GPUs as well [14], [15], [24]–[26].

Our 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 on 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). This observation enables us to create contention on remote caches by allocating memory on the target GPU, which is the essential ingredient enabling our covert and side channels. Specifically, we develop the first **microarchitectural covert and side-channel attacks across GPUs in a multi-GPU servers (an Nvidia DGX-1 server)**. In the covert channel attack, a trojan process is located on one GPU transferring secret information to a spy which is located

1

on another GPU. In our side channel attacks, the malicious process can monitor the shared L2 cache from a remote GPU and infer secrets about the victim process.

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 SM) [25]. In addition, these attacks also bypass isolation-based defenses such as partitioning-based [36] defense mechanisms that can be enabled for processes running within a single GPU. Moreover, previous side-channel attacks on a single GPU exploit the aggregate measures of contention on GPUs [14], [15], [20], [25], [35], [38]. The attacks that we develop in this paper, are the first Prime+Probe based timing attacks on L2 cache on GPUs. Our attacks extract contention information at the granularity of a single cache set, providing high-resolution attacks with fine-grained access time measurements, reducing the noise, and achieving high quality channels. 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.

The attacks also require resolving a number of new challenges that are specific to this environment and the userspace only nature of the attack. To develop an attack over the L2 cache we have to reverse engineer the cache architectural details from user space. Usually system level assistance like using huge pages [11] [19] and cache flush instructions provides additional information during reverse engineering, which are unavailable to us. We first reverse engineer the sharing of the caches discovering that they are configured such that each physical page gets cached at the GPU connected to the memory where it is placed. Thus, a GPU can remotely access the caches of other GPUs. We determine the timing characteristics corresponding to access times under different caching scenarios, and use them to develop eviction sets – collections of memory addresses hashing to the same cache set– both from local and remote GPUs. For the covert channel attack, the next challenge is to align the different discovered eviction sets such that the contention is created at the same physical set from both processes. Without this alignment, the two sides cannot know which eviction sets to use to cause the contention necessary for the attack. We solve all of the challenges above, enabling a high quality, high bandwidth, prime-and-probe covert channel across GPUs, 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 also explore developing side channel attacks. The attack relies on recovering the *memorygram* [19] of the accesses on the cache, and then inferring information about the victim from the distribution of the cache hits and misses. Specifically, 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 attacks which recovers the number of neurons in a hidden layer of a machine learning model [10], [25], [35].

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

- We reverse engineer the cache hierarchy and timing properties of the shared L2 cache in a multi-GPU environment from the user level. The reverse engineering allows us to understand the architectural details of the L2 cache in multi-GPU environment in this modern AI boxes.
- We identify and overcome challenges that arises in building covert and side channel attacks in a multi-GPU environment. These challenges include: finding conflict sets for the different cache sets from each process and then aligning the conflict sets across the two processes.
- We demonstrate the first cross GPU covert channel attack on the shared L2 cache as the attack medium. With a trojan process on one GPU and a spy process on another, we construct a reliable high-bandwidth covert channel.
- We demonstrate side channel attacks where we construct the memorygram of the accesses of a remote victim. We use the memorygram in two attacks: (1) fingerprinting applications on the victim GPU; and (2) identifying the number of neurons in a hidden layer of a machine learning model.

II. BACKGROUND AND THREAT MODEL

In this section, we overview the organization of the DGX-1 multi-GPU system from Nvidia which we use as the basis for our attacks and evaluations. We also present the threat model, defining the assumptions we make about the attacker access and capabilities.

A. Multi-GPU Systems

The demand for high performance computing in deep learning is rapidly growing. As neural networks grow deeper and training data sets become larger, the computational demands to train substantially exceed the capacity of a single GPU, for example requiring weeks to train a large DNN. Therefore, Multi-GPU systems are introduced to provide high throughput and high interconnect bandwidth to maximize neural network training performance. Multi-GPU systems overcome the memory limitation of a single GPU and offer significant parallelism, and interconnect and memory bandwidth. For example, in 2019, Nvidia used 1,472 V100 interconnected GPUs to bring down the training of a BERT network to 53 minutes [1]. Other throughput bound applications could also scale their performance using multiple GPUs.

We develop the attacks in this paper on a Nvidia's Pascal-based DGX-1 system [27]. Figure 1 shows the organization and network topology of this system. DGX-1 box consists of eight Tesla P100 GPUs. DGX-1 also includes two CPUs (connected through QuickPath Interconnect (QPI)) for boot, storage management, and application coordination. The PCIe
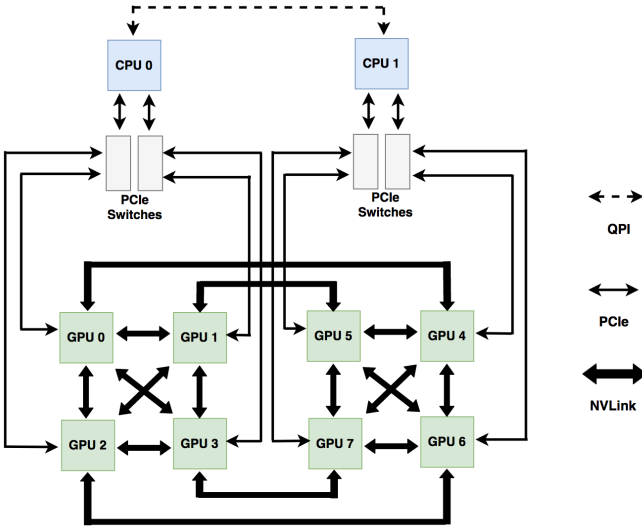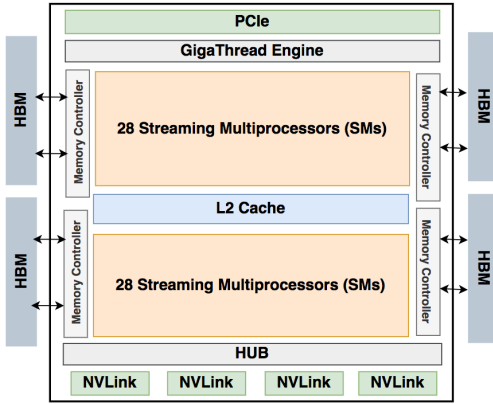
Fig. 1: Nvidia DGX-1 topology

Fig. 2: Pascal P100 GPU Architecture

links between the GPUs and CPUs enable access to the CPUs' bulk DRAM memory to enable working set and dataset streaming to and from the GPUs. The GPUs are connected in a hybrid cube-mesh network topology, with 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 others memories by using Nvidia provided CUDA APIs.

GPUs in DGX-1 box are Nvidia's Tesla P100 based on Pascal architecture which is shown in Fig. 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.

B. 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 CPU or on a single GPU. However, in our multi-GPU threat model, our attacks span multiple GPUs that are connected via NVLink-V1(as shown in Figure 1). The trojan or the victim process is located on a GPU (e.g. GPU 0) and the spy process is located in another GPU (e.g. GPU 1). Note that if both are located on the same GPU, then prior covert channel attacks on GPUs may be used [25]; however, we explain later how there are a number of advantages for conducting the attack remotely.

We assume an attacker with normal user access, capable of launching applications on one or more of the GPUs at the same time. The attacker does not have access to any specialized system support or supervisor 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, online 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.

The attacks are conducted on a DGX-1 box which consists of Pascal based Tesla P100 GPUs programmed using CUDA 10.0. We expect that with some fine tuning the attacks can be ported to other Multi-GPU systems. Although we focused on prime-probe attacks exploiting difference in timing between cache hits and misses, we expect that other sources of leakage such as performance counter values can also be used in these attacks [10], [25], [35].

III. REVERSE ENGINEERING CACHE ORGANIZATION

In multi-GPU system, a GPU can access the memory of a remote GPU that is connected via NVLink. Our attacks are cache based timing attacks. However, the cache hierarchy and its properties is not well documented. For this reason, we reverse engineer the cache hierarchy and its timing characteristics in this section.

A. Caching organization and timing properties

In the first set of experiments, our goal is to understand the overall cache hierarchy as well as the timing properties of different access types (hits vs. misses, local and remote). The DGX-1 offers a uniform address space, and virtual pages can be allocated to physical pages that belongs in any of the GPU HBM DRAM memory (i.e., a NUMA organization). The Pascal GPUs have two levels of data cache, L1 and L2. A programmer can bypass L1 data caching by using specific data loading primitives (specifically, *__ldcg()*). However, L2 data caching cannot be bypassed and all data and instructions get cached in L2.

In traditional cache-based timing attacks, an attacker needs to distinguish the cache hit and miss time for different cache levels in order to identify data/cache sets being accessed by the other process (either as part of a covert or side channel attack).

3

In the cross-GPU L2 based timing attack, the attacker needs to understand where data gets cached, and the hit and miss timing properties of both local and remote GPU's caches. Since our attack relies on creating contention between a remote GPU and a local GPU, we developed a microbenchmark to probe for these properties.

We allocate a buffer in the memory of one of the GPUs and use accesses to it to derive both local and remote access times. To find both the remote and local access time, we first populate the L2 cache with the data from a buffer in DRAM with the 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 and avoids L1 caching. 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 shared buffer to avoid any contention in the L2 cache as the access path of the shared buffer is separate than 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.
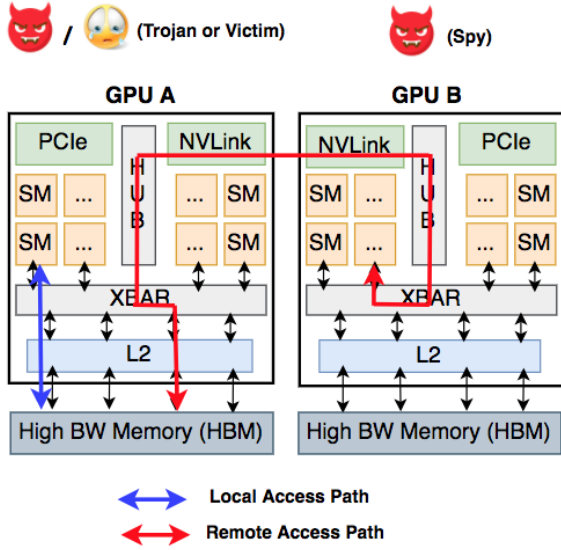

Fig. 3: Accessing a remote GPU's memory through NVLink

To measure *remote* L2 hit and miss time, we allocate a buffer on the remote GPU and we use *cudaDeviceEnablePeer-Access* to access the remote GPU's memory. Remote buffer allocation and accessing it does not create any context on the remote GPU, so our two processes have separate contexts that are created on two different GPUs (local and remote).

The local and remote GPU L2 and DRAM access time is shown in Fig 4 histogram. We have made 48 accesses in each loop to measure both local and remote GPU. 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 on the figure, there are four clusters of accesses with respect to the timing, varying from just over 250 cycles to over 850 cycles.

When examining the accesses we discover that the fastest accesses (green on the figure) occur to cached accesses to memory pages from the GPU where the memory is allocated. The next group of accesses correspond to local cache misses: DRAM accesses to the local HBM. The next two clusters correspond to cache hits on memory that is mapped to a remote GPU, and cache misses to this remote memory respectively. This experiment indicates that each L2 caches the data for the memory pages mapped to its own memory. It also provides us with timing thresholds to distinguish between cache hits and misses, to both local and remote GPU caches. We repeated the experiment by selecting different peer-to-peer GPUs connected via NVLink (single-hop) and we have observed similar timing cycle range. NVidia runtime API throws error if the GPUs are not connected via NVLink.
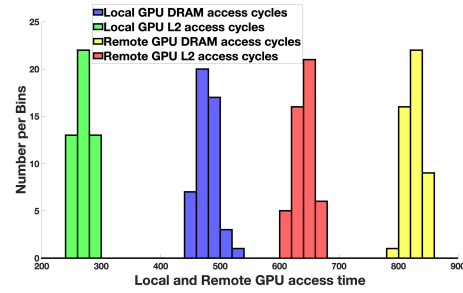

Fig. 4: Local and remote GPU access time

Thus, we understand the memory access pathways and caching to be as shown in Figure 3. 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 is cached on the remote GPU, rather 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 L2 cache of local GPU.** We use this shared remote L2 cache in GPU-to-GPU communication to build microarchitectural covert and side channel attacks.

B. 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. We also had to to work with the GPU computational model which required care to maintain synchronization. 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.* [23] explored different levels of memory hierarchy in GPUs. However, we could not use their attack directly because their reverse engineering process requires storing of all the timer values in shared memory which significantly limits the number of samples we are able to take. Jia *et al.* [13] 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.* [12] 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 chase experiment shown in Algorithm 1. Conceptually, this is 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.

The experiment proceeds as follows, a data buffer is allocated and the target index *targetIdx* is chosen in line 1 The target index is accessed in line 3 and the access time is recorded in a time buffer *sharedTimeBuff* allocated in shared memory in line 7. The target address access is followed by accessing other addresses in a loop from line 9. The number of addresses to be accessed is specified by *numOfElements*. The value of *numOfElements* starts with value of one in the first kernel launch.The access offset is set to 128 bytes which is the cache line size. The number of accesses are increased over subsequent kernel calls which signifies the number of addresses traversed. The target address are accessed again at the end of *for loop*. The second access time of the target address is recorded in another location of the shared buffer.

After the end of each kernel call these two access times are checked on the host side. The first access time of the target address is the DRAM access time and if the target address resides in the L2 cache then the second access time would be equivalent to the cache access time. However, if the access causes target address to be evicted, then it would be equivalent to the DRAM access time. The target address eviction in this case is caused by the accessing the last address that got accessed.This eviction of the target address indicates that the target address and the last address are in the same

cache set. Next, to find the rest of the addresses within the same cache set, we remove that last address that caused the eviction from the pointer chase in subsequent kernel calls and continue with the process to find more addresses that hash to the same set. Also, to find more eviction sets the attacker needs to change the target address and repeats the pointer chase process. To reduce the search space we adopted some optimization methodologies by skipping some address accesses. However, if an eviction is seen in the target address then we revert back and check all those last skipped addresses and determine which exact address causes the eviction of the target address. This processes can be optimized by observing the data belonging to a page is indexed consecutively in the cache.

Algorithm 1: Eviction Set Determination Algorithm

1 basePtr = &mainBuffer[*targetIdx*];
2 start = clock();
3 nxtIdx =__ldcg(basePtr);
4 dummy+=nxtIdx;
5 end = clock();
6 __(*threadfence()*);
7 sharedTimeBuff[0] = (end-start);
8 *dummy Operation*
9 **for** $i = 0$; $i < numOfElements$; $i = i + 1$ **do**
10   otherPtr = &mainBuff[nxtIdx];
11   nxtIdx = __ldcg(otherPtr);
12   dummy+=nxtIdx;
13   __*threadfence()*;
14 **end**
15 *dummy Operation*
16 start = clock();
17 nxtIdx =__ldcg(basePtr);
18 dummy+=nxtIdx;
19 end = clock();
20 __*threadfence()*;
21 sharedTimeBuff[1] = (end-start);
22 *dummy Operation*

We also observed that the derived eviction sets remain valid over application runs as long as the memory allocation size of the process remains unchanged. We also confirmed that the address placement in the cache is independent of the GPU which the kernel is launched on. These observations allow us to simplify the attack to avoid deriving the conflict sets online in every attack.

The cache line size is 128B and from our eviction set determination experiment, we also learn the associativity of the cache (16). We repeat the eviction set algorithm 1 with those recorded addresses only. 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. Also, the eviction pattern shows that the replacement policy is LRU (or pseudo-LRU) without randomization since the target address are evicted consistently after 16th address. Table I summarizes

TABLE I: 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

L2 cache parameters and architecture derived from our reverse engineering experiments.
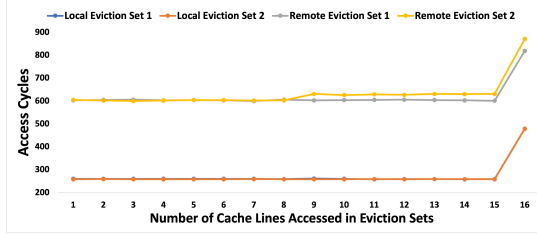

Fig. 5: Validating the eviction set determination

Figure 5 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.
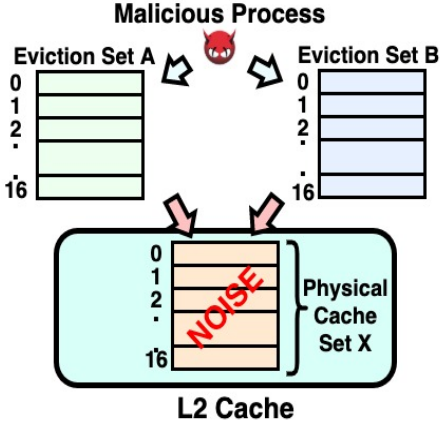

Fig. 6: Eviction set aliasing issue

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 (Figure 6). Specifically, there are two eviction sets A and B determined by the malicious process that happen to index to the same physical cache set X, 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 inferring a victim 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 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.

IV. 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 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 [24]). The attack is conducted from user level and do not require any system level features such huge pages or flush instructions that are necessary for many attacks.

In this attack, the Trojan (the transmitter of the covert channel) is located on a local GPU, GPU A, and Spy (the receiver) is located on the remote GPU, GPU B, and accesses the memory of the GPU1 to synchronize and receive information (the opposite is also possible). These two processes communicate covertly over the shared L2 cache of GPU A. First, the spy primes a cache set. To communicate "1", the trojan would access it's own data, evicts the spy's data, and fills up the cache set and to communicate "0", the trojan process does nothing. The spy 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 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 overcome them next.

A. Aligning the cache sets

At the stage, the two processes have derived each eviction sets covering the L2 cache. However, all they 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
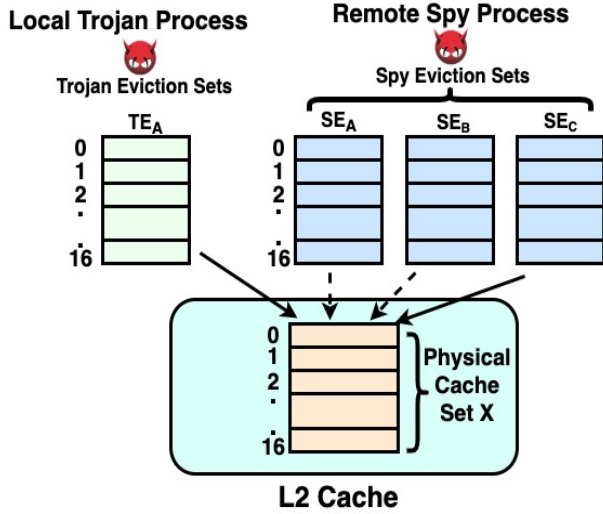
Fig. 7: Eviction set alignment among multiple processes

Assume again that the trojan process is located on GPU A and the spy process has been launched on GPU B and they are connected via NVLink. Both processes allocated their buffer on GPU A and share the L2 cache on that GPU. In this scenario, the trojan is a local process and the spy is remote. In a single run of the malicious applications, one trojan eviction set is checked with another eviction set of the spy process. In Fig 7, we can see a local trojan process eviction set TE_A launched on GPU A and the remote spy process launched on GPU B have three eviction sets SE_A, SE_B and SE_C. The eviction set of the local trojan process is checked against three eviction sets of the spy process that could be located in the same physical cache set X. The set matching experiment reveals that the trojan eviction set TE_A is not mapped to the spy eviction set SE_A and SE_B shown by dotted arrows. But the trojan eviction set TE_A is mapped to the spy eviction set SE_C.

The eviction set mapping kernel is shown in Algorithm 2. Eviction set is accessed from line 5 - 13 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 14 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 17. The kernel algorithm is same for both the trojan and spy processes. The only difference between them is the number of outer loops that decides how many times a cache set would be probed. The trojan process has a faster access compared to the spy process as the memory is local to the trojan process. So the value of *numMainLoop* is much higher for the trojan process compared to the spy process. For our work, we have selected a value of 400000 and 150000 for the local trojan and remote spy 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 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.

Algorithm 2: Eviction set alignment across processes

```
1  for i = 0; i < numMainLoop; i = i + 1 do
2      idxTemp = startIdx;
3      timer1 = 0;
4      dummy1 = 0;
5      for i = 0; i < numOfCacheLines; i = i + 1 do
6          dataPtr = &mainBuff[idxTemp];
7          start = clock();
8          idxTemp = __ldcg(dataPtr);
9          dummy1+=idxTemp;
10         end = clock();
11         timer1+=(end-start);
12         __threadfence();
13     end
14     timer2+=(timer1/numOfCacheLines);
15     dummy operation
16 end
17 timeBuffMain = (timer2/(numMainLoop));
```

B. Putting it together: Covert channel attack

The trojan process (launched on GPU A) allocates the data buffer on the same GPU in step 1 and the spy process gets launched on another GPU (B), but allocates the buffer on the remote GPU A, where the trojan process is launched. The first access of both the trojan and the spy 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 8. The trojan (or sender) is located on GPU A and the spy (receiver) is located on GPU B. Step 1 and 2 of the attack represent the determination the eviction sets of both processes. These are followed by the alignment step (Step 3 on 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.

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 a SM in the GPUs. Hence, when the communication takes place over a single cache set, a single thread block on both trojan and spy side would access their own eviction set whose mapping was determined from
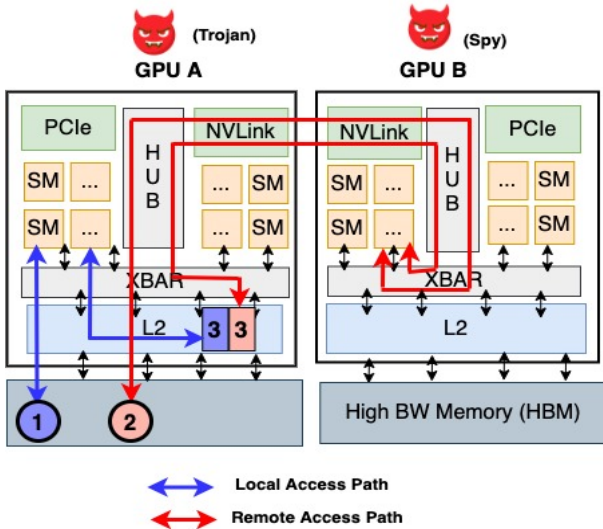
Fig. 8: Cross GPU covert channel attack

the set aligning step. We leveraged the GPU parallelism by increasing the number of thread blocks. Each thread block, in both trojan and spy, would access different eviction sets that are already mapped to have a faster communication. The trojan thread block consists of a single warp of threads (32 on our machines). All 32 threads in a thread block of the trojan 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 spy 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 to 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 increasing the parallelism during the data copy. To send a "1" the trojan process accesses the cache set, replacing the data placed there by the spy, 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 trojan process.

C. Covert Channel Evaluation

In this section, we evaluate the multi-GPU covert channel attack. All experiment 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 side 1Mb across the covert channel. We vary the number of cache sets we use in the attack. Fig. 10 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 have been transferred covertly.

The y-axis shows the timed access cycles measured from the remote spy 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 trojan and the spy processes are located on different GPUs, we tune parameters on the trojan side that controls the cache access frequency to communicate the covert message successfully to the spy side.
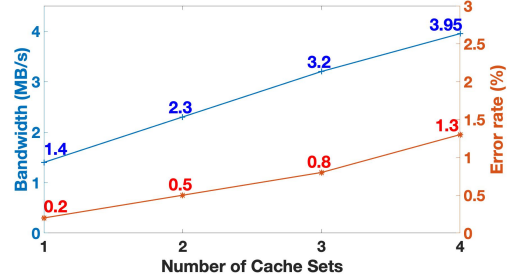

Fig. 9: Bandwidth and Error rate in covert channel

The bandwidth and the error rate are shown in Fig. 9. 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 trojan to spy. 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 in parallel and with an average error rate of 1.3% measured over 1000 runs. Adding additional sets improves bandwidth but also results in a higher error rate.**

V. SIDE CHANNEL ATTACK

We also demonstrate proof of concept side channel attacks. The attack primarily uses a spy probe a remote cache and recover a *memorygram* of the accesses to the cache, which is a collection of cache hits and misses for the different cache sets over time. Previously, memorygram have been used in cache side channel attacks for website fingerprinting [32] on CPUs. This access pattern correlates with the activity on the remote GPU, allowing us to infer information about the applications running on this GPU. The target of our side channel attack is application fingerprinting on a target GPU in a DGX-1 box. Our side channel attack model is demonstrated in Fig. 3. Specifically, the attacker is located on GPU B, and accesses the eviction sets it pre-constructed in its own buffer that is allocated on a remote GPU A. By having the eviction sets for each L2 cache set and measuring the access time on each set, the spy can remotely infer whether the local application has accessed the set (replacing its own data) or not. The memory
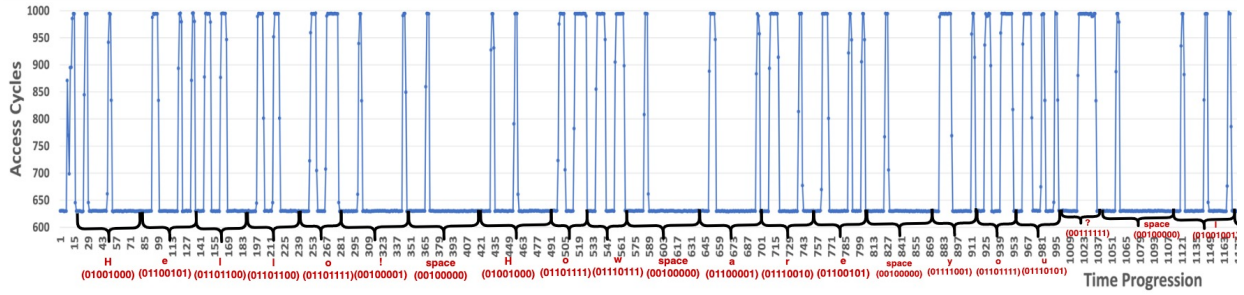
Fig. 10: Cross GPU covert message received by spy process

footprint of applications get recorded over a period of time which reveal the access pattern of the applications on different L2 cache sets. We have demonstrated side channel attack on different genre of applications. Our first side channel attack is on high performance applications and our second attack is on deep learning applications.

A. Application Fingerprinting using a side channel attack

We demonstrate a side channel where we finger print the remote HPC application based on the memorygram. Specifically, we pre-train a deep learning network to identify applications based on their memorygram. This attack can serve as a first step of future attacks where we identify a target application, and then infer information about it. This attack can be used to identify and reverse engineer the scheduling of applications on a multi-GPU system (simply by spying on all other GPUs in a GPU-box), and identify a target GPUs that are running a specific victim application, and even identify the kernels running on each GPU.

In our proof of concept attack, we used six different applications from NVidia toolkit [28] as our victim applications. Our application set include common HPC workloads like vectoradd, histogram, blackscholes, matrix multiplication, quasirandom and welshtransform.

Example memorygrams of victim applications are shown in Fig. 11; note that these can be different in each run because the conflict sets hash to arbitrary sets within the cache. There is some structure, because the hashing preserves page boundaries; that is, the addresses within a single page will hash to consecutive sets in the physical cache. The X-axis of each image is the execution timeline of the spy application and the Y-axis is the cache set number. The yellow dots represent a cache miss on the L2 each, indicating a likely victim application's access. The image shows the cache misses that occurred on 256 sets of L2 cache. Each victim application leaves a unique memory footprint.

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 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,

TABLE II: Average misses over all cache sets

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

keeping a sufficiently large test set ensures that we evaluate the generalization capabilities with good confidence.

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 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 12. 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.

B. Side channel attack on 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. Due to time limitations available for revisions, we demonstrate only the principles of the attack and evidence that it can be successful.

Our victim attack is a Multi-layer Perceptron (MLP) model with 1 hidden layer built using pytorch [30]. The application trains the MLP using the MNIST digit recognition data set [5]. We have use 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. 13. Visually, we can see that the intensity of misses increases as the size of the hidden layer increases, reflecting the additional computations during training.
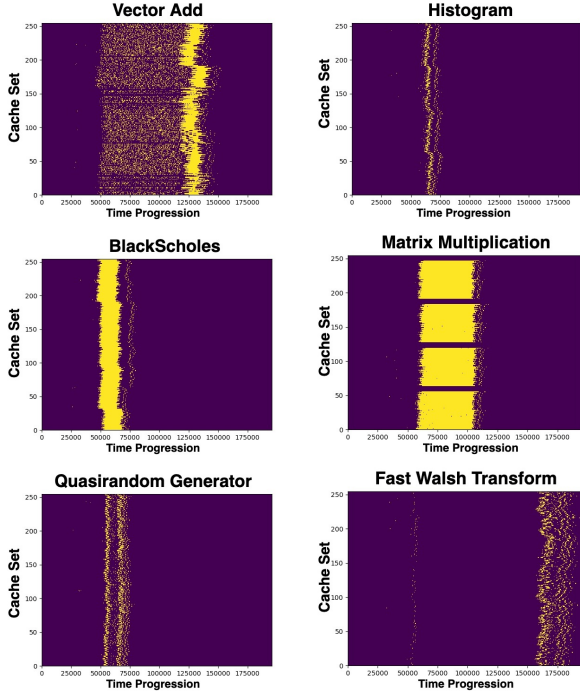
9

Fig. 11: Memorygram of 6 applications

Fig. 12: Confusion Matrix. BS (Black Scholes), HG (Histogram), MM (Matrix Multiplication), QR (Quasi Random), VA (Vector Addition), WT (Walsh Transform)

Table II shows the average number of cache set misses; we see separation which allows us to infer the configuration. Fig. 14a and Fig. 14b show the memorygram of the application with 128 and 512 number of neurons. The memorygram data is richer, showing the pattern of misses over time, and we believe we can use a classifier on this data to infer more detailed information about the model. For example, the model was configured to run two epochs in Fig. 15 (two full passes through the training set). The number of epochs is a hyperparameter which we are able to infer visually in Fig. 15.

VI. 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 L2 cache and as a result, adding noise to the covert or side channel attacks.
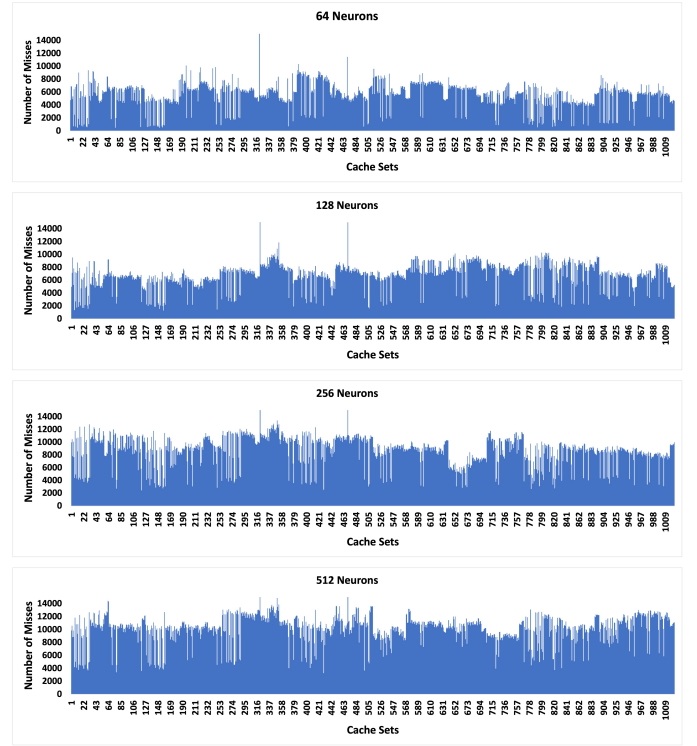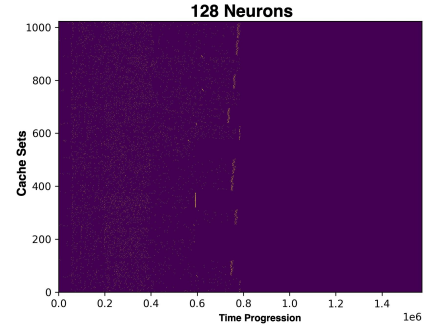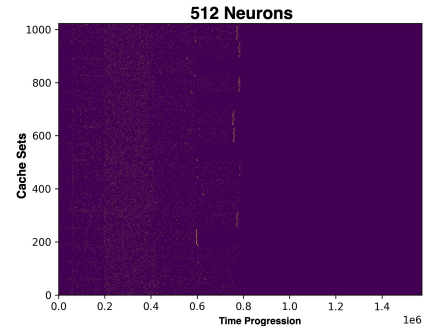

Fig. 13: Cache misses per set

(a) Memorygram of MLP with 128 neurons

(b) Memorygram of MLP with 512 neurons

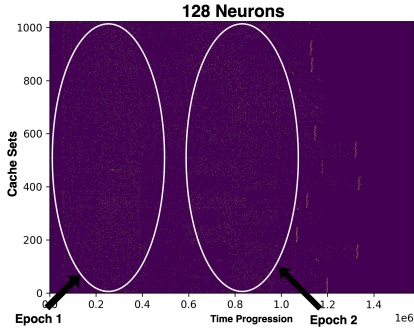Fig. 14: Memorygram of the MLP application

Fig. 15: Memorygram for a two-epoch experiment

For mitigating noise, we propose to leverage concurrency limitations of GPUs using similar approaches as prior work [24] to force exclusive execution of spy or trojan on GPUs. Based on 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 maximum number of thread block per SM. For example, in covert channel attacks, if we control the resource demand of our trojan on GPU A and spy 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 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 spy (or trojan) on GPU reducing noise.

VII. POSSIBLE MITIGATIONS

Defenses against microarchitectural covert and side channel attacks on CPUs and GPUs can potentially apply to cross-GPU attacks with some adaptations. One solution is static or dynamic partitioning of shared resources [6], [17], [18], [34], [36]. For example, Nvidia designed Multi-Instance GPU (MIG) Technology [29] in their new generations of discrete GPUs (Ampere). In this design, a single GPU can be securely partitioned into separate GPU instances for multiple users with the isolated paths through the entire memory system; the on-chip crossbar ports, L2 cache banks, memory controllers, and DRAM address busses are all assigned uniquely to an individual instance. However, MIG feature requires privileged access and is not available in Pascal and Volta based DGX machines, still leaving these boxes vulnerable to microarchitectural attacks.

To minimize the performance overhead of these partitioning-based defense mechanisms, they can only be triggered when contention is detected on a shared resource (similar to the proposed framework in [36]). In multi-GPU systems, the detection of cross-GPU covert or side channel attacks is possible by monitoring the traffic over NVLinks and access patterns on L2 and memory (accessible through hardware performance counters). In addition, some prior works [16] propose to place the data along with the thread block that accesses it in the same GPU to minimize the remote traffic in multi-GPU systems, and as a result to improve the performance. Although inherent GPU-to-GPU communications can not be completely eliminated in multi-GPU systems, making these cross-GPU data transfers more coarse-grained in normal applications will significantly increase the detection accuracy of high-bandwidth attacks, leading to more efficient defenses.

VIII. 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. [24] 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. [26] develop a similar microarchitectural covert channel on GPU's shared last level translation lookaside buffer(TLB) and Ahn et al. [3] implement covert channel attacks on shared on-chip interconnect on GPUs. In a completely different environment, Dutta et al. [7] developed covert channel attacks between CPU and GPU through shared LLC and ring bus in integrated CPU-GPU systems. [9] Conducted a microarchitectural attack on the shared memory in the intel based integrated CPU-GPU systems.

GPU side channel attacks can be categorized in two different threat models: (1) the spy launches the GPU kernel and measures the leakage from the CPU (host) side by exploiting memory coalescing [2], [14] or shared memory bank conflicts [15] and their correlation with the execution time of GPU kernel, or by collecting hardware performance counters [33], Electromagnetic traces [8], [10] and power consumption traces [21]. Most of these attacks have been implemented to extract the encryption key. (2) the spy is co-located on the GPU with the victim process and measures the contention on the shared resources through hardware performance counters. Through these side channel attacks, the attacker can implement website fingerprinting, inter-keystroke timing attack [25], workload fingerprinting [20], [38], or Neural Network model extraction attacks [25], [35].

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 [29], [36].

IX. 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 covert channel and side channel prime-and-probe based attacks across different GPUs. This attack expands our understanding of the threat model faced by these systems. For example, we show that defenses designed to protect GPUs against covert and side channel attacks are not set up to prevent these new attacks, which motivates new defenses that can mitigate them.

References

[1] "Nvidia achieves breakthroughs in language understanding to enable real-time conversational ai," 2019, accessed November, 2021 from https://nvidianews.nvidia.com/news/nvidia-achieves-breakthroughs-in-language-understandingto-enable-real-time-conversational-ai.

[2] J. Ahn, C. Jin, J. Kim, M. Rhu, Y. Fei, D. Kaeli, and J. Kim, "Trident: A hybrid correlation-collision gpu cache timing attack for aes key recovery," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 332–344.

[3] J. Ahn, J. Kim, H. Kasan, L. Delshadtehrani, W. Song, A. Joshi, and J. Kim, "Network-on-chip microarchitecture-based covert channel in gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 565–577. [Online]. Available: https://doi.org/10.1145/3466752.3480093

[4] AMD, "Amd crossfire guide for direct3d® 11 applications," 2017.

[5] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[6] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 1–21, 2012.

[7] S. B. Dutta, H. Naghibijouybari, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 972–984.

[8] Y. Gao, H. Zhang, W. Cheng, Y. Zhou, and Y. Cao, "Electro-magnetic analysis of gpu-based aes implementation," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196042

[9] W. HE, W. Zhang, S. Sinha, and S. Das, "Igpu leak: An information leakage vulnerability on intel integrated gpu," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE Press, 2020, p. 56–61. [Online]. Available: https://doi.org/10.1109/ASP-DAC47756.2020.9045745

[10] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 385–399. [Online]. Available: https://doi.org/10.1145/3373376.3378460

[11] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 591–604.

[12] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional gpus: Software-based compute and memory bandwidth reservation for gpus," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 29–41.

[13] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[14] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *IEEE International Symposium on High Performance Computer Architecture*, ser. HPCA'16. Barcelona Spain: IEEE, March 2016, pp. 394–405. [Online]. Available: http://ieeexplore.ieee.org/document/7446081

[15] Z. H. Jiang, Y. Fei, and D. Kaeli, "A novel side-channel timing attack on gpus," in *Proceedings of the on Great Lakes Symposium on VLSI*, ser. VLSI'17, 2017, pp. 167–172.

[16] H. Kim, R. Hadidi, L. Nai, H. Kim, N. Jayasena, Y. Eckert, O. Kayiran, and G. Loh, "Coda: Enabling co-location of computation and data for multiple gpu systems," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, sep 2018. [Online]. Available: https://doi.org/10.1145/3232521

[17] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou, "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)*, February 2009.

[18] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud

computing," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

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

[20] S. Liu, Y. Wei, J. Chi, F. H. Shezan, and Y. Tian, "Side channel attacks in computation offloading systems with gpu virtualization," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 156–161.

[21] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli, "Side-channel power analysis of a gpu aes implementation," in *33rd IEEE International Conference on Computer Design*, ser. ICCD'15, 2015.

[22] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: cross-cores cache covert channel," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 46–64.

[23] X. Mei and X. Chu, "Dissecting gpu memory hierarchy through microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, 2016.

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

[25] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2139–2153. [Online]. Available: https://doi.org/10.1145/3243734.3243831

[26] A. Nayak, P. B., V. Ganapathy, and A. Basu, "(mis)managed: A novel tlb-based covert channel on gpus," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 872–885. [Online]. Available: https://doi.org/10.1145/3433210.3453077

[27] Nvidia, "Nvidia dgx-1 system architecture white paper," 2017.

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

[38] P. Zou, A. Li, K. Barker, and R. Ge, "Fingerprinting anomalous computation with rnn for gpu-accelerated hpc machines," in *2019 IEEE*

[29] Nvidia, "Nvidia multi-instance gpu," 2021, https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[31] G. Saileshwar, C. W. Fletcher, and M. Qureshi, "Streamline: a fast, flush-less 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*, 2021, pp. 1077–1090.

[32] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 639–656.

[33] X. Wang and W. Zhang, "An efficient profiling-based side-channel attack on graphics processing units," in *National Cyber Summit (NCS) Research Track*, K.-K. R. Choo, T. H. Morris, and G. L. Peterson, Eds. Cham: Springer International Publishing, 2020, pp. 126–139.

[34] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2007.

[35] J. Wei, Y. Zhangy, Z. Zhou, Z. Liy, and M. Abdullah Al Faruque, "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)*, 2020.

[36] Q. Xu, H. Naghibijouybari, S. Wang, N. Abu-Ghazaleh, and M. Annavaram, "Gpuguard: Mitigating contention based side and covert channel attacks on gpus," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '19. New York, NY, USA: ACM, 2019, pp. 497–509. [Online]. Available: http://doi.acm.org/10.1145/3330345.3330389

[37] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.
International Symposium on Workload Characterization (IISWC), 2019, pp. 253–256.

13