Leaky Buddies: Cross-Component Covert Channels on Integrated CPU-GPU Systems

Sankha Baran Dutta
Department of CSE
University of California, Riverside
Riverside, California, USA
sdutt004@ucr.edu

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

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*—Graphics Processing Units (GPUs) are ubiquitous components used across the range of today's computing platforms, from phones and tablets, through personal computers, to high-end server class platforms. With the increasing importance of graphics and video workloads, recent processors are shipped with GPU devices that are integrated on the same chip. Integrated GPUs share some resources with the CPU and as a result, there is a potential for microarchitectural attacks from the GPU to the CPU or vice versa. We consider the potential for covert channel attacks that arise either from shared microarchitectural components (such as caches) or through shared contention domains (e.g., shared buses). We illustrate these two types of channels by developing two reliable covert channel attacks. The first covert channel uses the shared LLC cache in Intel's integrated GPU architectures. The second is a contention based channel targeting the ring bus connecting the CPU and GPU to the LLC. This is the first demonstrated microarchitectural attack crossing the component boundary (GPU to CPU or vice versa). Cross-component channels introduce a number of new challenges that we had to overcome since they occur across heterogeneous components that use different computation models and are interconnected using asymmetric memory hierarchies. We also exploit GPU parallelism to increase the bandwidth of the communication, even without relying on a common clock. The LLC based channel achieves a bandwidth of 120 kbps with a low error rate of 2%, while the contention based channel delivers up to 400 kbps with a 0.8% error rate. We also demonstrate a proof-of-concept prime-and-probe side channel attack that probes the full LLC from the GPU.**

I. Introduction

In recent years, micro-architectural covert and side channel attacks have been widely studied on modern CPUs, exploiting optimization techniques and structures to exfiltrate sensitive information. A preponderance of these studies exploit CPU structures examining channels through a variety of contention domains including caches [27], [32], [41], [45], [49], branch predictors [9], random number generators [8], and others [5], [28]. Modern computing systems are increasingly heterogeneous, consisting of a federation of the CPU with GPUs, NPUs, other specialized accelerators, as well as memory and storage components, using a rich interconnect. It is essential to understand how micro-architectural attacks manifest within such complex environments (i.e., beyond just the CPU).

In this paper, we demonstrate for the first time covert channel attacks between a CPU and integrated GPU. Although covert channels have been demonstrated on a variety of CPU structures, as well as on discrete GPUs [23], [24], [36], [37], we believe our attacks are significantly different from prior work because they operate across heterogeneous components. Specifically, to the best of our knowledge, all prior demonstrated covert channels are symmetric, with both the sender and receiver being identical: typically threads or processes access a resource that they use to create contention. In contrast, cross-component attacks occur between two entities that can have substantially different computational models, and that share have asymmetric views of the resources. As a result, not only does the attacks necessitate careful reverse engineering of asymmetric views of the resource from both side, it also requires solutions to new problems that arise due to the asymmetry in sharing the resources. Moreover, we believe this is the first attack demonstrated on heterogeneous environments, providing important insights into how this threat model manifests in such systems, and extend our understanding of the threat model and guide further research into defenses.

An iGPU is integrated on the die with the CPU and shares resources such as the last level cache and memory subsystem with it. This integration creates the potential of new attacks that exploit common resources to create interference between these components, leading to cross-component micro-architectural attacks. Specifically, we develop covert channels (secret communication channels that exploit contention) on integrated heterogeneous systems in which two malicious applications, located on two different components (CPU and iGPU) transfer secret information via shared hardware resources. Section II provides an overview of the integrated CPU-GPU systems architecture and our threat model.

We demonstrate for the first time the vulnerability of these types of widely-used systems to microarchitectural attacks.

978-1-6654-3333-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ISCA52012.2021.00080

These cross-component attacks require solving new problems due to the asymmetric nature of the two communication ends (one on the CPU, and the other on the GPU). These problems include the different views of the memory hierarchy, the need for synchronization across heterogeneous components with frequency disparity, reconciling different computational models and memory hierarchies, and creating reliable fine-grained timing mechanisms. We also demonstrate the possibility of the more dangerous side-channel attacks. These new attacks provide concrete examples for the first time of cross-component attacks in heterogeneous systems, expanding our understanding of microarchitectural attacks and guiding mitigation strategies.

We consider two possibilities for creating covert channels in cross-component systems: (1) Contention through directly shared microarchitecture resources. In the case of the integrated CPU-GPU system we use in our experiments, the lowest level of the cache (the LLC) is shared and serves as our example of this type of channel; and (2) Contention through time multiplexed resources such as shared buses, cache ports, computational units and similar resources. In such resources, if both components use the resource at the same time, there is a perceived delay as their requests contend for the use of the limited resource. We illustrate this type of channel by building a covert channel attack on the ring-bus interconnect the CPU and GPU to the LLC cache. We construct two covert channel attacks: (1) A Prime+Probe covert channel attack using the shared LLC (Section III); and (2) a contention based covert channel using contention on the shared bus to reach the LLC (Section IV). For each attack, we had to solve a number of unique challenges that arise due to the asymmetric nature of the channels (e.g., the view of the memory hierarchy is different from the two sides, with critical implications on how we can build a successful attack). The LLC based channel achieves a bandwidth of 120 kbps with an error rate of 2% while the ring-bus based channel achieves a bandwidth of 400 kbps with a 0.8% error rate. We also demonstrate a prime and probe side channel attack where the GPU is able to spy on LLC accesses generated by the CPU. We discuss the potential mitigations in Section VII and compare our work to other related attacks in Section VIII.

In summary, the contributions of this paper are:

- We present a new class of attacks that span different components within a heterogeneous system.
- We show a number of new challenges that arise in cross-component attacks due to the asymmetry between the two communication ends. These include matching the attack cycle from two different computational models with different clock cycles, and reverse engineering and understanding asymmetric views of the memory hierarchy, as well as others. We believe that some of these issues generalize beyond our specific environment.
- We build and characterize two real covert channel attacks on an integrated CPU-GPU system.
- We build a proof of concept prime-probe side-channel attack with GPU spying on the cache activity of the CPU.

Section IX discusses possible future research and presents our concluding remarks.

II. Background and Threat Model

In this section, we introduce the organization of Intel's integrated GPU systems, to provide the background necessary to understand our attack. We also present the threat model, outlining our assumptions on the attacker's capabilities.

A. Intel Integrated CPU-GPU systems

Traditionally, discrete GPUs are connected with the rest of the system through PCIe bus, and have access to a separate physical memory (and therefore memory hierarchy) than that of the CPU. However, starting with Intel's Westemere in 2010, Intel's CPUs have integrated GPUs (*iGPU*) incorporated on the same die with the CPU, to support increasingly multimedia heavy workloads without the need for a separate (bulky, expensive, and power hungry) GPU. This GPU support has continued to evolve with every generation providing more performance and features; for example the Iris Plus on Gen11 Intel Graphics Technology [17] offers up to 64 execution units (similar to CUDA cores in Nvidia terminology) and at the highest end, over 1 Teraflop of GPU performance.

For general purpose computing on integrated GPUs, the programmer uses OpenCL [1] (equivalent to CUDA programming model on Nvidia discrete GPUs [38]). Based on the application, programmers launch the required number of threads that are grouped together into work-groups (similar to thread blocks in Nvidia terminology). Work-groups are divided into groups of threads executing Single Instruction Multiple Data (*SIMD*) style in lock step manner (called wavefronts, analogous to warps in Nvidia terminology). In integrated GPUs the SIMD width is variable, changing depending on the register requirements of the kernel.

iGPUs reside on the same chip and connect to the same memory hierarchy as the CPU (typically at the LLC level). Figure 1 shows the architecture of an Intel SoC processor, integrating four CPU cores and an iGPU [15]. The iGPU is connected with CPUs and the rest of the system through a ring interconnect: a 32 byte wide bidirectional data bus. The GPU shares the Last Level Cache (*LLC*) with the CPU, which serves as the last level of the GPUs cache hierarchy. The GPU

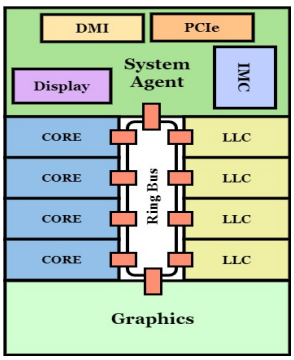

Fig. 1: Intel SoC architecture

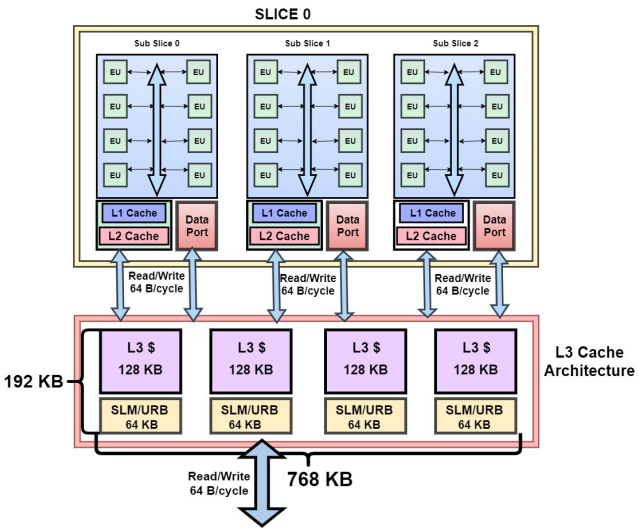

Fig. 2: Intel integrated GPU architecture

and CPU can access the LLC simultaneously. However, there is an impact on the access latency due to factors such as delays in accessing the bus and access limitations on the LLC ports. We characterize the contention behavior in Section IV. The GPU and CPU share other components such as the display controller, the PCIe controller, the optional eDRAM controller, and the memory controller.

The architecture of the iGPU is shown in Figure 2. A group of 8 EUs (analogous to CUDA cores) is consolidated into a single unit which is called a *Subslice* (similar to SM in Nvidia terminology) and typically 3 subslices create a *Slice*. The number of slices varies with the particular SoC model even within the same generation, as the slices are designed in a modular fashion allowing different GPU configurations to be created. Experimentally, we discovered that multiple work-groups are allocated to different subslices in a round-robin manner. The global thread dispatcher launches the work-groups to different subslices. A single SIMD width equivalent number of threads in a single subslice is launched to EUs in a round-robin manner as well. A fixed functional pipeline (not shown in the figure) is dedicated for graphics processing.

The iGPU uses three levels of cache (in addition to the shared LLC). The first two levels, L1 and L2, are called sampler caches and are used solely for graphics. The third level cache, L3, is universal and can be used for both graphics and computational applications. We explain the organization of the L3 cache in more detail in Section III-D. In each slice, there is also a shared local memory (SLM), a structure within the L3 complex that supports programmer managed data sharing among threads within the same work-group [15].

B. Threat Model

In this paper, we build two covert channel attacks originating from an integrated GPU to the CPU or vice versa. We also demonstrate a proof-of-concept side channel attack allowing the GPU to spy on the CPU. In a covert channel, two processes (a *Trojan* sending data to a *Spy*) communicate covertly using an indirect channel. Previously established covert channels were between similar processes and within the same physical device, either a CPU [35] or GPU [36], but not spanning both. In contrast, our covert channel differs in that the trojan and the spy processes communicate across different heterogeneous components, each featuring a different execution model, memory hierarchy and clock domains. Specifically, the trojan process launches a kernel on the GPU and the spy process operates completely on the CPU during communication. We also demonstrate the communication in the other direction (in fact, we implement bidirectional covert channels). We explore two different covert channels, one using a Prime+Probe style attack on the LLC, and another that uses contention as the two processes concurrently access a shared resource to implement the communication.

We assume that the trojan and spy processes are both separate user-level processes without additional privileges, one running on the GPU and another on CPU. There is no explicit sharing between them (for example sharing of memory objects). The communication on the LLC occurs over pre-agreed sets in the cache. Such agreement is not required in a contention based attack and can be relaxed by dynamically identifying sets to communicate (but we do not pursue such an implementation). On the GPU side of our attacks, the program uses the GPU through user-level OpenCL API calls (we suspect that channels could also be established using OpenGL or other graphics calls). All of our experiments are on a Kaby Lake i7-7700k processor, which features an integrated Intel's Gen9 HD Graphics Neo. We use OpenCL version 2.0 (Driver version 18.51.12049), running Ubuntu version 16.04 LTS (which uses Linux Kernel version 4.13). The attacks were developed and tested on an unmodified but generally quiet system (not running additional workloads) on the GPU side of the attack. Current iGPUs are not capable of running multiple computation kernels from separate contexts concurrently and therefore no noise is expected on the GPU side.

III. LLC-based Covert Channel

This section presents the first covert channel attack: a Prime+Probe channel using the shared LLC cache. Prime+Probe is one of the most common strategies of cache-based attacks [41]; it is also one of the most general strategies because it does not require sharing of parts of the address space as required by other strategies, for example, those requiring sharing to be able to flush data out of the caches. In Prime+Probe, first the spy process accesses its own data and fills up the cache (priming). Next, the trojan either accesses its own data (replacing the Spy's) to communicate a "1", or does nothing to communicate a "0". Finally, the spy can detect this transferred bit by re-accessing its data (probe) and measuring the access time. If the time is high, indicating a cache miss, it detects a "1", otherwise a "0".

A. Attack Overview and Challenges

In this attack, the CPU and GPU communicate over the LLC cache sets. Figure 3 depicts the overview of the attack. We illustrate the attack at a high level using a trojan process launched on the GPU, communicating the bits to the CPU but the opposite is also possible. The Spy process which is receiving the bits is launched on the CPU. Communication from GPU to CPU is a 3 step process. The first two steps are for handshaking before the communication to make sure that the two sides are synchronized, which is especially important for heterogeneous components that can have highly disparate communication rates. The GPU initiates the handshake by priming the pre-agreed cache set and letting the CPU know that it is ready to send. Once the CPU receives the signal by probing the same cache set, the CPU acknowledges it back by priming a different cache set and sending ready to receive signal back to GPU in the second phase. GPU receives ready to receive signal by probing the same cache set that was primed by CPU. This ends the handshaking phase and the attack moves to the third step when GPU sends the data bit to CPU. For sending 1, GPU primes the LLC cache set that is probed by CPU. If GPU wants to send 0, it doesn't prime the cache set but CPU still probes. This 3 phase communication repeats communicating the secret bits covertly from the GPU process to the CPU process.

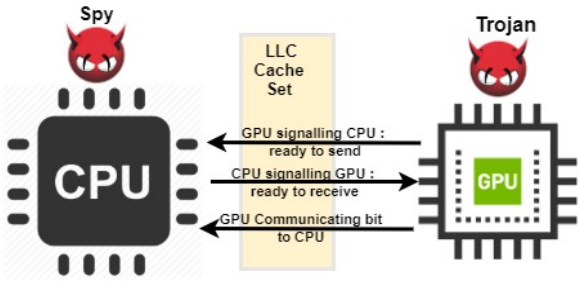

Fig. 3: LLC based CPU-GPU covert channel overview

Although at a high level this attack strategy is similar to other covert channel attacks, there are a number of unique challenges that occur when we try to implement the channel between the CPU and GPU. The challenges generally arise from the heterogeneous nature of the computational models on the two components, as well as the different memory hierarchies they have before the shared LLC. We overview these challenges and our approach to them briefly next.

- **Lack of a GPU timer:** Prime+Probe attacks rely on the ability to time the difference between a cache hit and a cache miss to implement communication. Usually, a user-level hardware counter is available on the system to measure the access latency. While this is true on the CPU side, unfortunately, OpenCL on iGPUs does not provide any such means to the programmer. We describe this problem and the custom user-level timer we develop to overcome it in Section III-B.
- **Using SVM to reverse engineering the shared LLC from the GPU:** Modern GPUs come with their own page tables and paging mechanisms. When a CPU process initializes and launches the GPU kernel, the CPU page table is shared with the GPU in this scenario. This sharing allows us to reverse engineer the cache from the CPU using established techniques [51] and use these results on the GPU.
- **Asymmetric view of LLC from GPU cache hierarchy:** We discover that the view of the LLC from the GPU is substantially different from the CPU–this is a type of problem that arises due to the asymmetric nature of the channel. Within the iGPU, there are three more levels of cache. We discover that the GPU caches are not inclusive relative to the LLC (unlike the CPU caches), which requires us to understand the GPU L3 (L1 and L2 are disabled by OpenCL) in detail in order to control evictions from it rather than relying on inclusivity to cause evictions. Another unique problem arising in this environment is that the indexing of the GPU L3 is dissimilar to that of the LLC: the eviction set for the GPU L3 can map to different LLC sets and cause significant noise, which is atypical in modern cache hierarchies. We needed to find controlled eviction sets (which we call *pollution sets*) at L3 level such that the targeted LLC sets are evicted from L3 by the pollution sets without causing spurious accesses to the sets we are using in the LLC. We describe this challenge in Section III-D.
- **Matching the communication rate across heterogeneous components:** Since the spy and the trojan use completely different computation models operating at substantially different clock rates, determining how to best implement the channel to improve bandwidth and reduce noise is also a new problem introduced by asymmetry. We address this problem by a combination of trying to match the access rate by using the parallelism on the GPU, but also by optimizing the length of the prime-probe loop on the two communication ends.

B. Building Custom Timer

Access to a high-resolution timer is essential to the ability to carry out cache based covert channels; without it we are unable to discriminate a cache hit from a cache miss, which is the primary phenomena used in the communication. Although Intel based integrated GPUs have a timer, by default, the manufacturer does not provide an interface to query it in OpenCL based applications. OpenCL programs executing on Intel devices are compiled using the Intel graphics compiler *(IGC)* [4], [16]. In debug mode, it is possible to query an overloaded timer function in the program. This is not available to the programmer in default mode and requires a superuser's permission for installation. In our end-to-end covert channel threat model, the attacker has no privileged access. Therefore, we need to come up with an alternative approach to measure the access latency within the GPU application.

We leverage GPU parallelism and hardware Shared Local Memory *(SLM)* to build the custom timer. Shared local memory in Intel based iGPUs is a memory structure, shared across all EUs in a subslice. 64 Kbytes of shared local memory is available per subslice. Shared local memory is private to all the threads from a single work-group. We launch a work-group for which a certain number of threads are used to conduct the attack and the rest of the threads are used to increment a counter value stored in shared memory. The threads that are responsible for carrying out the attack read the shared value as timestamps before and after the access to measure the access time (the principle of this technique was used in CPU attacks on the ARM where the hardware time is not available in user mode [30]). Due to branch divergence within the wavefronts (SIMD width of threads), the execution of two groups of threads in a single wavefront gets serialized. To avoid such effects, we use the threads in the first wavefront to access the cache, and threads in other wavefronts of the same work-group to count. Note that all of these threads (from several wavefronts) form a single work-group, assigned to the same subslice, and are able to use the same shared memory.

Each LLC cache set consists of 16 ways that can be probed in parallel from the GPU using 16 threads (thread id 0 - 15). However, the timer should start from wavefront boundary *i.e.* above 32 threads (thread ID>31) to avoid branch divergence. Accordingly, the threads involved in conducting the attack are threads 0 to 15 while the counter increment is implemented by threads 32 and above to the end of work-group. Ideally only 2 wavefronts can be used one for probing and one for the timer. However, we found out that the timer resolution obtained by using a single wavefront is not adequate to distinguish between access latency of different memory hierarchy levels; we used all remaining 224 threads to implement the counter.

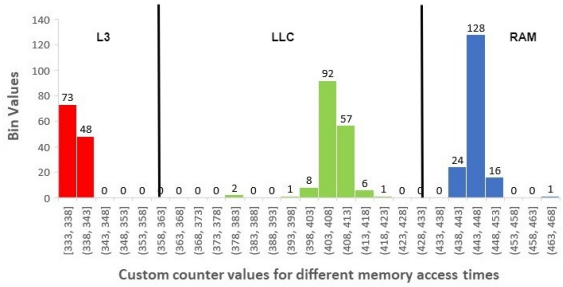

Fig. 4: Custom Timer Characterization

Algorithm 1 demonstrates the custom timer code inside the GPU kernel. Data accessed from the iGPU, using OpenCL, can get cached into the LLC and the L3. To conduct a covert channel attack, the attacker needs to distinguish 3 levels of access time, *i.e.* system memory, LLC and L3. In line 1 of algorithm 1, the variable *volatile __local counter* is declared, which is used as the timer. The *volatile* keyword makes sure that the counter variable is not cached inside the thread's registers. The timer variable is declared in the shared memory of the device using the *__local* keyword. Shared memory uses a separate data path than that used for accessing L3, which

Algorithm 1: Custom Timer Algorithm

```
1  volatile __local counter
2  cl_uint start,end,idxVal
3  cl_ulong average
4  cl _float access_time
5  if thread_ID>SIMD length then
6      for i = 0; i < n; i = i + 1 do
7          atomic_add(counter,1)
8      end
9  else
       /* Measure time over x accesses  */
10     if thread_ID<16 then
11         average= 0
12         idxVal = idx_buffer[thread_ID]
13         for i = 0; i < x; i = i + 1 do
14             start = atomic_add(counter,0)
15             idxVal = data_buffer[idxVal]
16             end = atomic_add(counter,0)
17             average + = end − start
18         end
19         access_time = (cl _float)(average/x)
       /* Clear data from L3 but not LLC
          */
       /* Repeat 5 to 19 for LLC access */
       /* Repeat 5 to 19 again for L3
          access                        */
```

makes sure that there is no resource contention that can lead to erratic counter updates. To test the custom timer, we launched a kernel with 1 work-group consisting of the *max* number of threads per work-group. Threads over a single wavefront are used to increment the counter atomically as shown in lines 5 - 8 in the *if* section of the algorithm. Atomic operation on the variable ensures that the variable is accessed and incremented properly. In lines 9 - 18, the data is accessed and the value of the counter is read atomically. A number (x) of memory accesses is timed and averaged. The first access represents the measurement from the system memory. To measure the access time from the LLC, the data is cleared from the L3 but made sure that it is not cleared from the LLC and then 5 - 19 is repeated to measure the access time from LLC. Now as the data is both cached in LLC and L3, repeating steps 5 - 19 yields the L3 access time.

Figure 4 shows our experiment with the timer measuring access time from the different levels of the hierarchy (shown in different colors). The access times obtained from the counter values are clearly separated enabling us to distinguish between accesses from the three levels of hierarchy.

C. Building LLC Conflict Sets from both CPU and GPU

The next challenge in the attack is the formation of eviction sets (addresses which has the same cache set) to be able to prime and probe the sets used in the attack [43]. We first briefly describe this process from the CPU side which is similar to

other attacks on the LLC. However, building an eviction set from the GPU side in a way that is compatible with the sets built on the CPU side presents challenges that we overcome by leveraging OpenCL Shared Virtual Memory (SVM) and zero copy feature.

Deriving LLC conflict sets from the CPU: Modern LLCs are divided into a number of slices that vary with the processor architecture. The cache slice selection depends on a complex index hashing scheme designed to evenly distribute the addresses across the slices. The Intel architecture that we are using has 8 MB last level cache divided into 4 slices of 2 MB each. The cache is 16-way set associative, with 64-byte cache lines (a total of 2048 cache sets per slice). We use approaches proposed by prior work [20], [27], [34], [51] to reverse engineered index hashing. On our processor, we discover that the index hashing algorithm selects a slice using 2 bits computed as follows. We use huge pages, which are available to user-level code, to avoid having to resolve the virtual to physical mapping and simplify the attack.

$$S_0 = b_{36} \oplus b_{35} \oplus b_{33} \oplus b_{32} \oplus b_{30} \oplus b_{28} \oplus b_{27} \oplus b_{26} \oplus b_{25} \oplus b_{24} \oplus b_{22} \oplus b_{20} \oplus b_{18} \oplus b_{17} \oplus b_{16} \oplus b_{14} \oplus b_{12} \oplus b_{10} \oplus b_6 \quad (1)$$

$$S_1 = b_{37} \oplus b_{35} \oplus b_{34} \oplus b_{33} \oplus b_{31} \oplus b_{29} \oplus b_{28} \oplus b_{26} \oplus b_{24} \oplus b_{23} \oplus b_{22} \oplus b_{21} \oplus b_{20} \oplus b_{19} \oplus b_{17} \oplus b_{15} \oplus b_{13} \oplus b_{11} \oplus b_7 \quad (2)$$

Building GPU LLC conflict sets using SVM: Deriving the conflict set from the GPU side is complicated by the fact that the GPU has its own page tables [18] which are different from the CPU ones. Therefore, we need to form the LLC eviction set from GPU side as well. To simplify this problem, we observe that OpenCL on intel GPUs allows the programmer to allocate memory with the same virtual address space using *Shared Virtual Memory* (SVM) [14] and the same physical address space through zero-copy buffers [13] from the user level. Specifically, when a CPU process initializes and launches the GPU kernel, on shared pages the eviction set identified from the CPU side also holds for the GPU after the GPU kernel is launched. Please note that this sharing is within the address space of the process launching the GPU side of the attack; no sharing is required between the Spy and Trojan.

D. Bypassing GPU Caches and handling memory view asymmetry

One of the challenges of generating the Prime+Probe pattern from the GPU is that memory accesses are filtered through the L3 cache on the GPU side (L1 and L2 are used only for graphics workloads on this system, or otherwise they have to be bypassed as well). We address this problem by generating an access pattern to remove the LLC conflict set addresses from the GPU cache so that when they are accessed again they cause an access in the LLC (which is necessary to carry out the Prime+Probe). Thus, to be able to create such reference patterns, we must first reverse engineer the L3 cache on the GPU.

A standard approach to last level caches leverages the cache inclusiveness property [51]; in fact, relaxing inclusion has been even proposed as a defense against these attacks [26]. With inclusive caches, data evicted from the lower level of caches also gets evicted from the higher level caches. As a first step of understanding the L3, we first determine whether it is indeed inclusive: we discover that it is not, providing another example of the challenges posed due to the asymmetric nature of cross-component channels. Next, we reverse engineer the structure of the cache, and finally, develop conflict sets that allow us to control the traffic that gets presented to the LLC. We discover another problem that occurs due to the asymmetry of the caches: the hash function used to index the L3 GPU cache is incompatible with the hash function for the LLC. Specifically, addresses that cause eviction in a single set in the GPU cache may hash to multiple sets in the LLC, causing unexpected self-interference. We had to understand this effect to come up with access patterns that avoid self-interference. We describe our reverse engineering experiments next.

L3 inclusiveness: To check whether the L3 is inclusive, we carried out the following experiment. We create a buffer shared by the CPU and the GPU and identify a set of *n* addresses that are accessed first by the GPU. Initially, the caches are cold, and the data is brought from memory and cached in both LLC and L3. Next, the CPU accesses the same data bringing it into its caches, and then flushes the data removing it from all the cache levels using *clflush*. If the LLC is inclusive of the L3 cache, the removal of the flushed data from the LLC will cause back-invalidations to evict the data from the L3 cache of the GPU as well. Repeating and timing the accesses on the GPU, we observed that the data is accessed from L3, indicating that the L3 cache is not inclusive.

Constructing L3 Conflict Sets considering effect on LLC: Due to the GPU parallel execution model, the associativity of the L3 cache is substantially higher than the LLC (64-way, vs 16-way for the LLC). Due to this mismatch in size, and in the index hashing (how addresses get mapped to sets in the caches), coming up with a conflict set for LLC introduced a novel challenge: we had to find a way to spill addresses from the L3 to the LLC without causing self-interference due to the additional addresses necessary to cause eviction from the higher associativity L3. We describe our reverse engineering effort of the L3, and how we addressed this problem, next.

The L3 is organized into slices with slice of size 768 KB. A slice is further divided into 4 cache banks each configured into 128 KB of L3 cache and 64 KB of Shared Local Memory (SLM). As shown earlier in Figure 2, when SLM is declared, SLM has a dedicated access pathway, which is separated from L3–this facility was critical to enable us to build the counter in SLM without interfering with the cache accesses.

During the attack phase on the LLC, we start with the addresses that are in the same LLC set and slice (selected from the LLC eviction set earlier). In both priming and the probing phases, these addresses need to be evicted from the

L3 so that they can be observed at the LLC level during the implementation of the prime and probe. Given that the L3 is non-inclusive, the addresses cannot be evicted from the CPU counterpart, and instead need to create an eviction set on L3 from the GPU side to conduct a successful attack, we call this L3 eviction sets as *pollution sets* to discriminate them from the conflict sets needed to evict values from the LLC.

Due to the larger associativity of the GPU L3, we need a larger set of addresses in the pollution set to cause the eviction of the LLC target addresses. During these experiments, we discovered that the hash indexing function of the GPU L3 is inconsistent with that of the LLC: the pollution set for a cache set on the L3 can map to different LLC sets, causing a mismatch between the pollution set and eviction set, and introducing self-interference in the attack. We reverse engineer the relationship between them to discover that the 64 ways of the L3 are strided across 4 different sets in the LLC across the 4 LLC slices. If we blindly produce the pollution set, many of the addresses in it would also cause accesses to the target LLC set, causing self-interference and failure of the attack. Consider the target addresses t_0, t_1 to t_{15} as shown in Figure 5 that are mapped to the same LLC cache set (A) which resides on slice (0) of the LLC. These addresses also map to the same cache set in L3. During the attack phase, these addresses need to be evicted from L3 (and eventually the LLC). If we simply create an eviction set for L3, these addresses have an equal chance to hash into any of the 4 slices in the LLC at the same set position. Without controlling where they hash, this will cause interference with the target LLC set (addresses p_0, $p_1 \dots p_{64}$ on the figure also accessing the target LLC set.)

To understand the relationship between the mapping within the L3 cache and the corresponding mapping within the LLC, we first conduct a standard pointer chasing experiment to derive conflict sets within the L3 similar to prior work [22]. Once we have conflict sets in the L3 we study how they hash across the cache banks in both the L3 and LLC. The L3 cache is partitioned into 4 banks and each bank is again partitioned into 8 sub-banks. The number of sets per cache bank is 32 that requires 5 bits in the address bits for mapping. There are 4 cache banks which require additional 2 bits in the address for mapping. Each cache bank is again divided into 8 cache sub-banks which require additional 3 bits in the address. As a result, a total of 10 bits (5 bits for cache set + 2 bits for cache bank + 3 bits for sub-banks) determine the set of the L3 in which a cache line resides. We discovered that these bits are the LSB bits of the address after accounting for the cache line offset bits–that is, there is no index hashing in L3. To verify the eviction set, we gathered the addresses with the same 16 bits in the LSB and conducted the eviction set test. As the replacement policy is pseudo-LRU (pLRU) [19], accessing the other addresses multiple times (5 times or more in our experiments) guarantees stable eviction of the target address through the pLRU.

To evict target addresses from both L3 and LLC without causing interference to the LLC, we constrict the target addresses for the pollution set to hash to a different slice than the

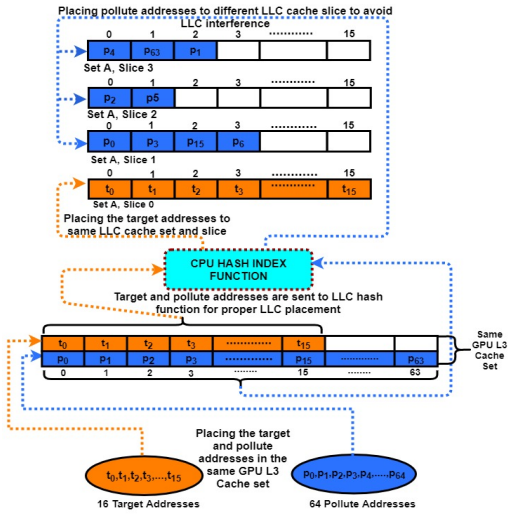

Fig. 5: LLC and L3 eviction set mapping. In this figure, we show a single set on the GPU L3 which is 64-way set associative. The target addresses need to be evicted which requires us to come up with the eviction set shown in blue. The cache lines in this set can hash into any one of 4 sets in the four slices of the LLC cache set; each LLC set is 16-ways. We have to make sure that only the target address set hash to the slice where the target set resides to avoid creating self-interference.

slice where the target LLC line resides. We accomplish this by working with the index hashing function for the LLC presented in Equations 1 and 2. That is, addresses from the LLC conflict set all hash to the same slice per these two equations; we pick the addresses in the pollution set such that they hash to the other three slices as shown in Figure 5.

E. Putting it all together–LLC Channel

As shown in Figure 6, the spy process is launched on the CPU side (CORE 0). CPU CORE 1 launches the GPU trojan process in step 1. Before each bit transfer, a handshaking takes place in steps 2 - 5 to ensure synchronization of spy and trojan. The actual bit transmission is done in steps 6 and 7. A separate LLC cache set is used in each phase of the attack. To conduct the attack, we launched one work-group that is allocated to a sub-slice. The implementation requires synchronization between threads which can only be obtained within a work-group. We launch the maximum number of threads (256 threads) permissible within a single work-group. The first 16 threads are used to perform the prime+probe attack. The threads above the wave-front boundary (32) are used to perform the custom counter increment.

The GPU initiates handshaking as data is transferred to the CPU. First, the GPU trojan process signals that it is ready to send the data. Step 2 indicates that GPU primes LLC set S_A and then probing is done from the CPU side as shown in step 3. After GPU priming is over, the CPU *spy* process probes the same set S_A as shown in 3.

The second phase of the handshaking indicates to the GPU *trojan* process by the CPU *spy* process that it is ready to

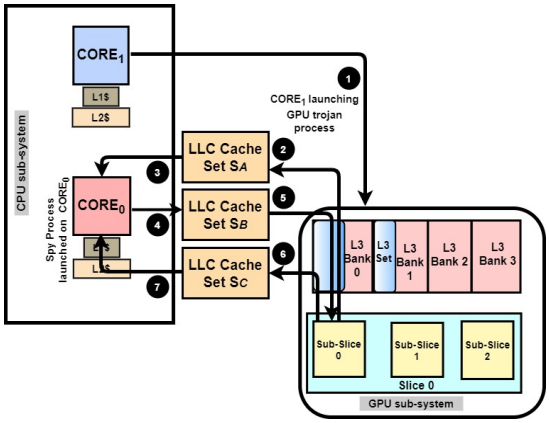

Fig. 6: LLC based CPU-GPU covert channel details

receive. The CPU primes the LLC set S_B in step 4. The GPU then probes S_B in step 5. Probing on the LLC from the GPU side requires eviction on the L3 level again. We use our custom timer to measure the delay as described in subsection III-B.

The two sides are now ready to exchange the data bit over LLC set S_C as shown in steps 6 and 7. The priming step 6 on the GPU side is similar to step 2 in the first phase of the handshaking. The probing step 7 on the CPU side is similar to step 3. Step 1 is conducted once to launch the kernel on the GPU side. Steps 2 - 7 are conducted within the kernel in a *for all* loop for the number of bits that are required to be transferred.

We also built a reverse channel where the Trojan is on the CPU communicating to a Spy on the GPU. The attack details are similar to the opposite direction channel described above, but with the roles reversed. Specifically, the CPU initiates the handshake by priming set S_A while the GPU receives it by probing the same set. Next, the GPU sends a ready to receive signal by priming set S_B, and the CPU probes the same set to receive it. Finally, the CPU sends the communication bit to the GPU using set S_C.

IV. Contention Covert Channel

Even with absent direct sharing of stateful microarchitectural components (such as the LLC), contention may arise when the two components share a bandwidth or capacity limited microarchitectural structure such as buses or ports. In such situations, measurable contention can also be achieved if the two processes running on the two components access the same structure concurrently (observing slowdowns). Although there are likely to be a number of such shared contention domains on our system, we implement an attack based on contention on the ring bus connecting the CPU and GPU to the LLC. Specifically, when both the CPU and GPU generate traffic to the LLC, they each observe delays higher than when only one of them does, providing a way to communicate two states by either creating contention or not.

Since contention relates to concurrent use of the shared resource, it requires accurate synchronization between the two sides, which is challenging in the presence of the clock frequency disparity between CPU and GPU. The CPU runs at 4x the speed of the GPU and the data access delay cannot be observed if the GPU data access is lower than a limit. Through our systematic study, we identified the parameters that contribute in creating a robust contention based channel with a low error rate and high bandwidth. We also devised a parameter that controls the frequency disparity between the computational resources. We describe the attack in more detail in the remainder of this section.

Attack Overview: The attack creates contention on the ring bus between the CPU and GPU used to access the LLC. During the attack, the CPU and GPU generate addresses chosen from their own pre-allocated memory buffers. The CPU and GPU buffers are chosen to map to different LLC sets to avoid LLC conflicts distorting the contention signal. With the two processes accessing disjoint sets in the cache, the contention occurs strictly on the shared resources leading to the LLC.

The attack overview is present in Figure 7. The CPU process is launched in CORE 0 and a GPU process is launched in CORE 1 as shown in steps 1 and 2. The processes launch each carries out data allocation and initialization. The trojan process launched on CORE 1 launches the GPU kernel as shown in step 3. The data is accessed by the CPU and GPU simultaneously. The first access will warm up the cache and bring the CPU and GPU data to the LLC, steps 4 and 5. Subsequent memory accesses would hit the LLC and generate contention among the shared resources as shown in step 6. This contention among the shared resources gets reflected during the data access by the CPU.

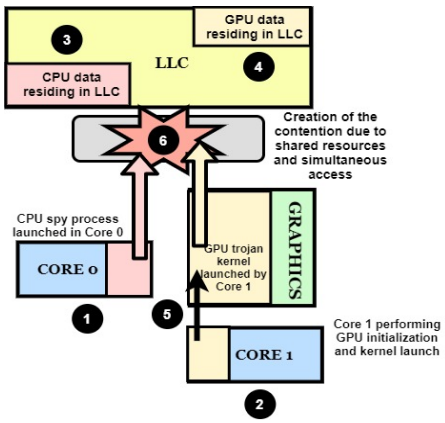

Fig. 7: Contention channel attack methodology

Contention Channel Implementation: To build the covert channel, we need to identify different parameters that contribute towards building the channel to be able to systematically create and optimize the attack. For the CPU, T_{CPU} is the time required to access S_{CPU} bytes of data. With the simultaneous access from the GPU, the access time is increased by T_{OV}. The total time $T_{TOTAL_{CPU}}$ required to access the data from the CPU during the simultaneous GPU access is given in Equation 3. The overhead created due to simultaneous access is a function of the S_{GPU} bytes of data

accessed by GPU, a number of threads launched $NUM_{Threads}$ and an Iteration Factor I_F reflecting how many iterations the data is accessed as shown in Equation 4. One constraint is to keep both, CPU and GPU data, in the last level cache. The total of S_{GPU} and S_{CPU} has to be less than the total size of the last level cache, as shown in Equation 5. Another constraint is that the LLC sets that are mapped to the CPU buffer should not coincide with the sets that are mapped to the GPU buffer, as shown in equation 6. The last two constraints ensure that we avoid LLC misses and only measure the latency due to contention on the ring bus. Communicating 1 and 0 through the contention based channel is also related to the iteration factor I_F. When 1 needs to be communicated then the GPU accesses S_{GPU} bytes of data for I_F number of times to create the contention. To communicate 0 the GPU does no access.

$$T_{TOTAL_{CPU}} = T_{CPU} + T_{OV} \tag{3}$$

$$T_{OV} = f(I_F).f(S_{GPU}).f(NUM_{Threads}) \tag{4}$$

$$\text{s.t.} \quad S_{CPU} + S_{GPU} \ll S_{LLC} \tag{5}$$

$$S_{CPU} \cap S_{GPU} = \emptyset \tag{6}$$

On the CPU side of the attack, a buffer size of S_{CPU} bytes has been created. The accesses are done at an offset of cache line size of 64b. So the number of accesses is equivalent to the number of cache lines in the allocated buffer. The data is accessed in a random pointer chasing manner to lower prefetching effects that may cause replacements of either the CPU or GPU data in the LLC. First, LLC is warmed up. The subsequent accesses would be serviced from the LLC. The size of the buffer is chosen to ensure that the data is evicted from local caches but not from the LLC. Each access time is measured by *clock_gettime()*.

On the GPU side, the number of cache lines needed to be accessed is divided among the number of threads launched. The number of memory addresses that each thread needs to access, *numElsPerThread*, is shown in Equation 7.

$$numElsPerThread = \frac{number\ of\ cache\ lines}{number\ of\ threads} \tag{7}$$

One of the novel problems presented by asymmetric covert channels is that the two sides have an asymmetric view of the resource; for example, the GPU and CPU operate at different frequencies, and the GPU must overflow the L3 cache to generate an access to the LLC, which unlike the CPU side requires deriving different conflict sets due to the different indexing scheme. Without calibration, this mismatch can lead to inefficient communication, reducing bandwidth and increasing errors. We introduce the notion of *Iteration Factor* I_F to allow us to align the two ends of the channel as shown in equation 4. For a given GPU buffer size, the execution time varies based on the number of work-groups launched. I_F (the number of iterations the data is accessed on the GPU) ensures that the ratio of GPU to CPU execution time is near 1.

V. Evaluation

In this section, we evaluate the two covert channels in terms of channel bandwidth and error rate.

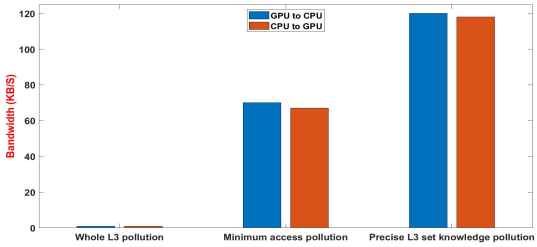

Fig. 8: LLC bandwidth (different L3 eviction strategies)

LLC-based Covert Channel: The GPU L3 cache is non-inclusive which requires it to be filled to overflow and access the LLC. Figure 8 shows the bandwidth of the channel on both directions (CPU-to-GPU and GPU-to-CPU channels) based on different strategies to overflow the L3. The naive way to establish the covert channel can be performed by clearing the whole L3 cache (we can use the GPU parallelism to accelerate this process); the advantage here is that we do not have to reverse engineer the L3 organization. However, clearing up the whole L3 data cache of 512 KB, even with thread-level parallelism, substantially reduces bandwidth. Figure 8 shows the bandwidth of the LLC based covert channel is 1 kb/s, when the whole L3 is cleared in every iteration. We improved this with our precise conflict set construction that eliminates interference from L3 to LLC which we described earlier in the paper. This bandwidth we achieved using this technique, is 70 kb/s for GPU-to-CPU channel (67 kb/s for CPU-to-GPU channel). Further optimization was achieved by carrying out the complete L3 reverse engineering and creating its eviction sets, determining the addresses that are in the same L3 set for precise eviction of the target addresses increasing bandwidth to 120 kb/s (118 kb/s for CPU-to-GPU channel). The error percentage observed was 2% (6% for CPU-to-GPU channel). We achieved a stable channel with a low error rate and high bandwidth through our optimization of precise L3 set eviction. However, the error rate is higher in the case of CPU-to-GPU channel.

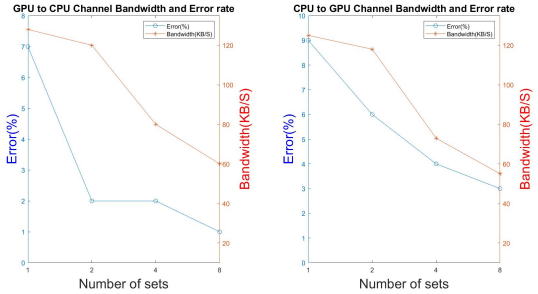

Fig. 9: Error and BW with number of LLC sets

To reduce the error rate and increase channel resilience we used multiple LLC sets. Monitoring cache misses over multiple sets provides us with better resolution than using

a single set for communication. However, the redundancy causes a reduction in available bandwidth; potentially we could have used these multiple sets to communicate multiple bits in parallel. Figure 9 shows the bandwidth and error rate with respect to the increasing of number of LLC sets. When we are using only 1 set then the error rate is 7% for GPU-to-CPU channel (9% for the CPU-to-GPU channel), which reduces to 2% as the number of sets doubled. For CPU-to-GPU channel that error rate reduces to 6%. However, the bandwidth reduces by 6.25% from 128 Kb/s to 120 Kb/s which is an acceptable reduction given the error rate reduces by more than 71%. The bandwidth reduces to 118 Kb/s from 125 Kb/s by doubling the cache set in the cases of CPU-to-GPU based channel. Increasing the number of sets does not provide any improvement in the error rate. However, the bandwidth reduces at a steady rate. In our attacks, we used 2 sets for all the 3 stages of attack resulting in using 6 LLC cache sets.

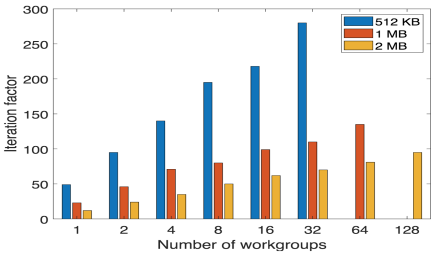

Fig. 10: Iteration Factor for different buffer sizes

Contention-based Covert Channel: CPU and GPU access the LLC using asymmetric pathways and computational models. This impacts the success rate of the communication between the two asymmetric sides. We introduce the concept of *Iteration Factors* to match the rate of communication between the two sides (as discussed in Section IV). Figure 10 shows the optimal iteration factor: keeping the CPU buffer size constant, as the GPU buffer size increases, the factor reduces correspondingly to enable overlap between the two sides.

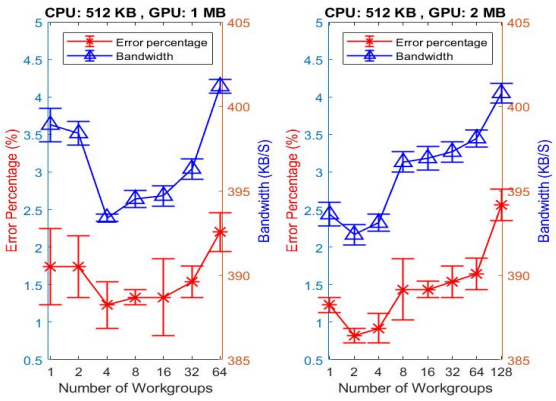

Fig. 11: Bandwidth and error for bus-based channel

As discussed in Section IV, in our contention based covert channel, buffer size on both CPU and GPU side and the number of work-groups that access to the GPU buffer, affect the contention pattern and consequently the channel bandwidth and error rate. We perform a search on the parameter space to obtain a channel with a low acceptable error rate and high bandwidth. Figure 11 shows the evaluation results of the contention based covert channel. The different graphs are for different GPU buffer size and a constant CPU buffer size of 512 KB. The GPU buffer sizes that we considered are 1 MB and 2 MB. Each result shows a confidence interval of 95% over 1000 runs of the experiment. The bandwidth and the error rate are shown for different number of work-groups (in the X-axis). We obtained an error rate that is lower than 2% for more than 90% of the configuration space. The lowest error rate that we obtained is 0.82% for CPU buffer size of 512 KB, GPU buffer size of 2 MB, and number of work-groups of 2. We can observe that the bandwidth follows the pattern of the error rate (lower bandwidth for low error rate).

VI. Tracking User's Cache Activity

To demonstrate the efficacy of our attack model we designed a proof of concept prime+probe based side-channel attack that spies on the LLC sets from the GPU side and observes the CPU side cache activities. We represent the cache activities in the form of memorygram [39] where the cache misses distribution are demonstrated over a period of time. In our experimental design, we have monitored cache sets in parallel from the GPU side measuring the cache activities occurring on the CPU side. A thread block is assigned a cache set under observation. On the CPU side, we have accessed 2 cache sets in a loop. Cache set 4 is accessed first in a loop with almost no delay in the subsequent accesses. After that cache set 2 is accessed with a fixed delay in the subsequent accesses of the cache set. Figure 12 shows the memorygram of 4 LLC sets. The X-axis represents time steps and Y-axis represents the cache set number. Each of the yellow vertical lines represents cache miss on that particular time step. The memorygram shows the interested sets as well as it's neighboring sets to better visualize the rate of cache activities.

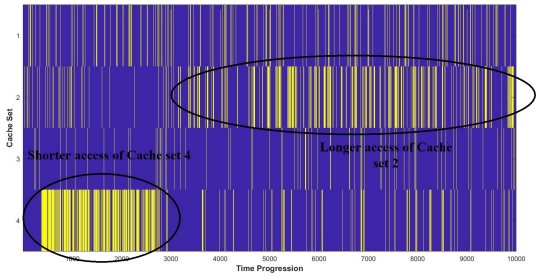

Fig. 12: Memorygram of the cache activities of 4 cache sets

The cache activity pattern of the application is seen in the memorygram. For example, a high number of accesses occurs in a shorter number of time steps (500-3000) in cache set number 4. This due to high frequency access of the set in a loop without any delay in between accesses. After time step 3000, the activity reduces in cache set number 4 as the CPU

side application now starts accessing cache set number 2 with a delay in subsequent accesses. The access starts after 3000 and continue till time step 10000. Other accesses observed are due to noise from other activities in the system. From the memorygram it is evident that the user-level activities can be monitored from the GPU side.

VII. Possible Mitigations

We believe classes of defenses that have been developed against other microarchitectural covert channels can also potentially apply to cross-component attacks on heterogeneous systems. These solutions include: (1) Static or dynamic partitioning of resources [6], [21], [29], [31], [46], specifically the LLC. These partitioning schemes can be extended to support different processors in heterogeneous systems. If the Spy and Trojan use different partitions of the cache, they are not able to replace each other's cache lines; and (2) Eliminating the contention among processes by traffic control in memory controllers [42], [44], such that memory requests from each processor are grouped into the same queue and possibly access the same memory bank/port. Prior work [25] demonstrated that an efficient memory scheduling strategy and isolating the CPU memory requests from the GPU memory requests will improve the system performance, since memory requests from the GPU seriously interfere with the CPU memory access performance. Such isolation can be applied to the ring bus connecting the CPU and GPU to the LLC (with LLC partitioning in place). Other solutions such as adding noise to the timer may also apply [33]. However, we build our customized timer using hardware resource (shared memory) available on GPU, so interfering with timing is not straightforward.

VIII. Related Work

Microarchitectural covert-channel and side-channel attacks have been widely studied on different resources on CPUs including the L1 cache [3], [7], [41] and shared LLC in multicore CPUs [11], [27], [30], [32], [50], [52]. A concurrent work [40] develops contention based side channel attacks on the CPU ring interconnect between multiple cores.

Some recent works demonstrates that GPUs are also vulnerable to microarchitectural covert and side-channel attacks. These works have been proposed on discrete GPUs with a dedicated memory. Jiang et al. [23], [24] present architectural timing attacks from the CPU to the GPU by exploiting key dependent memory coalescing behavior. Naghibijouybari et al. [36] construct several types of covert channels on different resources within a GPU. Naghibijouybari et al. also demonstrate a series of end-to-end GPU side channel attacks covering the different threat scenarios on both graphics and computational stacks, as well as across them [37]. They implement website fingerprinting, through GPU memory utilization API or GPU performance counters, track user activities as they interact with a website or type characters on a keyboard. In addition, they develop a neural network model extraction attack. On the defense side, Xu et al. [48] proposed a GPU-specific intra-SM partitioning scheme to isolate contention between victim and spy and eliminate contention based channels after detection.

All of these microarchitectural attacks and defenses have been proposed on a single processor (CPU or discrete GPU). In this paper, for the first time, we develop microarchitectural covert channels in more widely used integrated CPU-GPU systems. There have been a limited number of Rowhammer attacks on heterogeneous systems (but not timing attacks): Weissman et al. [47] study rowhammer attacks on heterogeneous FPGA-CPU platforms. The integrated GPU is available through APIs such as WebGL [2] even for remote Javascript programs. Frigo et al. [10] use WebGL timing APIs to implement rowhammer attack on integrated GPUs in mobile SOCs. They use WebGL timer to find the contiguous areas of physical memory to conduct the rowhammer. In response to this attack, both Chrome and Firefox disabled the WebGL timer [12]. We build a high resolution timer through iGPU hardware which (1) is precise to measure cache hits/misses to conduct cache attacks; and (2) can not be easily disabled.

IX. Concluding Remarks

We present the first microarchitectural covert channel attacks that span two different components in an SoC. Each component has a different view of the shared resource that they use to create contention, and typically a different computational model. We show that such attacks introduce novel difficulties that arise due to this asymmetry. For example, the LLC is inclusive on the CPU side, but non-inclusive on the GPU side. Moreover, the indexing of the GPU cache hierarchy is different from that of the LLC; as we create conflict sets to overflow the L3 on the GPU, we run the risk of creating self-interference with other sets on the LLC. We also needed to calibrate the communication loops to improve the bandwidth given the asymmetric pathways to access the channel. Having experience with these channels improves our understanding of the threats posed of microarchitectural attacks beyond a single component which is a threat model increasing in importance as we move increasingly towards heterogeneous computing platforms. We created two working channels: a Prime+Probe channel targeting the LLC, and a contention based channel exploiting contention on the shared access pathway to the LLC. Creating the channels required overcoming a set of challenges that we believe will be representative of those needed for cross-component attacks. Both channels achieve high bandwidth and low error rates.

X. Acknowledgement

Pacific Northwest National Laboratory is operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract No. DE-AC05-76RL01830. This work was supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR) through the Center for Advanced Technology Evaluation (CENATE) project, Contract #66150B. This work is also partially supported by National Science Foundation grants CNS-1619450, CNS-1955650 and CNS-2053383.

REFERENCES

[1] "OpenCL Overview, Khronos Group," 2018, https://www.khronos.org/opencl/.

[2] "WebGL Overview, Khronos Group," 2018, https://www.khronos.org/webgl/.

[3] B. B. Brumley and R. M. Hakala, "Cache-timing template attacks," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 667–684.

[4] A. Chandrasekhar, G. Chen, P.-Y. Chen, W.-Y. Chen, J. Gu, P. Guo, S. H. P. Kumar, G.-Y. Lueh, P. Mistry, W. Pan, T. Raoux, and K. Trifunovic, "Igc: The open source intel graphics compiler," in *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO 2019. IEEE Press, 2019, p. 254–265.

[5] J. Chen and G. Venkataramani, "Cc-hunter: Uncovering covert timing channels on shared processor hardware," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2014.

[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] D. A. O. Eran Tromer and A. Shamir, "Efficient cache attacks on aes, and countermeasures," in *Journal of Cryptology*, 2009, pp. 667–684.

[8] D. Evtyushkin and D. Ponomarev, "Covert channels through random number generator: Mechanisms, capacity estimation and mitigations," in *CCS*, 2016.

[9] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Understanding and mitigating covert channels through branch predictors," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 1, p. 10, 2016.

[10] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: Accelerating microarchitectural attacks with the gpu," in *Proceedings of IEEE Symposium on Security and Privacy*, 2018, pp. 357–372.

[11] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 897–912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss

[12] Mozilla, "WebGL timer extension," 2020, https://developer.mozilla.org/en-US/docs/Web/API/EXT_disjoint_timer_query.

[13] Intel. (2014) Getting the most from opencl™ 1.2: How to increase performance by minimizing buffer copies on intel® processor graphics. [Online]. Available: https://software.intel.com/sites/default/files/managed/f1/25/opencl-zero-copy-in-opencl-1-2.pdf

[14] Intel. (2014) Opencl 2.0 shared virtual memory overview. [Online]. Available: https://software.intel.com/en-us/articles/opencl-20-shared-virtual-memory-overview

[15] Intel. (2015) Intel processor graphics gen9 architecture. [Online]. Available: https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

[16] Intel. (2019) Intel graphics compiler. [Online]. Available: https://github.com/intel/intel-graphics-compile

[17] Intel. (2019) Intel processor graphics gen11 architecture. [Online]. Available: https://software.intel.com/sites/default/files/managed/db/88/The-Architecture-of-Intel-Processor-Graphics-Gen11_R1new.pdf

[18] Intel. (2019) Intel® open source hd graphics and intel iris™ plus graphics programmer's reference manual: Volume 5: Memory views. [Online]. Available: https://01.org/sites/default/files/documentation/intel-gfx-prm-osrc-kbl-vol05-memory_views.pdf

[19] Intel. (2019) Intel® open source hd graphics and intel iris™ plus graphics programmer's reference manual: Volume 7: 3d-media-gpgpu. [Online]. Available: https://01.org/sites/default/files/documentation/intel-gfx-prm-osrc-kbl-vol07-3d_media_gpgpu.pdf

[20] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Systematic reverse engineering of cache slice selection in intel processors," in *2015 Euromicro Conference on Digital System Design*. IEEE, 2015, pp. 629–636.

[21] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "Qos policies and architecture for cache/memory in cmp platforms," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 25–36. [Online]. Available: https://doi.org/10.1145/1254882.1254886

[22] 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)*, 2019, pp. 29–41.

[23] 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

[24] 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.

[25] M. W. Juan Fang and Z. Wei, "A memory scheduling strategy for eliminating memory access interference in heterogeneous system," in *The Journal of Supercomputing*, vol. 76, 2020, p. 3129–3154.

[26] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.

[27] M. Kayaalp, D. Ponomarev, N. Abu-Ghazaleh, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proceedings of the 53th Annual Design Automation Conference*, June 2016.

[28] S. K. Khatamifard, L. Wang, S. Köse, and U. R. Karpuzcu, "A new class of covert channels exploiting power management vulnerabilities," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 201–204, 2018.

[29] 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.

[30] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 549–564. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp

[31] 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.

[32] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Security and Privacy (SP)*, May 2015.

[33] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2012, pp. 118–129.

[34] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 48–65.

[35] 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.

[36] H. Naghibijouybari, K. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpus," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2017, pp. 354–366.

[37] 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: ACM, 2018, pp. 2139–2153. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243831

[38] Nvidia. (2019) Cuda c++ programming guide. [Online]. Available: https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf

[39] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1406–1418. [Online]. Available: https://doi.org/10.1145/2810103.2813708

[40] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security*

21). USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/paccagnella

[41] C. Percival, "Cache missing for fun and profit," in *BSDCan*, 2005.

[42] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding information leakage in the memory controller with fixed service policies," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Dec. 2015.

[43] P. Vila, B. Köpf, and J. F. Morales, "Theory and practice of finding eviction sets," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 39–54.

[44] Y. Wang and G. E. Suh, "Timing channel protection for a shared memory controller," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2014.

[45] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture." in *Computer Security Applications Conference (ACSAC)*, 2006.

[46] 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.

[47] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms," in *arXiv:1912.11523*, 2020.

[48] 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

[49] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2018.

[50] 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)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[51] Y. Yarom, Q. Ge, F. Liu, R. B. Lee, and G. Heiser, "Mapping the intel last-level cache." *IACR Cryptology ePrint Archive*, vol. 2015, p. 905, 2015.

[52] M. K. R. Yinqian Zhang, Ari Juels and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 990–1003. [Online]. Available: https://doi.org/10.1145/2660267.2660356