UNIVERSITY OF CALIFORNIA
RIVERSIDE

Covert- and Side-Channel Attacks on Integrated and Distributed GPU Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Sankha Baran Dutta

March 2022

Acknowledgments

My journey of the doctoral program is reaching it's completion and I can still remember the beginning of my journey. All those years enriched me with the most valuable qualities that I probably wouldn't have in any other way. All those acquired experiences wouldn't have been possible without few people. Without their inspiration and support this journey wouldn't have been possible.

Firstly, I would like to thank my dissertation committee members for their support and guidance towards my degree completion. I appreciate for their time and feedback.

My research would not have been possible without the guidance of my advisor Professor Nael Abu-Ghazaleh. His advice and supervision is the reason for bringing my work to it's completion. His attention to the details and his passion in a research topic is something that I admire immensely. His influence is not only limited to the technical aspects only. A good research gets regulated by not only the technical aspect and several other factors which I learnt from my professor. His motivation and influence in my PhD is invaluable and I cannot appreciate enough for his contribution in my research.

I have performed my research in conjunction with Pacific Northwest National Lab and I am greatly indebted to all the personnel I came to know from the organization during my research tenure. Specifically, I would like to acknowledge Dr Andres Marquez and Dr Kevin Barker for supporting me throughout my research endeavor. Dr Marquez provided a huge support in every way possible. I also enjoyed discussion with the other members of CENATE group like Dr. Joseph B Manzano, Dr Joshua D Suetterlein and Dr Ang Li. I absolutely cherish my collaboration with the group and I thank them immensely for their support.

I had wonderful opportunity to work with some of the finest researchers like Prof. Khaled Khasawneh, Dr. Hodjat Asghari Esfeden, Dr. Shafiur Rahman,Prof. Fatemah Alharbi, Ahmed Abdo, Esmaeil "Reza" Mohammadian Koruyeh, Abdulrahman Bin Rabiah, Jason Zellmer, Quazi Mishkatul Alam, Mahbod Afarin, Chao Gao, Md. Abdullah Al Mamun and Yicheng Zhang at our security group. I thoroughly enjoyed to have them as my friend and colleague with whom I was able to share my personal and professional life. SA special thank would go to Professor Hoda Naghibijouybari for her contribution in my work. Her contribution to my work is invaluable and I appreciate for her help.

My acknowledgement would be incomplete without thanking my wife and partner Bandhuli Sen. This journey comes with it's own uncertainties and I cannot appreciate enough for your steadfast presence in all of those moments. You have shared all the ups and downs equally and kept on motivating me to move forward. I cannot appreciate you enough for your presence which made this journey a lot easier.

I would also like to thank all my childhood friends for being there with me throughout my life and shaping my life in the best way possible. I am extremely lucky to have you guys as my friends.

Finally, I would like thank my parents for providing me an excellent education and making me capable so I could reach this position. I thank you both for everything.

To my wife, friends and parents for their presence and support.

ABSTRACT OF THE DISSERTATION

Covert- and Side-Channel Attacks on Integrated and Distributed GPU Systems

by

Sankha Baran Dutta

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2022
Dr. Nael Abu Ghazaleh, Chairperson

Graphics Processing Units (GPUs) were introduced as peripheral devices for accelerating graphics and multi-media workloads. The inherent parallel computational model of graphics rendering makes GPUs suited for other workloads that operate on massive data and that are throughput oriented. To enable such general purpose applications to leverage GPUs, Nvidia introduced Compute Unified Device Architecture (CUDA) that allowed general purpose computing on GPUs. GPUs are currently ubiquitous in all computing platforms, from portable devices to high-end servers on the cloud. Customarily, GPUs are available in a discrete form where the GPUs are connected to rest of the system as a peripheral device with its own separate memory.

This dissertation explores the security of emerging classes of GPUs to a type of microarchitectural attacks – those targeting the architecture of the computing devices– called covert- and side-channel attacks. The last decade has seen a rise in these types of attacks, primarily targeting CPU microarchitectural structures. Specifically, in these attacks an attacker uses malicious software that exploits resource sharing on the underlying architecture to either communicate secret data through covert channel or to extract information from the victim application indirectly by observing measurable contention. While the majority of these attacks have targeted conventional CPU resources, some recent work has shown that GPUs are also vulnerable to this type of attack.

This dissertation explores the feasibility of these attacks, and demonstrates several end to end attacks in two emerging GPU domains: (1) Integrated GPUs: GPUs are also increasingly offered as integrated processors on the same chip as CPUs, enabling lower form factors and cost, while providing support for multi-media workloads which are important for consumer machines. Chip manufacturers like Intel have GPU integrated in the same die as the CPU. GPUs are currently available in distributed form as well where multiple GPUs are connected by proprietary connectors. We show that attacks from the GPU on the CPU and vice versa are possible in these environments. To enable these attacks, we have to solve a number of unique challenges many of which originate due to the heterogeneous view of the shared resources between the CPU and integrated GPU. This is the first known attack of this type that crosses heterogeneous components, which has important implications to future heterogeneous computing designs; and (2) Multi-GPU high performance servers: on the other end, there is an emerging class of multi-GPU systems targeted at high-performance applications in general, and machine learning workloads in particular. We demonstrate a number of covert and side-channel attacks on this type of environment, exploiting remote sharing of GPU caches.

Our cache based covert channel obtained a bandwidth of 120 KB/s and 3.2 MB/s in the integrated and distributed GPU settings. We have also demonstrated side channel attacks in both the computing environments. Our work substantially expands our understanding of the threat models facing these important and emerging systems, and helps define how future systems should be built to mitigate these attacks.

Contents

List of Figures **xii**

List of Tables **xiv**

1 Introduction **1**
1.1 Motivation–Microarchitectural attacks 3
1.2 Challenges 5
1.3 Contribution 7
1.4 Thesis Organization 9

2 Background **11**
2.1 Discrete GPU architecture 12
2.1.1 Nvidia Multi-GPU DGX box 14
2.2 Integrated GPUs 16
2.3 Timing based Prime+Probe Cache Covert and Side Channel Attacks 19

3 Heterogeneous Attacks on Shared Microarchitectural Resources: Integrated GPU-CPU cross component attacks **22**
3.1 Introduction 23
3.2 Background and Threat Model 26
3.2.1 Intel Integrated CPU-GPU systems 26
3.2.2 Threat Model 29
3.3 LLC-based Covert Channel 31
3.3.1 Attack Overview and Challenges 31
3.3.2 Building Custom Timer 34
3.3.3 Building LLC Conflict Sets from both CPU and GPU 38
3.3.4 Bypassing GPU Caches and handling memory view asymmetry . . . 39
3.3.5 Putting it all together–LLC Channel 45
3.4 Evaluation 46
3.5 Side Channel: Tracking user's cache activity 49

4 Contention based covert channel on Integrated CPU-GPU systems 51

4.1 Introduction 52

4.2 Threat Model 53

4.3 Attack Details 54

4.4 Contention Channel Implementation: 55

4.5 Evaluation: 57

5 Covert and Side Channel Attacks on Multi-GPU Systems 60

5.1 Reverse Engineering Cache Organization 61

5.1.1 Caching organization and timing properties 62

5.1.2 Determining Cache Eviction Sets 65

5.2 Covert Channel Attack and Challenges 72

5.2.1 Aligning the cache sets 73

5.2.2 Putting it together: Covert channel attack 75

5.3 Covert Channel Evaluation 78

5.4 Side Channel Attack 80

5.5 Noise Mitigation 84

5.6 Possible Mitigations 85

6 Spying on Deep Learning Framework in Multi-GPU systems 87

6.1 Introduction 88

6.2 Cache based Prime+Probe multi-GPU model extraction attack 89

7 Concluding Remarks 96

Bibliography 100

List of Figures

2.1 Nvidia GPU architecture 14
2.2 Nvidia GPU architecture 16
2.3 Intel CPU-GPU architecture 18
2.4 Cache PRIME+PROBE attack 20

3.1 Intel SoC architecture 27
3.2 Intel integrated GPU architecture 28
3.3 LLC based CPU-GPU covert channel overview 32
3.4 Custom Timer Characterization 37
3.5 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. 44
3.6 LLC based CPU-GPU covert channel details 45
3.7 LLC bandwidth (different L3 eviction strategies) 47
3.8 Error and BW with number of LLC sets 48
3.9 Memorygram of the cache activities of 4 cache sets 49

4.1 Contention channel attack threat model 54
4.2 Contention channel attack methodology 55
4.3 Iteration Factor for different buffer sizes 58
4.4 Bandwidth and error for bus-based channel 59

5.1 Accessing a remote GPU's memory through NVLink 63
5.2 Local and remote GPU access time 64
5.3 Pointer chase experiment for eviction set determination 66
5.4 Validating the eviction set determination 70
5.5 Eviction set aliasing issue 71
5.6 Eviction set alignment among multiple processes 74

5.7 Cross GPU covert channel attack 77
5.8 Bandwidth and Error rate in covert channel 79
5.9 Cross GPU covert message received by spy process 80
5.10 Memorygram of 6 applications 82
5.11 Confusion Matrix (1200 samples per class). BS (Black Scholes), HG (Histogram), MM (Matrix Multiplication), QR (Quasi Random), VA (Vector Addition), WT (Fast Walsh Transform) 83

6.1 Prime+Probe based multi-GPU model extraction attack's threat model . . 90
6.2 Memorygram of the MLP application 92
6.3 Histogram of the cache misses of the MLP application 93
6.4 Memorygram for a two-epoch experiment 95

List of Tables

5.1 L2 cache architecture 70

6.1 Average misses over all cache sets 91

Chapter 1

Introduction

The end of Dennard's scaling where we can no longer maintain the power density of chip design as the feature size of transitors continues to shrink, have necessitated a switch from a single core to multi-core architectures [10]. Another side effect of this trend is that the only way to continue to gain performance is to introduce specialized accelerators targeted towards specific important computational domains or classes of computations. In addition to traditional computational devices such as Central Processing Units (CPU), we have seen other varied accelerators like Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), and other Application Specific accelerators. These accelerators are integrated as separate device to which computation are offloaded or could be integrated with a CPU on the same die. The goal of this dissertation broadly is to investigate the feasibility of microarchitectural attacks, which have been investigated in conventional systems, on such emerging heterogeneous devices.

GPUs are one of the most successful examples of application specific accelerators.

Due to multimedia demands, GPUs became one of the most popular accelerators. Primarily, GPUs were introduced for graphics rendering as they are ideal for conducting same operation over multiple data points which fits the gaming purposes. Graphical programming languages like OpenGL are used to program the devices and using the devices for general purpose computing was not possible. Nvidia introduced CUDA in 2007 to program the GPUs for general purpose computing. Currently, GPUs are now one of the most inherent and vital accelerators available. GPUs are now present in the top 500 supercomputers in the world. Traditionally, GPUs come as a separate device that is connected to the rest of the system through a PCIe bus.

GPUs are currently available in integrated form as well, where the GPU is integrated on the same chip as the CPU. The integrated GPUs are meant for small workload like screen rendering, but could be programmed for general purpose computing as well. GPUs have been an important computational platform enabling a variety of data intensive workloads such as deep neural networks, scientific kernels, cryptocurrency mining and many others.

The size of these workloads continues to increase: for example, training of large deep networks often requires both computational and memory resources that far exceed those of a single GPU. In response to these trends, distributed multi-GPU platforms have emerged, and are available both in cloud and high-performance computing cluster settings. For example, the Nvidia DGX series [75] offers a number of server class GPUs that are interconnected through a combination of proprietary high bandwidth interconnect (NVLink) and PCIe. DGX class machines are available on most major cloud providers. Other GPU manufacturers are also starting to offer similar products; for example, AMD's crossfire allows building relatively inexpensive multi-GPU configurations [5]. 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. As GPUs are becoming ubiquitous, from server class GPUs in major computing platforms to desktop integrated GPUs, exploring and evaluating the vulnerabilities in these modern heterogeneous and distributed platforms would be crucial.

As such platforms, as well as other heterogeneous platforms continue to be developed, it is critical to think not only about their performance, but also their security. Towards this end, the overall goal of this dissertation is to explore a class of attacks that has been demonstrated on conventional systems on heterogeneous systems at different scales: from the integrated systems on a chip present in mobile and other end user devices, to multi-GPU machines used in high performance computing servers in data centers and on the cloud. By demonstrating these attacks, and developing their principles, we hope to improve the understanding of the threats facing these systems and motivate the development of defenses and design principles that mitigate them.

1.1 Motivation–Microarchitectural attacks

Microarchitecture is a term that refers to the hardware components of a processor that are used to implement the processor logic, and optimize its performance. Microarchitectural components include the caches, branch predictors, the reorder buffer, Translation Lookside buffers, the load store queue, the reorder buffer, the data prefetcher, and many other components and policy logic that make up a computational processor. Microarchitectural attacks originate from these shared microarchitectural resources where a malicious process by accessing these shared resources is able to compromise the security or integrity of other processes. Specifically, these attacks can break the isolation guarantees enforced by the system enabling accesses across protection boundaries.

Modern systems are capable of running multiple applications concurrently by sharing the computational resources. Sharing of these resources could lead to information leakage of the sensitive data. Attackers leverages this resource sharing to extract the relevant information. The leakage channels could be broadly classified into covert and side channel attacks. In the side channel attacks, the attacker uses the architectural resource sharing to reveal a victim's secrets. In covert channel attacks, an attacker known as trojan reveals some secret information to another attacker called spy leveraging resource sharing. Any shared microarchitectural resources could be used as the potential attack surface *e.g.* cache, RAM, memory ports, interconnects, bus, etc. To conduct the attack by monitoring and measuring the competing resources, attackers uses different leakage vectors like timing, power, thermal emission, electromagnetic emission, sound, etc. Microarchitectural attacks can be broadly divided into software based attacks, the one that can be conducted through software and hardware attacks that require hardware assistance. Software based attacks are potentially more dangerous as they don't require any physical access of the device.

A large number of software based microarchitectural attacks have been demonstrated on various shared resources. For example, Prime+Probe based attacks on different cache levels [23, 43, 56, 64, 108], fault injection attacks on RAM like rowhammer attacks [22, 92, 94, 103] and recently speculative execution attacks like spectre and meltdown [48,50,54]. Attackers also demonstrated attacks using other shared resources like the busses and interconnects [82] to extract information from the victim applications. However, till now, most of the cache attacks were demonstrated within a single computing components. Though the demonstrated cache attacks were conducted on different shared resources, they were tied within same components like either a single CPU or GPU. In this scenario, the attackers have the similar view of the system and resources. However, as the computing infrastructure is becoming increasingly heterogeneous, it is important to evaluate the security aspect in the modern heterogeneous and distributed platforms. In the mentioned platforms, attackers have different view of the systems and the memory hierarchy. Also the components involved the attack have different clock frequencies compared to the previous established attacks where the attacker and the victim are placed within the same components. Through our work, we demonstrated microarchitectural vulnerabilities in the a heterogeneous environment. The next section would provide an overview of our contribution.

1.2 Challenges

Conducting a microarchitectural attack requires a detailed knowledge of the target microarchitecture. However, the microarchitectural details are often not disclosed by the manufacturers to the users leaving the reverse engineering of the resource on the attacker. There could be additional challenges that attacker have to resolve which could be specific to the platform and attack environment. Challenges we encountered in our projects varied from one to another which is unique to the specific platform. A brief overview of the challenges we encountered in our research:

1. Our attacks span the range of classes of GPU computing. The first attack is on an integrated CPU-GPU systems on desktop processors where the CPU and GPU are tightly integrated on the same die. This attack is the first to show how contention arises between different components in a heterogeneous system through shared microarchitectural resources, or contention over interconnect. Our second attack is on the distributed multi-GPU systems on DGX servers. In both cases we had to reverse engineer the resources on which we conducted the attack e.g. caches and memory hierarchy. GPUs are a representative of accelerators in general, and we hope

2. Previously demonstrated microarchitectural attacks were demonstrated on a single component like either CPU or GPUs. However, in both of our work we performed a cross device attack (CPU to GPU and multi-GPU). This imposes an unique challenge to the attacker as the view of the system is not uniform. As our attacks are primarily memory based, we need to understand the memory hierarchy, data path and the caching mechanism of the system.

3. A successful attack not only depends on the understanding of the microarchitecture but also depends on several other factors like the availability of tools in the software level (e.g. timer in cache based timing attacks), system level support (e.g. huge pages for virtual indexing of caches). In our research we have encountered challenges where we don't have the necessary tools or the support to conduct the attack. This require to come up with unique solutions which though specific to the device but can be

generalized in other attacks in similar scenario as well.

1.3 Contribution

To the best our knowledge, **this dissertation is the first to do an in-depth study and demonstrate microarchitectural attacks on modern heterogeneous computing systems both at the scale of consumer devices and high performance computing systems**. In particular, we demonstrate microarchitectural attacks on both heterogeneous CPU-GPU systems and distributed GPU systems that lies diametrically opposite in the spectrum of computing platforms. Specifically our contributions are as follows:

1. **Developed a Prime+Probe based cache timing attacks on a integrated CPU-GPU system..** Integrated CPU-GPU systems shares several components among which Last Level Cache (LLC) for caching their data. We demonstrated both covert and side channel attack on the shared LLC. In our covert channel attack, we have placed the trojan on GPU and spy on CPU. The trojan and spy communicated with each other over LLC cache set. To conduct this attack we have reverse engineer several architectural components that was not documented. We also conducted the attack in the reverse direction as well where the trojan is located on the CPU and the spy on the GPU. In the side channel, we have our spy on the GPU and the victim is located on CPU. The malicious process located on GPU monitors the last level cache activities by measuring the cache hits and misses.

2. **Developed a contention based bus attacks on a integrated CPU-GPU sys-**

tem.. Integrated GPU is connected with the rest of the system through a 32 Byte bi-directional data bus called the Ring Bus. The ring bus is used both by the CPU and the GPUs. We used the ring bus to develop a covert channel across the CPU and GPU systems. Leveraging the GPU parallelism to balance the frequency between the components is required to develop a successful attack.

3. **Developed a Prime+Probe based cache timing attacks on a distributed GPU system**. DGX is a distributed GPU system where server grade GPUs are connected to each other with Nvidia proprietary links. Based on the DGX versions the internal architecture of the DGX box changes. Depending on the link topology, the GPUs in the DGX machine can access each other's memories and the data gets cached in the remote GPU only. For example, if a kernel launched on GPU A allocate their memory on GPU B, data gets cached on GPU B only. We leveraged the remote caching mechanism to conduct a Prime+Probe based cache timing attack on the remote GPU. We demonstrated both covert and side channel attack. In the covert channel attack we located trojan on one GPU and spy on the other. Both spy and trojan allocate their memory on the same GPU and communicate covertly over the cache sets. Our attack is completely conducted from the user level with no system level assistance like huge pages. We have to completely reverse engineer the cache architecture from the user level to conduct the attack. We used this knowledge to develop the side channel attack. Our side channel attack monitors the cache sets and get the memory footprint of the high performance applications. Then we used machine learning techniques to classify the application type. Our side channel attack

allows the attacker to determine the application to launch their attack on the intended GPU.

1.4 Thesis Organization

The dissertation is organized into the following chapters. Before going into the details of our, we provide a detailed background of our attack environment in chapter 2, overviewing both integrated CPU-GPU and distributed multi-GPU environment. This chapter also provides an overview of the cache Prime+Probe based cache timing attack which would be convenient in understanding the following work chapters. Chapter 3 presents the Prime+Probe timing attack on integrated CPU-GPU environment. First we provide our threat model and assumption and then provide the detail of our attacks. We provide a detailed discussion on our challenges encountered while developing and conducting the attack.

Chapter 4 presents a contention based attack on the ring bus that connects the CPU and GPU with the rest of the system. It achieves high bandwidth communication with low error rate. The chapter provides the challenges and details of algorithm used to conduct the attack.

In Chapter 5 we detail of our distributed GPU system attack on the DGX boxes. The chapter overviews the attack surface as well as the challenges we encountered and our methodology to overcome the challenges. As the attacks have been conducted from user level, the challenges we encountered differ significantly from previous attacks which benefit from system support. We demonstrate a channel on remote GPU caches achieving high bandwidth and low error rate channel. We also build a side channel attack which allows the attacker to fingerprint the application running on the GPU. This in turn allows the attacker to find the target GPU in the multi-GPU environment. Finally, Chapter 7 presents some concluding remarks and future directions enabled by the research.

Chapter 2

Background

Modern computing platforms often consist of different computing components appropriate for varied applications, interconnected using custom networks and complex memory hierarchies. Graphics Processing Units *(GPUs)* are one of the most widely-used accelerators present in almost all the modern computing systems. Although initially developed for the graphics rendering purposes, currently GPUs are used for general purpose computing as well. Manufacturers provide the drivers and a programming framework to program the GPU devices. In this chapter, we present an overview of GPU based systems used at different scales. The chapter also introduces the principles of cache based prime and probe attacks, which is a covert and side channel attack technique that we leverage in several of the attacks explored in this dissertation [56, 85].

GPUs have a different computational model, exploiting massive multi-threading and Single-Instruction Multiple Data (SIMD) style parallel processing. GPUs are programmed using custom programming languages and libraries designed to support their computational model. For example, CUDA [76] is the programming framework used on Nvidia GPUs. Similarly, OpenCL [46], another programming framework used to program all the GPUs.

The GPU driver is responsible to get all the commands from the user and submit these commands to the GPU. Upon receiving a command, the GPU starts execution as an offload device requiring system assistance for operation. In contrast to CPUs, GPUs are throughput based machines suitable for data-intensive regular applications. GPUs are capable to launching large number of threads that could operate on individual data points independently making GPUs suitable for parallel programming. Programmers identify the sections of their application that are suitable for the GPU and port their implementation. GPUs are currently available in different forms. But they could be classified into two major categories, discreet and integrated. Discreet GPUs comes as a separate device and gets connected to system through peripheral connectors like PCIe bus. Integrated GPUs are smaller version of the discreet GPUs and are integrated with the CPU on the same die. The integrated GPUs though computationally weaker compared to their discreet counterpart, there are several advantages like power saving, lesser access latency. The following sections would provide a detail about the discrete and integrated GPU architectures.

2.1 Discrete GPU architecture

Traditionally, GPUs were only available as a separate device that connects with the rest of the system through PCIe. Figure 2.1 shows the overview of the while Nvidia GPU system architecture. For our research, we used a server grade pascal P100 GPUs. Though the architectural specification changed over time, the basic GPU architecture remained same. GPUs consists of simple single precision processing units called Scalar Processors (SP) which is grouped together in a single Streaming Multiprocessor (SM). The number of SMs and SPs in SMs varies over the GPU generations. GPU also consists of double precision for double precision operations and also special functional units responsible for carrying special functions like the trigonometric functions. Discrete GPUs comes with it's own RAM which is also known as the Global Memory not present on the same die as the GPU chip. The global memory is connected to the chip through bus. The size, architecture and the access speed of the global memory improved over time. Our GPUs comes with 3D stacked memory [72] with a 16 GB space and High Bandwidth Memory (HBM) version 2. GPU chip is connected with the RAM with a 512B interface. Generally, programmers first transfers data from the host memory to the device's global memory for GPU to operate. GPU also consists of various types of memories in the form of caches fit for various purposes. Constant memory is a cache that holds the constant data doesn't change throughout the kernel execution. Another cache is the shared memory which is a programmable scratchpad memory. Shared memory are present within each SMs and are shared by the threads within a thread block. Due to high RAM access latency, data points that gets reused are brought to the shared memory from the RAM and gets reused. There are also traditional caches like the L1 and L2 cache. The L1 cache is present within each SM and usually shared with the shared memory. The programmer can configure the storage space between L1 and shared memory based on their requirement. However, the L2 cache is shared between all the threads launched in the GPU. Our L2 cache is of 4 MB and caches both instruction and data. GPUs also consists of additional fast storage in the form of registers. However, the thread executes in a group in a lock step manner known as *warp* operating on the same instructions. However, separate warps can operate on different instructions.

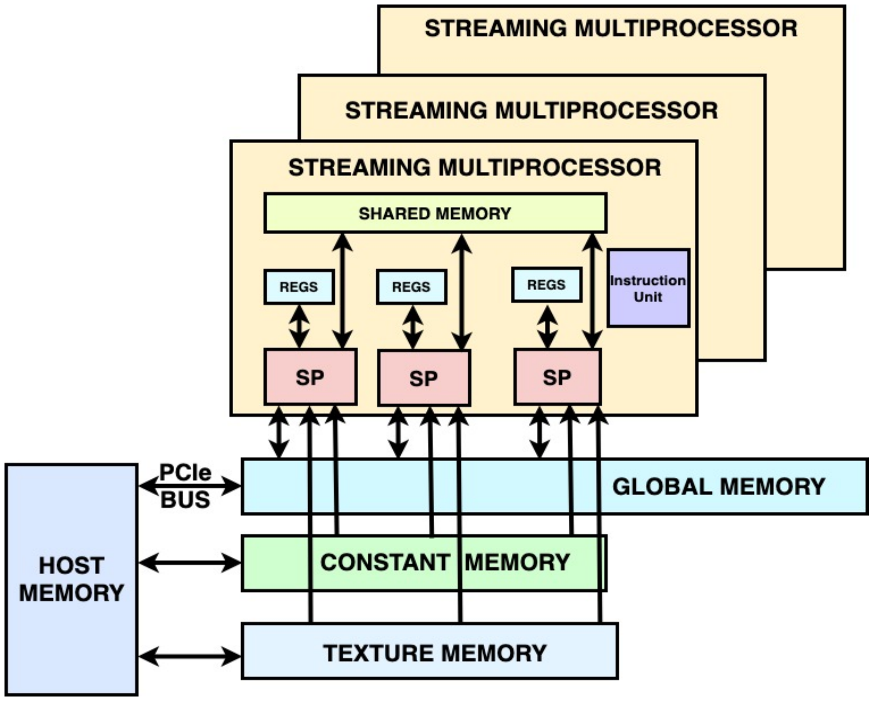


Figure 2.1: Nvidia GPU architecture

2.1.1 Nvidia Multi-GPU DGX box

We have experienced a breakthrough and expansion in the deep learning domain in the last decade [20]. However, training the deep learning models is a compute and memory intensive job. CPUs are latency based devices which require long execution time to train the deep learning models. GPU,on the other hand, is a throughput based device which is applicable for the deep learning purposes. However, the computational and memory requirement to train a deep learning model often exceeds a single GPU. Multiple GPUs could be used to train such models. However, traditionally GPUs are connected to the system with only PCIe bus. Nvidia introduced scalable link interface *SLI* [74] to use multiple GPUs for frame rendering. But it was limited in it's use and also have a low bandwidth. To accelerate the training time, Nvidia introduced AI boxes, called DGX, tuned for training the deep learning models and significantly decreasing the training time. Figure 2.2 shows the DGX-1 architecture with 8 pascal P100 [72] GPUs connected in a hypercube fashion. Pascal P100 GPU have similar architecture as mentioned in the previous section. Each GPU is connected with 4 other GPUs, 3 from the same plane and another GPU directly across from another plane. For example, *GPU 0* is connected with *GPU 1*- *GPU 3* on the same plane and *GPU 4* from another plane. The GPUs are connected with a high bandwidth Nvidia proprietary link called NVLink [73]. 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 via NVLink can access each other's global memory by using Nvidia provided APIs. Nvidia driver throws an error if the GPUs are not connected with via Nvlink. All the GPUs are connected with the rest of the system by using PCIe switches. DGX-1 comes with dual 20 core intel xeon processor that are connected to each other using intel Quick Path Interconnected *(QPI)*.

The link's topology and the GPUs within the DGX servers changed over the next generation. After DGX-1, Nvidia released DGX-2 with higher NVLink bandiwtdh and DGX-V with volta V100 GPUs. DGX-V uses NVSwitch [71] which have higher bandwidth compared to NVLink. Unlike NVLink in DGX-1, nvswitch link topology contains a switch-

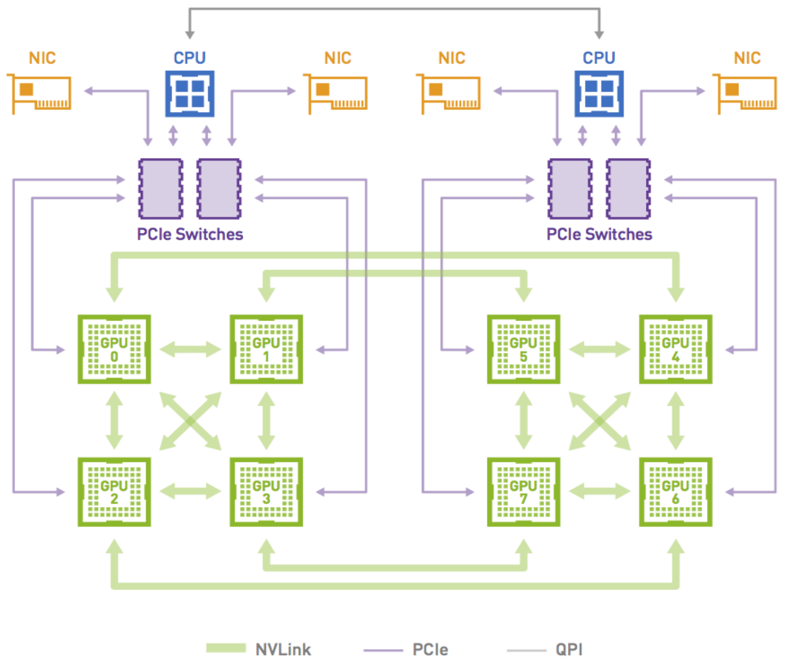


Figure 2.2: Nvidia GPU architecture

ing hub that allows all the GPUs to access other GPUs using same NVidia provided APIs. DGX-A is the latest DGX server that nvidia released that is equipped with NVidia A100 GPUs. For our work, we used DGX-1 for all our experimentation purposes.

2.2 Integrated GPUs

Currently, GPUs are available in the integrated form as well, where the GPU has been integrated on the same die as the CPU as shown in Figure 2.3. Integrated GPUs *iGPUs* are smaller version of the discrete GPUs, with lesser computational capability compared to their discrete counterpart. However, iGPUs have their advantages like lower power requirement. iGPUs have lower access latency to the memory as they are tied with the CPU on the same die and not via PCIe. The iGPUs share the system RAM for its main storage purposes. Typically it is limited to 4 GB by the driver. Though iGPUs are primarily used for system graphics rendering purposes but can be used for general purpose computing as well through OpenCL [46] framework. iGPU's architecture is identical to the discrete GPUs. They come with simple single precision Execution Units (*EUs*) similar to SPs of Nvidia GPUs. Typically 8 EUs are grouped together to form a Subslice, similar to SMs in the Nvidia GPUs. The number of subslices varies with the GPU generation. We used Gen9 [30] intel GPUs in our work which contains three subslices with 8 EUs per subslice giving to 24 EUs in total. iGPUs contains several cache levels like L1, L2 and L3 cache that is within the GPU chip. L1 and L2 caches are private to each subslice and are segregated into data and instruction caches. L1 and L2 cache is used exclusively for the graphics rendering purposes. However, the L3 cache is unified and used for caching both instructions and data. L3 is used both for graphics and general purpose computing as well. iGPUs also consists of user controlled scratchpad cache called local memory per subslice similar to shared memory in the discrete GPUs. iGPUs also have other faster storages like constant and texture cache and huge number of register files. Similar to the discrete counterpart, constant cache is used for storing constant data and texture cache for graphics. Registers are private to each thread. In discrete GPUs users can limit the register used per thread through compilation flags. But in integrated GPUs there is no such option. The compiler decides the register pressure by the user. As mentioned earlier, the threads in GPU work in a lockstep manner where certain number of threads share the instruction counter to operate on same instruction. In case of Nvidia, these number of threads is fixed to 32. Though iGPUs have similar mechanism but the number of threads that shares the instruction counter is not fixed and depends on the register pressure. If the register pressure is high then the thread count in a warp reduces and opposite in case of low register pressure. iGPU is connected to the rest of the system through a 32 Byte bi-directional data path called Ring Bus. All the system data is accessed through this bi-directional data bus. The data bus is shared with the CPU counterpart as well. The ring bus connects the CPU cores and GPU with the last level cache and other system components like the display controller, memory controller. Such interconnect structure is scalable to future generation as well. iGPU shares the Last Level Cache (LLC) with their CPU counterpart as well. All the data and instruction first gets cached to the LLC and then gets cached to the L3 for general purpose usages. Our work involves both the LLC and L3 cache architecture.

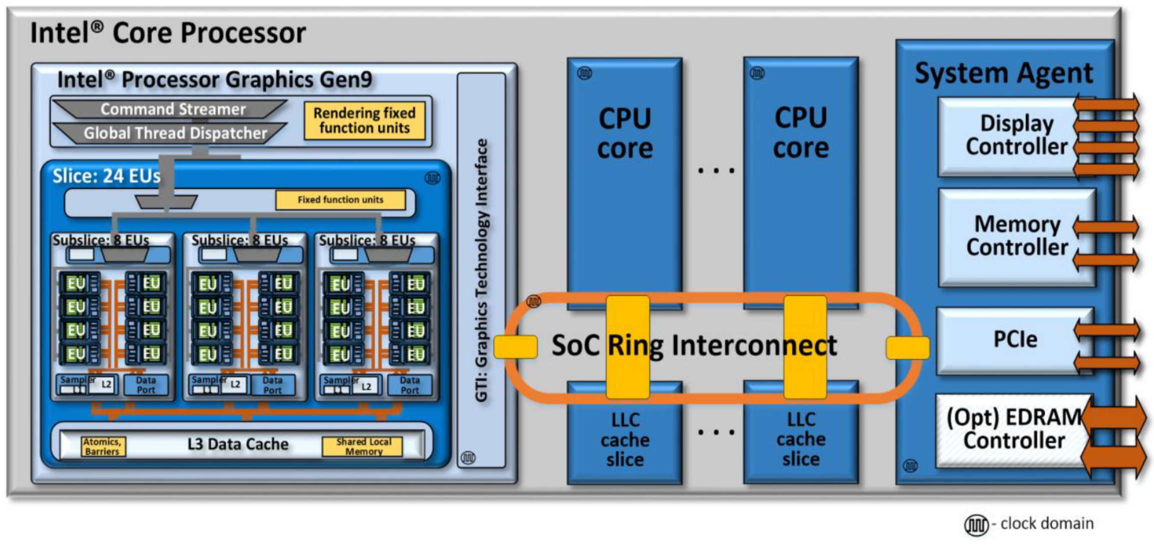


Figure 2.3: Intel CPU-GPU architecture

2.3 Timing based Prime+Probe Cache Covert and Side Channel Attacks

Microarchitectural attacks are on the rise and becoming a greater concern with the introduction of new attacks. Cache based attacks is one such microarchitectural attack that utilizes cache hits and miss timings to infer victim's information. Caches bridges the performance gap between fast CPU cores and slow RAM as caches are faster memories compared to RAM. The data when accessed are brought from RAM to cache first. The subsequent accesses occurs from the cache which is much faster than accessing RAM. The caches have limited storage capacity, much lower than the RAM storage capacity. So the data in the caches gets replaced as the new data arrives. Modern set associative caches are divided into sets and each sets have several cache lines. When new data arrives, a line in the cache set gets replaced. The cache line replacement policies changes and it is an information that the manufacturers doesn't disclose. The cache set allocation is guided by the physical address of the data. Caches have lower access time compared to RAM access and a higher access time when he tries to access a data that got replaced. The attacker uses these cache hit and miss access time difference either to communicate or infer secret information. In covert channel attacks, two attackers communicate with each other covertly over a pre-agreed cache set. An attacker, called trojan, sends sensitive data over a shared cache set to another attacker called spy. In covert channel attack, both the sender and receiver process are within the attacker's control. The attackers uses a covert method for communicating secret data as there could be no direct channel through which the attacker could communicate or the direct channel could be monitored. In side channel attack, the attacker monitor cache activities by measuring hits and misses of a victim application over cache sets. Figure 2.4 shows the basic mechanism of Prime+Probe based cache timing attack. Attacker A_1 first access the Cache Set A in step 1 and fills up the cache set with it's own data. This step of the attack is known as Prime step. In the next step 2, another attacker A_2 or a victim V access the same cache set and fills up the cache set with it's own data. In the last step 3, attacker A_1 process comes back and access the same cache set by accessing the same piece of data and measures the access time. If the data is evicted because of access by other processes, then the attacker would record a miss by having a higher access time or a hit time if the data still resides in the cache. These hit and miss times reveals the user's secrets to the attacker.

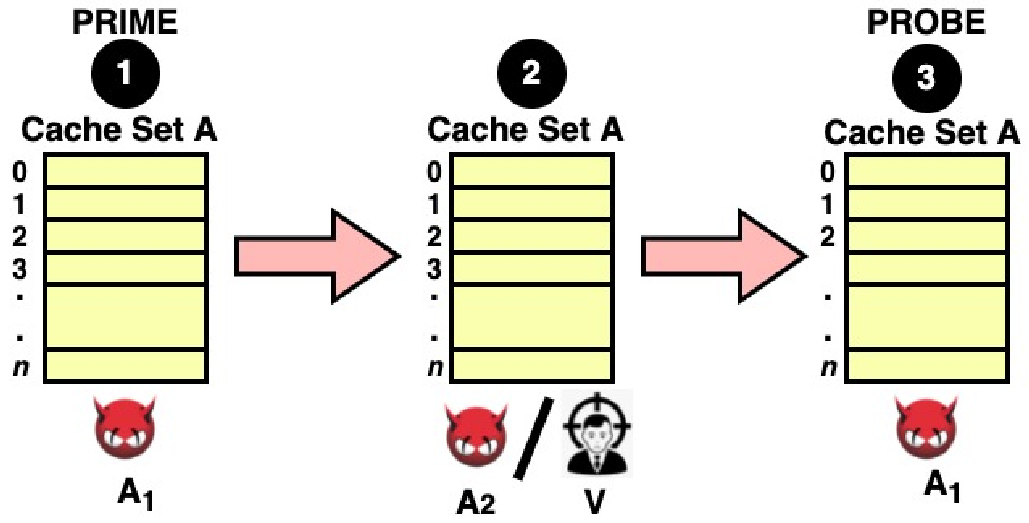


Figure 2.4: Cache PRIME+PROBE attack

To conduct a cache based Prime+Probe timing attack, the cache requires to be shared among multiple processes. Manufacturers doesn't make the cache information public and the attackers require to reverse engineer the cache architecture. Cache level and it's inclusiveness property plays an important role in executing this cache based attack as well. If the attacker uses the lowest cache level, *e.g.* Last Level Cache (LLC), as the attack surface, then the higher level caches require to be cleared for the memory load request to reach the target cache level. If the caches are not inclusive, the higher level would require an explicit eviction. However, if the caches are inclusive, then cache replacement at the lower level would also evict data from the higher level as well, relieving the attacker from explicit eviction.

Chapter 3

Heterogeneous Attacks on Shared Microarchitectural Resources: Integrated GPU-CPU cross component attacks

The goal of this work is to investigate is the feasibility of software-based microarchitectural attacks on integrated heterogeneous systems. These systems consist of multiple components with different computational models and different connectivity within the system. Our goal is to demonstrate how microarchitectural contention can be leveraged by such systems to leak information across heterogeneous components (in our case, a CPU and a GPU or vice versa). We identify that contention arises in at least two forms: on shared microarchitectural structures such as the Shared Lowest Level Cache (LLC), or contention on shared interconnect which does not hold state. In this chapter, we focus on the first form of these attacks, leveraging the shared LLC.

As the first known microarchitectural attack across heterogeneous components, we had to identify and overcome a number of challenges, that we believe will be common to this class of attacks. First, the computational model is significantly different across the two different components introducing asymmetry in how they can generate the accesses. Second, the view of the shared resource within the memory hierarchy is substantially different from the two components, which significantly complicates the attack. We also had to identify and solve other challenges that are perhaps more specific to our attack environment.

3.1 Introduction

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. Current desktop CPUs are essentially Systems-on-Chips (SoCs) as they are comprised of different computational components, CPU cores and accelerators, within the same die aimed towards accelerating varied workloads. One of the most frequent accelerator are the integrated GPUs to accelerate the graphics and screen rendering. Intel based integrated GPUs comprised of 60% of the GPU market share among the total GPU sold in 2021. Integrated GPUs are computationally less powerful to it's discrete counterpart resulting in sharing several microarchitectural components like the last level cache, bus, RAM.

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. 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. 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).

Although covert channels have been demonstrated on a variety of CPU structures, as well as on discrete GPUs [40, 41, 67, 69], 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. We demonstrate for the first time the vulnerability of these types of widely-used systems to microarchitectural attacks.

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.

In this chapter, we are going to discuss about the covert channel created due to contention through directly shared microarchitecture resources like the shared the lowest level of the cache (the LLC). We have developed a Prime+Probe covert channel attack using the shared LLC. The LLC based channel achieves a bandwidth of 120 kbps with an error rate of 2%. We also demonstrate a prime and probe side channel attack where the

GPU is able to spy on LLC accesses generated by the CPU.

In summary, the contributions of our work 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 a proof of concept prime-probe side-channel attack with GPU spying on the cache activity of the CPU.

3.2 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.

3.2.1 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

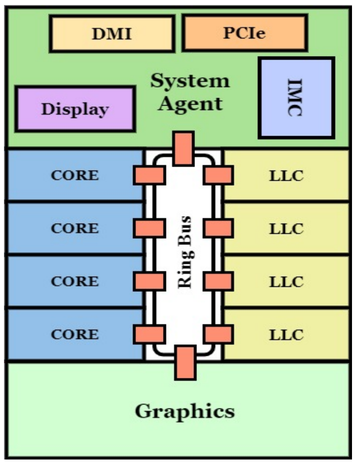


Figure 3.1: Intel SoC architecture

integrated GPUs (*iGPU*) incorporated on the same die with the CPU, to support increasingly multi-media 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 [32] 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 [46] (equivalent to CUDA programming model on Nvidia discrete GPUs [76]). 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

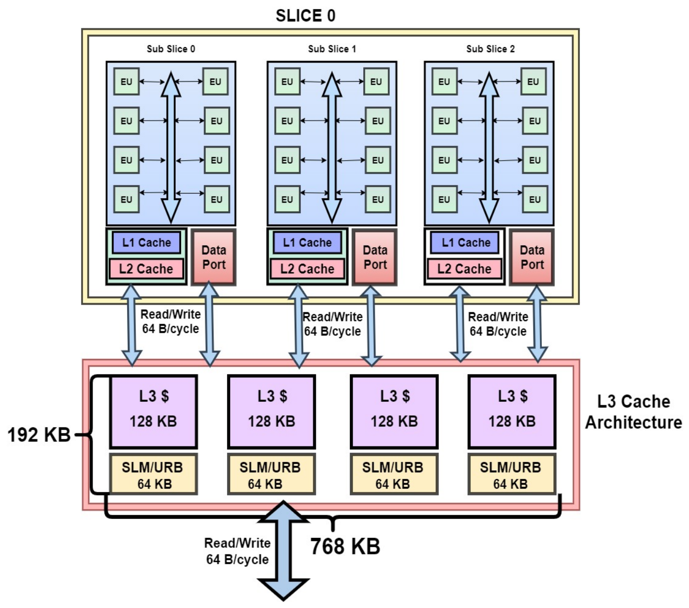


Figure 3.2: Intel integrated GPU architecture

the CPU (typically at the LLC level). Figure 3.1 shows the architecture of an Intel SoC processor, integrating four CPU cores and an iGPU [30]. 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 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 4.4. 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 3.2. 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 3.3.4. 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 [30].

3.2.2 Threat Model

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 [64] or GPU [67], 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. In this chapter, we are going to look into the LLC based covert chanel in detail.

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.

3.3 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 [85]; 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”.

3.3.1 Attack Overview and Challenges

In this attack, the CPU and GPU communicate over the LLC cache sets. Figure 3.3 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.

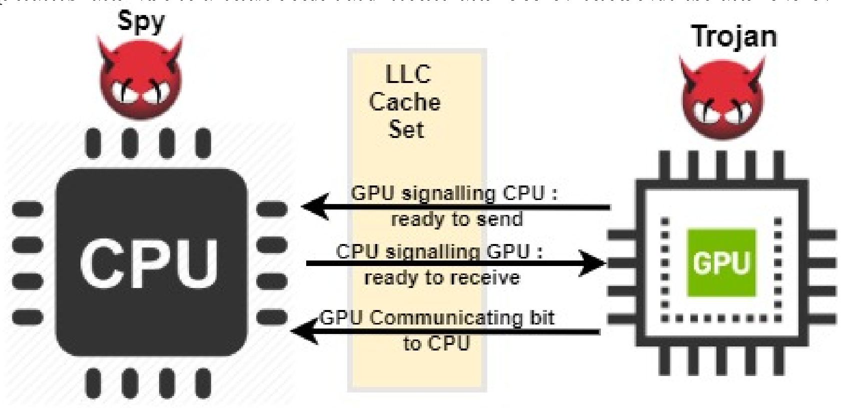

Spy
Trojan
LLC Cache Set
CPU
GPU
GPU signalling CPU : ready to send
CPU signalling GPU : ready to receive
GPU Communicating bit to CPU

Figure 3.3: 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 3.3.2.

- **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 [110] 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 3.3.4.

- **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.

3.3.2 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)* [7, 31]. 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 [53]). 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.

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 */
```

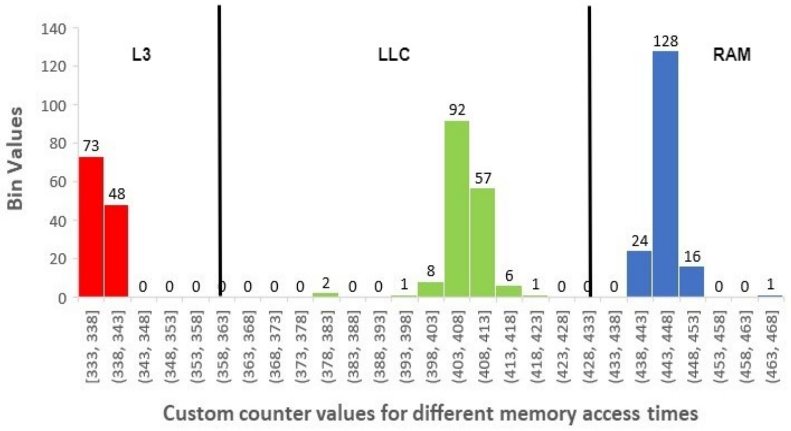


Figure 3.4: Custom Timer Characterization

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 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 3.4 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.

3.3.3 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 [96]. 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 [36, 43, 63, 110] 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.

$$\begin{split} S_0 = b_{36} \oplus b_{35} \oplus b_{33} \oplus b_{32} \oplus b_{30} \oplus b_{28} \oplus b_{27} \oplus b_{26} \oplus b_{25} \oplus b_{24} \oplus b_{22} \oplus b_{20} \oplus b_{18} \\ \oplus b_{17} \oplus b_{16} \oplus b_{14} \oplus b_{12} \oplus b_{10} \oplus b_6 \end{split} \tag{3.1}$$

$$\begin{split} S_1 = b_{37} \oplus b_{35} \oplus b_{34} \oplus b_{33} \oplus b_{31} \oplus b_{29} \oplus b_{28} \oplus b_{26} \oplus b_{24} \oplus b_{23} \oplus b_{22} \oplus b_{21} \oplus b_{20} \oplus b_{19} \\ \oplus b_{17} \oplus b_{15} \oplus b_{13} \oplus b_{11} \oplus b_7 \end{split} \tag{3.2}$$

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 [33] 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) [29] and the same physical address space through zero-copy buffers [28] 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.

3.3.4 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 [110]; in fact, relaxing inclusion has been even proposed as a defense against these attacks [44]. 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 3.2, 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 3.5 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, $p_1 \ldots p_{64}$ 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 [38]. 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) [34], 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 slice where the target LLC line resides. We accomplish this by working with the index hashing function for the LLC presented in Equations 3.1 and 3.2. 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 3.5.

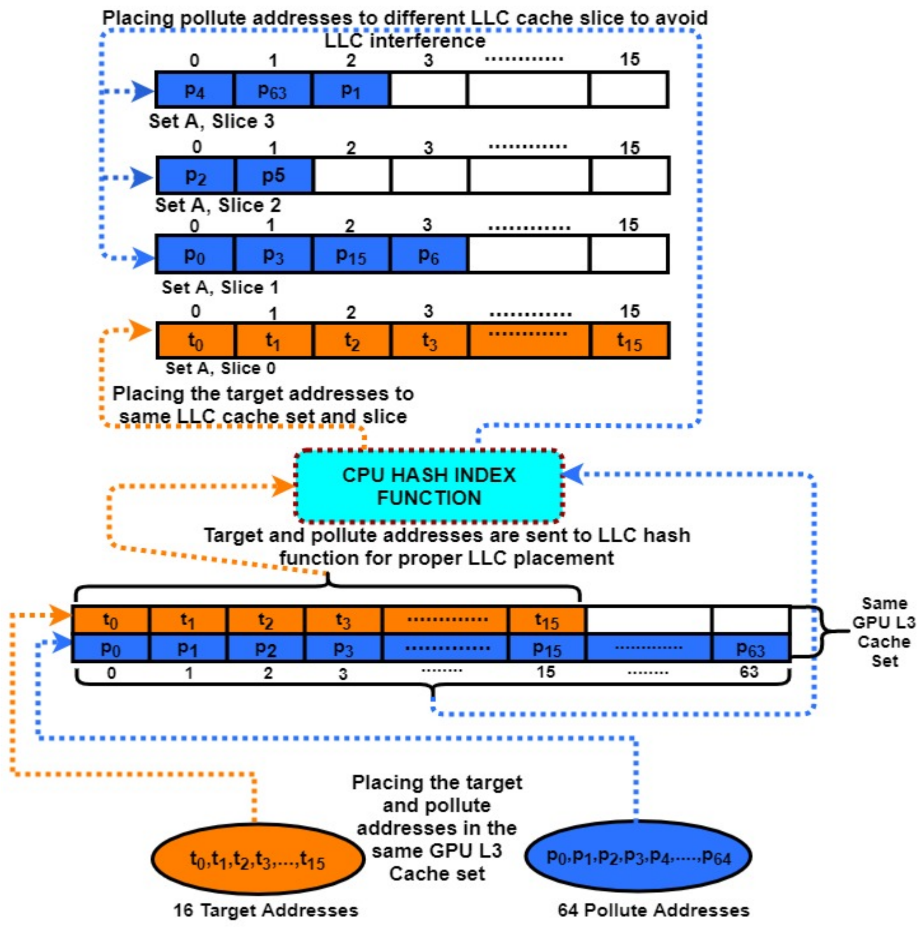

Placing pollute addresses to different LLC cache slice to avoid LLC interference
Set A, Slice 3
Set A, Slice 2
Set A, Slice 1
Set A, Slice 0
Placing the target addresses to same LLC cache set and slice
CPU HASH INDEX FUNCTION
Target and pollute addresses are sent to LLC hash function for proper LLC placement
Same GPU L3 Cache Set
Placing the target and pollute addresses in the same GPU L3 Cache set
$t_0, t_1, t_2, t_3, \ldots, t_{15}$
16 Target Addresses
$p_0, p_1, p_2, p_3, p_4, \ldots, p_{64}$
64 Pollute Addresses

Figure 3.5: 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.

3.3.5 Putting it all together–LLC Channel

As shown in Figure 3.6, 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.

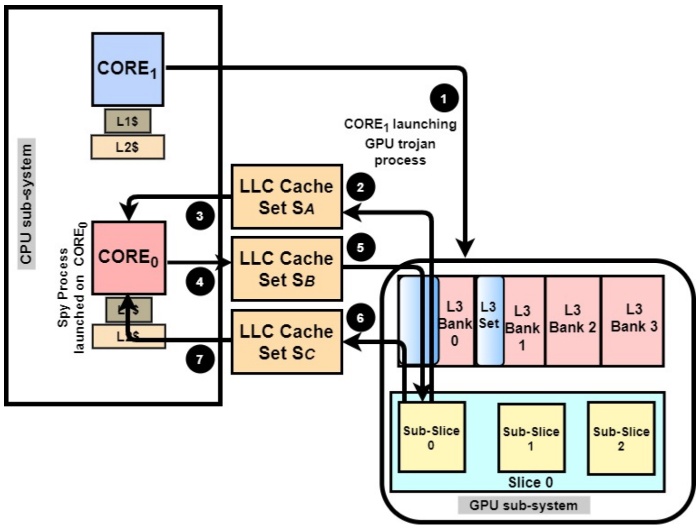


Figure 3.6: LLC based CPU-GPU covert channel details

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 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 3.3.2.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.

3.4 Evaluation

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

LLC-based Covert Channel: The GPU L3 cache is non-inclusive which requires it to

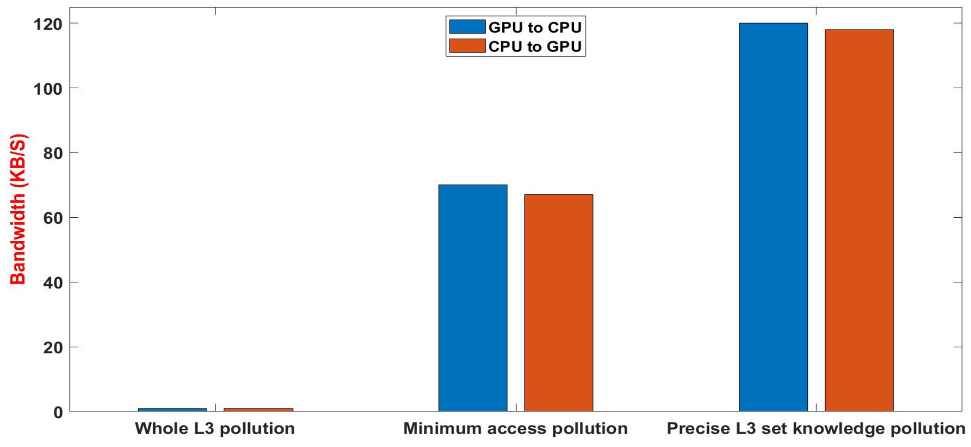


Figure 3.7: LLC bandwidth (different L3 eviction strategies)

be filled to overflow and access the LLC. Figure 3.7 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 3.7 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.

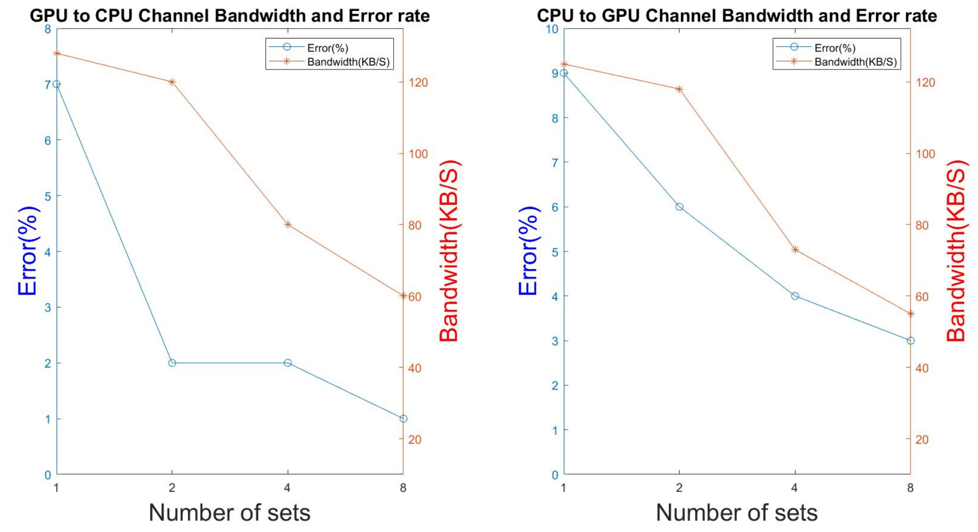


Figure 3.8: Error and BW with number of LLC sets

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 3.8 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.

3.5 Side Channel: Tracking user's cache activity

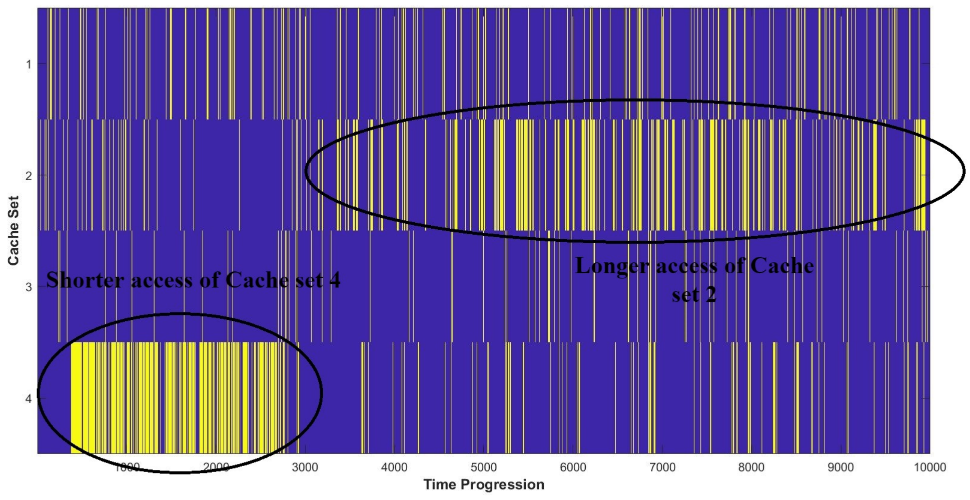


Figure 3.9: Memorygram of the cache activities of 4 cache sets

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 [81] 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 3.9 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.

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

Chapter 4

Contention based covert channel on Integrated CPU-GPU systems

Having demonstrated attacks on shared resources, we next explore whether attacks are possible using shared buses and interconnect. In these type of resources, there is not stored state who eviction we can use to detect leakage. Instead, contention can lead to slowing access through the contested resource, an effect which can be used to modulate information. Such channels are more tricky the build since the timing is critical: contention can be created/observed only when the two sides are active simultaneously.

The desktop processors comprising of the integrated GPU and CPU within a single die, are connected together with the rest of the system through a shared bus called the *ring bus*. Both the computational components uses the ring bus for all the transmission purposes. As the computational components operates independently, their simultaneous usage would give rise to an observable increased traffic in the ring bus. The remainder of the chapter discusses in detail about the detail of our threat model and the attack methodology.

4.1 Introduction

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 the chapter.

4.2 Threat Model

The threat model of our contention based attack is shown in Figure 4.1. In our contention based attack we established a covert channel attack on the shared ring interconnect. The trojan is located on GPU and spy is located on CPU. Both the trojan and spy would access their own data simultaneously. In the process, the first access would be from the main system memory. The accessed data then gets cached in all levels of caches. The attackers on both sides accesses data that is greater than the higher level caches and reaches the last level cache. We have used the reverse engineering knowledge from our cache based attack to determine the data placement in the LLC. So here used hugepages to determine the data placement in the LLC slice and set. The trojan and spy data are placed in different sets. We haven't made any special assumptions regarding the system state. The GPU runs a single applications as our integrated GPUs doesn't have support to run multiple application simultaneously. On the CPU side, we don't have any user application running and only the system processes.

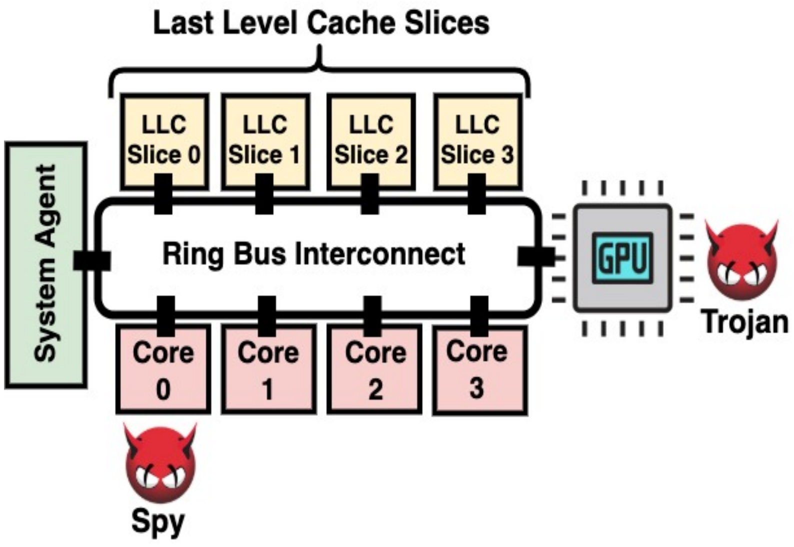


Figure 4.1: Contention channel attack threat model

4.3 Attack Details

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 4.2. 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

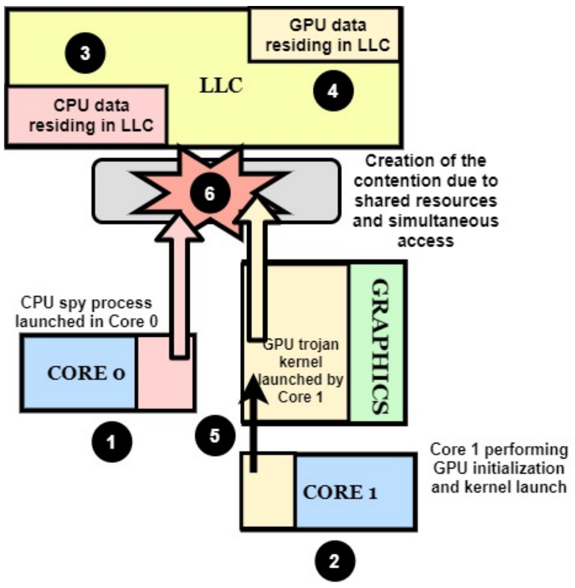

GPU data residing in LLC
3
LLC
4
CPU data residing in LLC
6
Creation of the contention due to shared resources and simultaneous access
CPU spy process launched in Core 0
CORE 0
GPU trojan kernel launched by Core 1
GRAPHICS
1
5
Core 1 performing GPU initialization and kernel launch
CORE 1
2

Figure 4.2: Contention channel attack methodology

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.

4.4 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 4.1. 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 4.2. 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 4.3. 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 4.4. 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.

$$T_{TOTAL_{CPU}} = T_{CPU} + T_{OV} \tag{4.1}$$

$$T_{OV} = f(I_F).f(S_{GPU}).f(NUM_{Threads}) \tag{4.2}$$

$$\text{s.t.} \quad S_{CPU} + S_{GPU} \ll S_{LLC} \tag{4.3}$$

$$S_{CPU} \cap S_{GPU} = \emptyset \tag{4.4}$$

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 4.5.

$$numElsPerThread = \frac{number\ of\ cache\ lines}{number\ of\ threads} \tag{4.5}$$

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 4.2. 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.

4.5 Evaluation:

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. Specifically the CPU operates at a rate of 4x compared to GPU operating frequency. To conduct the attack the CPU and GPU requires to lessen the frequency disparity. To overcome this challenge, we introduce the concept of *Iteration Factors* to match the rate of communication between the two sides. Figure 4.3 shows the optimal iteration factor:

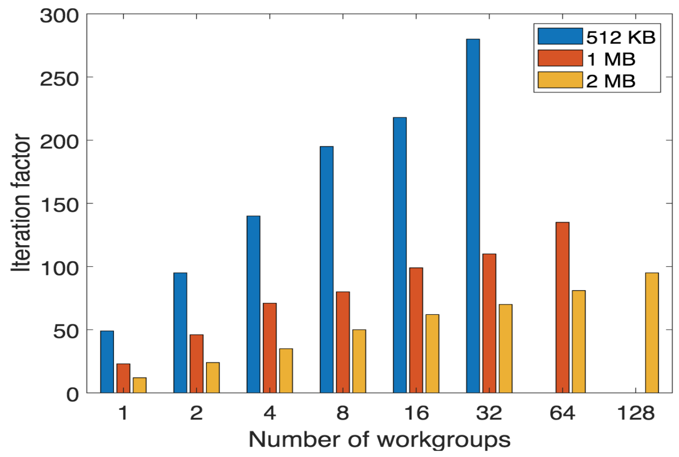


Figure 4.3: Iteration Factor for different buffer sizes

keeping the CPU buffer size constant, as the GPU buffer size increases, the factor reduces correspondingly to enable overlap between the two sides.

As discussed in Section 4.4, 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 4.4 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

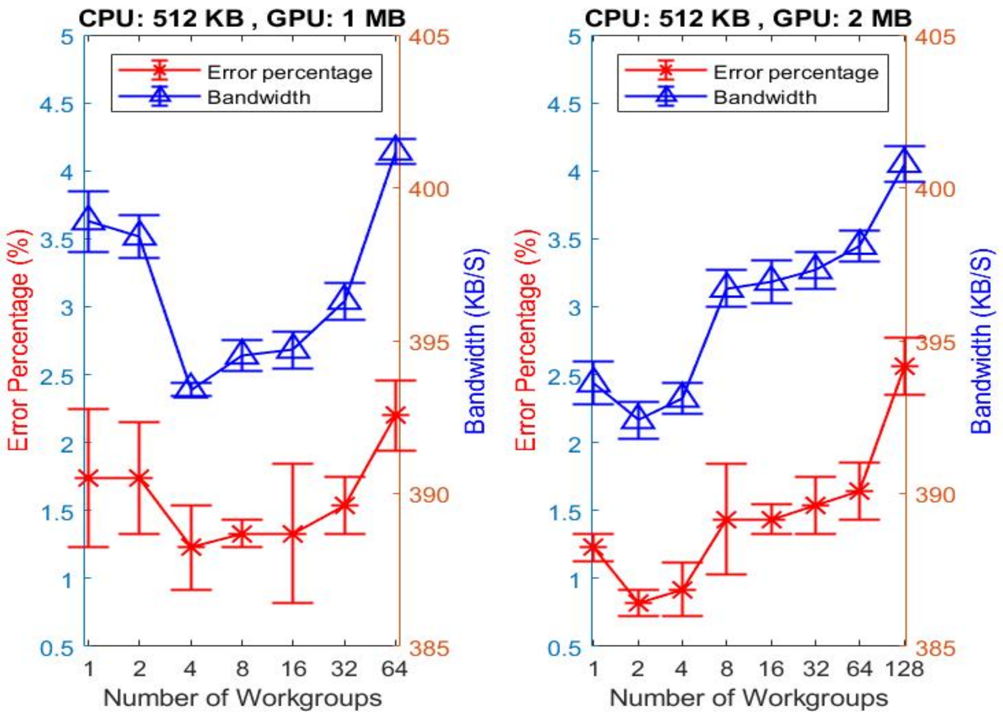


Figure 4.4: Bandwidth and error for bus-based channel

pattern of the error rate (lower bandwidth for low error rate).

Chapter 5

Covert and Side Channel Attacks on Multi-GPU Systems

GPUs have been an important computational platform enabling a variety of data intensive workloads such as deep neural networks, scientific kernels, cryptocurrency mining and many others. The size of these workloads continues to increase: for example, training 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, and are available both in cloud and high-performance computing cluster settings. For example, the Nvidia DGX series [75] offers a number of server class GPUs that are interconnected through a combination of proprietary high bandwidth interconnect (NVLink) and PCIe. DGX class machines are available on most major cloud providers. 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 chapter, we explore whether multi-GPU machines are vulnerable to both covert channel and side channel attacks. 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 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.

5.1 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.

5.1.1 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). 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.

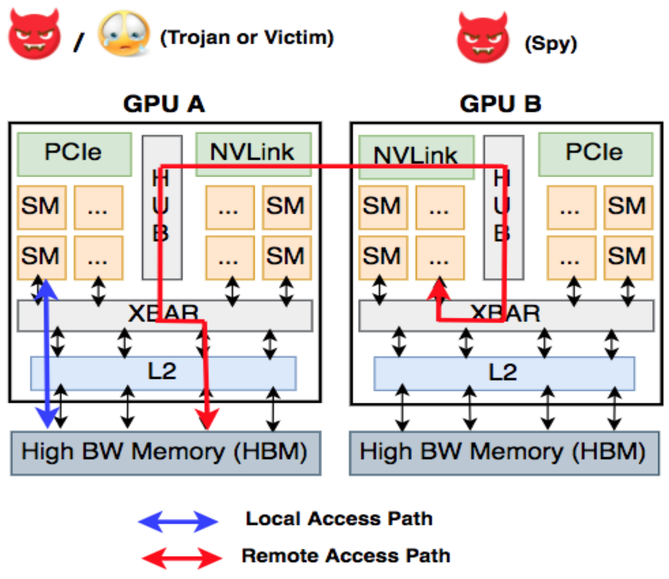


Figure 5.1: Accessing a remote GPU's memory through NVLink

If we want to find *local* L2 hit and miss time, we first specify the current device using *cudaSetDevice*. So if the local device is GPU A, then we allocate a buffer on the local device on GPU A. To measure *remote* L2 hit and miss time, we need to set the remote GPU A as the current GPU to allocate a buffer first on the remote GPU. Then the current GPU needs to be changed to GPU B and we are able to access to the remote GPU A using *cudaDeviceEnablePeerAccess*.

The local and remote GPU L2 and DRAM access time is shown in Fig 5.2 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 cache 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.

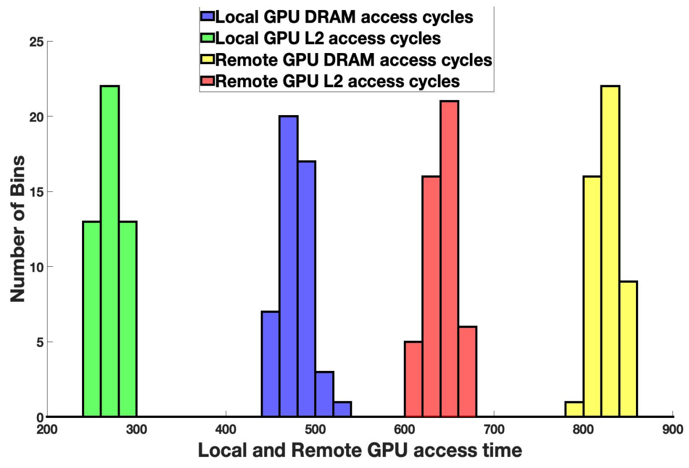


Figure 5.2: Local and remote GPU access time

Thus, we understand the memory access pathways and caching to be as shown in Figure 5.1. 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. Thus, this design enables the L2 caches to continue not to implement cache coherence because only one copy of a cache line can exist in the L2s.

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.

5.1.2 Determining Cache Eviction Sets

To conduct a successful Prime+Probe attack, an attacker needs to find a set of addresses that index into the same cache set. The number of addresses in this set should at least match the associativity of the cache, such that access to the set replaces current entries in that cache set; such a set is called *an eviction set.* Traditional Prime+Probe based attacks on CPU caches require knowledge of the cache architecture and the attacker sometime benefits from system support to facilitate the attack. For example, using huge pages allows the attacker to simplify the eviction set derivation since the page resides in contiguous physical memory, making the cache effectively partially virtually-indexed. However, our entire attack on the shared L2 cache is conducted from the virtual address space without any system level support, necessitating that we discover eviction sets individually for all sets in the cache.

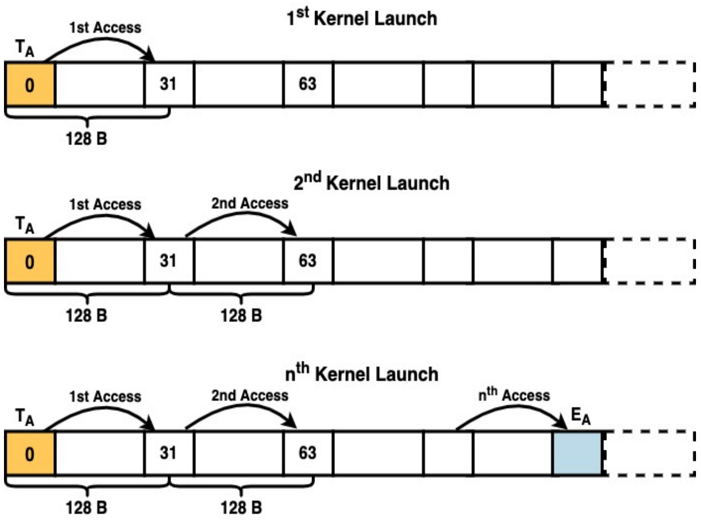


Figure 5.3: Pointer chase experiment for eviction set determination

There are some previous studies that explore the L2 cache architecture in the recent GPU architectures. Mei *et al.* [65] explored different levels of memory hierarchy in GPUs. However, their fine grained pointer chasing method was not applicable for the L2 cache as their reverse engineering process requires storing of all the timer values in shared memory which is far less than available. Our experiment, however, requires recording the access time twice which reduces the memory requirement considerably. Jia *et al.* [39] 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 non for the multi-GPU scenario). Also forming an eviction set in both the local and remote GPUs is not straightforward from the information provided in their work. Jain *et al.* [38] provided detailed information about the L2 reverse engineering as well as the architectural details. However, they modified the driver virtual to physical address translation making the address placement of the data to be consecutive in the physical address space. Of course, this property does not hold under our threat model since modifying the driver requires privileged access.

Deriving eviction sets proceeds by accesses an expanding set of addresses while timing the accesses. Once sufficient addresses are accessed that hash to the same set, we will start observing cache misses since the addresses will not all fit in the cache set. Once we have a set of addresses with a conflict set, we can test this set (removing an address at a time) to see if the misses stop (indicating that the removed address is part of the set). Specifically, we use a pointer chase experiment shown in Figure 5.3 (pseudocode shown in Algorithm 2). First, a data buffer is allocated and the target index *targetIdx* is chosen in line 1 (T_A in Fig 5.3). 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 as shown in the Figure 5.3. 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 which is shown as $\mathrm{E_A}$ in Fig 5.3. This eviction of the target address indicates that the target address $\mathrm{T_A}$ and the address $\mathrm{E_A}$ are in the same cache set. Next, to find the rest of the addresses within the same cache set, we remove the $\mathrm{E_A}$ 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 $\mathrm{T_A}$ 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.

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 GPU which the kernel is launched on. These observations allow us to simplify the attack to avoid deriving the conflict sets online 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 2 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

Algorithm 2: Eviction Set Determination Algorithm

```
basePtr = &mainBuffer[$targetIdx$];
start = clock();
nxtIdx =__ldcg(basePtr);
dummy+=nxtIdx;
end = clock();
__(*threadfence()*);
sharedTimeBuff[0] = (end-start);
*dummy Operation*
**for** $i = 0;\ i < numOfElements;\ i = i + 1$ **do**
    otherPtr = &mainBuff[nxtIdx];
    nxtIdx = __ldcg(otherPtr);
    dummy+=nxtIdx;
    __*threadfence()*;
**end**
*dummy Operation* start = clock();
nxtIdx =__ldcg(basePtr);
dummy+=nxtIdx;
end = clock();
__*threadfence()*;
sharedTimeBuff[1] = (end-start);
*dummy Operation*
```

Table 5.1: L2 cache architecture

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

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

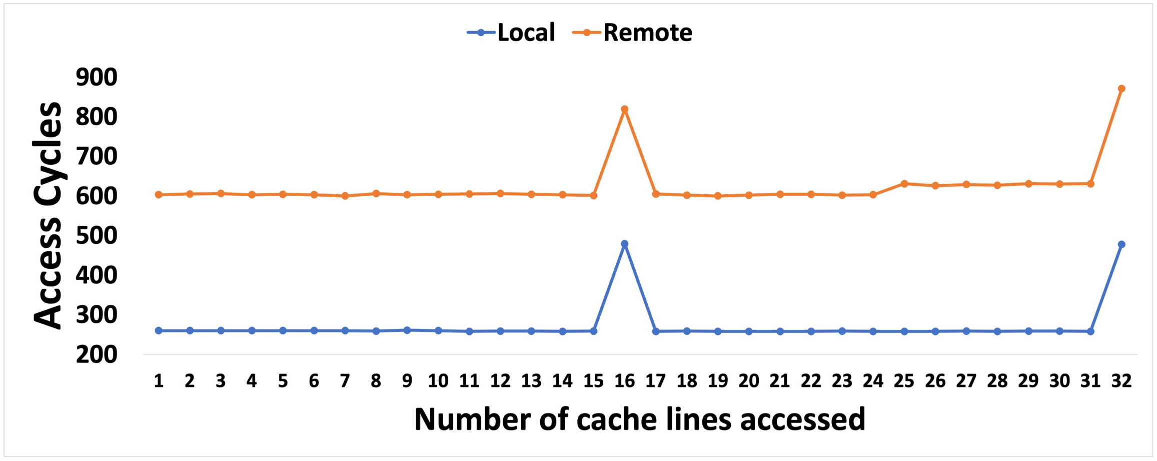


Figure 5.4: Validating the eviction set determination

Figure 5.4 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. New eviction set have been accessed after an interval of 16 addresses. It can observed that there is an eviction of the target address after every 16th access. The lower line corresponds to the local GPU cache accesses and higher one corresponds to a remote GPU cache access. This behavior confirms the LRU replacement policy with a deterministic replacement for the eviction set access.

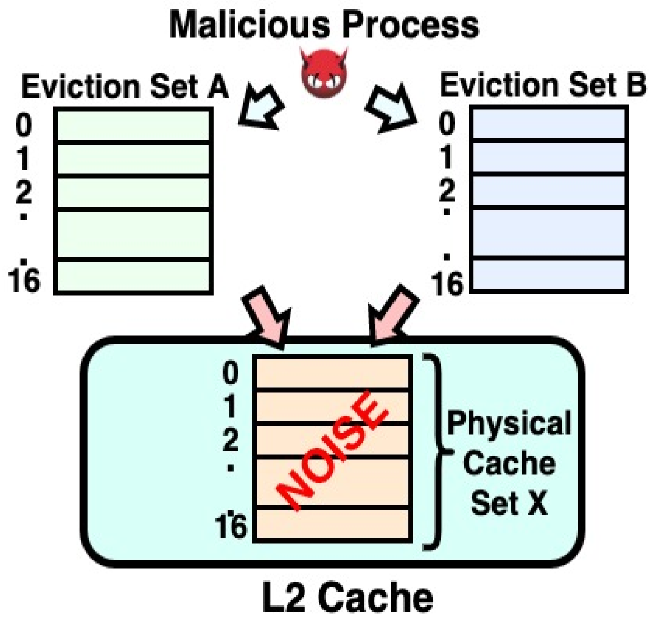


Figure 5.5: Eviction set aliasing issue

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 5.5). 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.

5.2 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 [68]).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.

5.2.1 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 to enable the processes to discover and agree on the sets to use for the signalling and communication as shown in Figure 5.6.

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 5.6, we

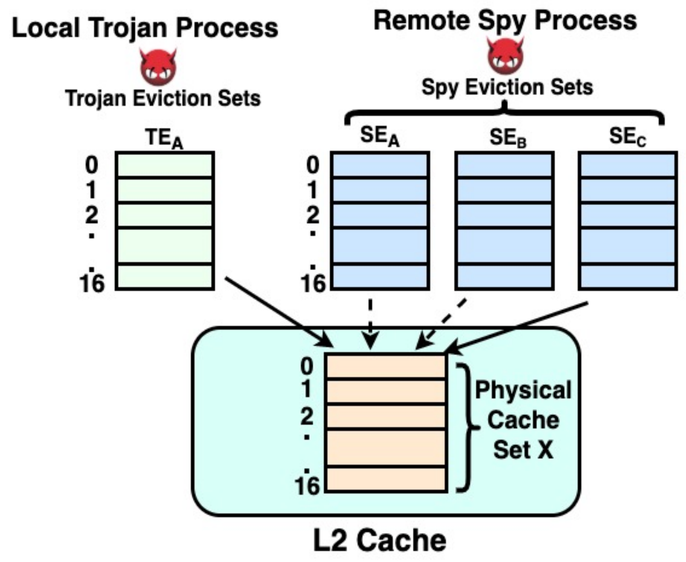


Figure 5.6: Eviction set alignment among multiple processes

can see a local trojan process eviction set $\mathrm{TE_A}$ launched on GPU A and the remote spy process launched on GPU B have three eviction sets $\mathrm{SE_A}$, $\mathrm{SE_B}$ and $\mathrm{SE_C}$. 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 $\mathrm{TE_A}$ is not mapped to the spy eviction set $\mathrm{SE_A}$ and $\mathrm{SE_B}$ shown by dotted arrows. But the trojan eviction set $\mathrm{TE_A}$ is mapped to the spy eviction set $\mathrm{SE_C}$.

The core part of the eviction set mapping kernel is shown in Algorithm 3. 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.

5.2.2 Putting it together: Covert channel attack

In this section we are going to 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

Algorithm 3: 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));
```

be serviced from the L2 cache of GPU A.

The overall flow of the covert channel attack is shown in Figure 5.7. 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 witholding contention on these sets.

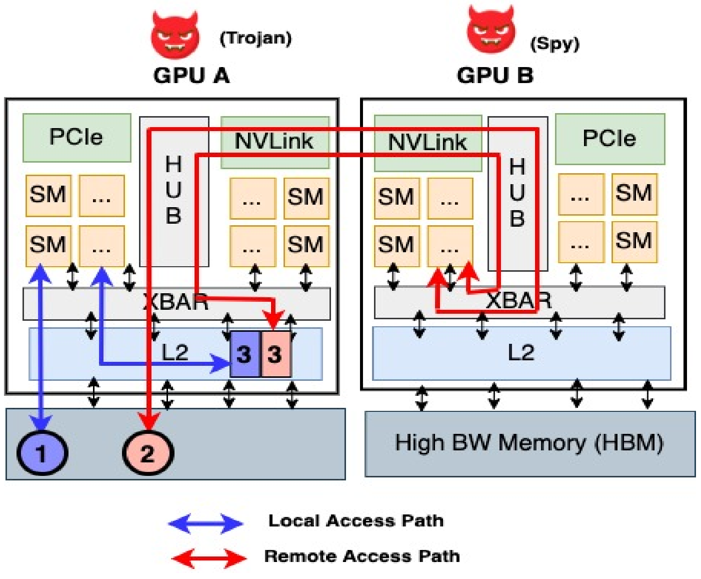


Figure 5.7: Cross GPU covert channel attack

From the previous step of cache set alignment, we have been able to determine the cache sets that are mapped among the malicious processes. This allows us to select the cache sets that would be used during the covert channel communication process. For each cache set we have allocated a thread block that would be launched to 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 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.

5.3 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. 5.9 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.

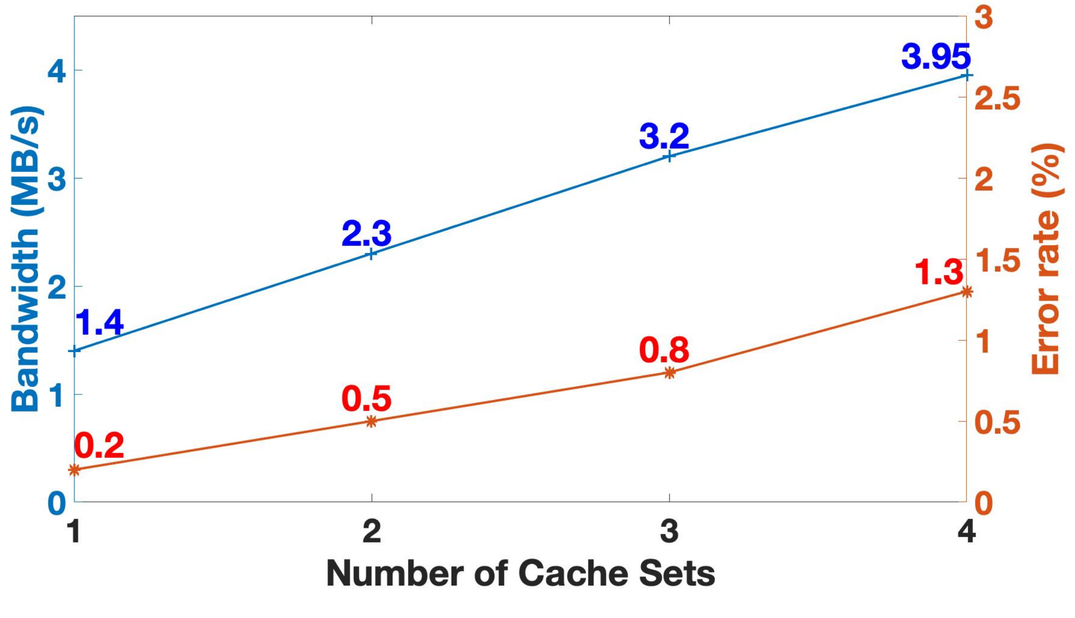


Figure 5.8: Bandwidth and Error rate in covert channel

The bandwidth and the error rate are shown in Fig. 5.8. We have measured the

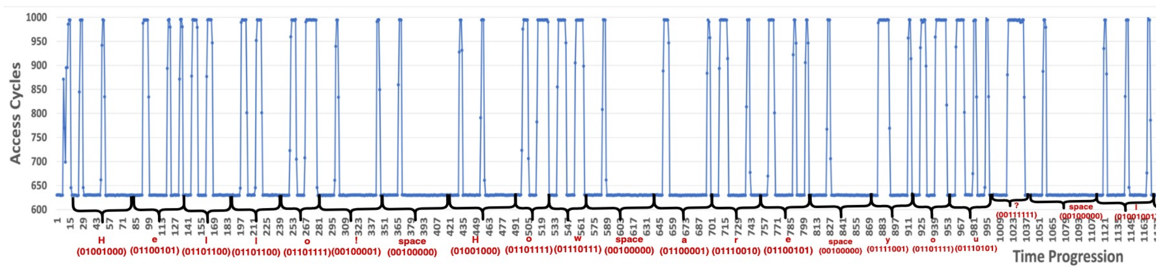


Figure 5.9: Cross GPU covert message received by spy process

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. In out covert channel, **the best bandwidth we were able to achieve is 3.2 MB/s over 3 cache sets in parallel and with an average error rate of 1% measured over 1000 runs.**

5.4 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 [89] 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. 5.1. 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 footprint of applications get recorded over a period of time which reveal the access pattern of the applications on different L2 cache sets.

We demonstrate a specific application where we finger print the remote 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. Specifically, our side channel attack could 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 [79] as our victim applications. Our application set include common HPC workloads like vectoradd, histogram, blackscholes, matrix multiplication, quasirandom and welshtrans-

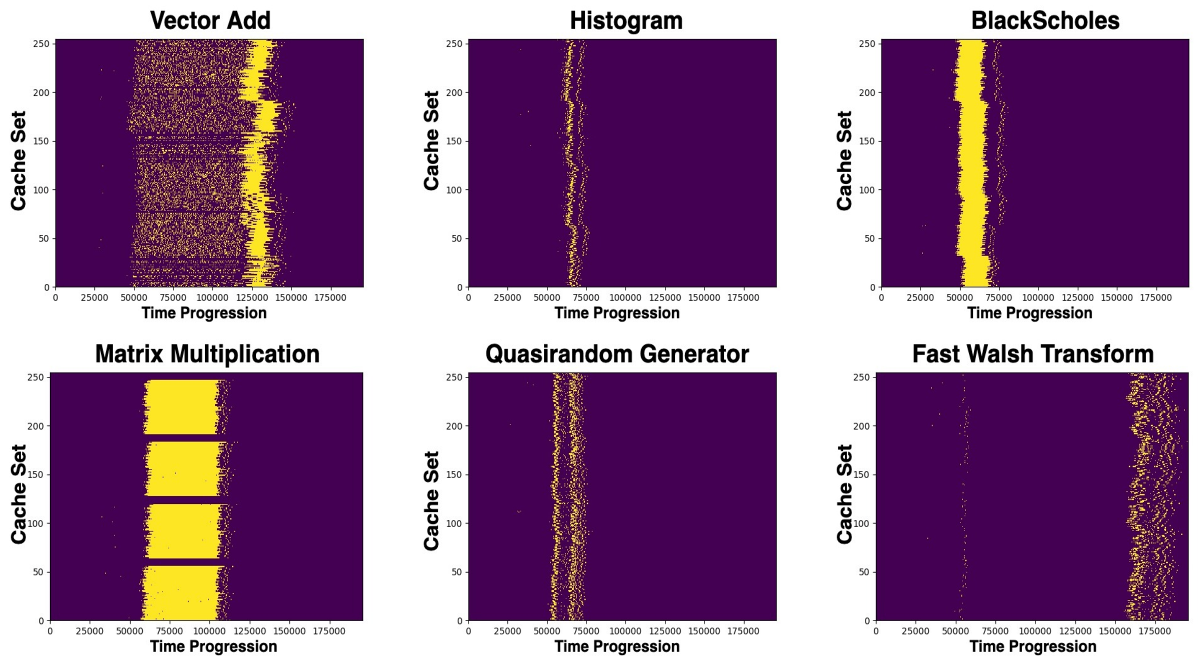


Figure 5.10: Memorygram of 6 applications

form. Example memorygrams of victim applications are shown in Fig. 5.10; 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. It is visually clear that each victim application leaves a unique memory footprint on L2 cache sets that we monitored.

In order to identify the application once we have a memorygram 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 of the six applications. 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, keeping a sufficiently large test set ensures that we evaluate the generalization capabilities with good confidence.

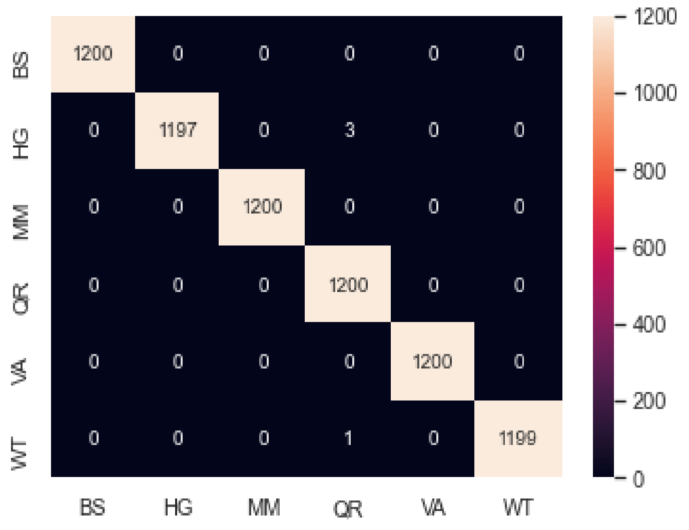


Figure 5.11: Confusion Matrix (1200 samples per class). BS (Black Scholes), HG (Histogram), MM (Matrix Multiplication), QR (Quasi Random), VA (Vector Addition), WT (Fast Walsh Transform)

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 5.11. 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.

5.5 Noise Mitigation

We developed our attacks in a quiet environment where spy and trojan are the only processes running on two GPUs. 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.

For mitigating noise, we propose to leverage concurrency limitation of GPUs using similar approaches as [68] to force exclusive execution of spy or trojan on GPUs. Based on leftover policy for GPU multiprogramming, thread blocks of 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.

In our covert channel attacks, we launch just one thread block on each SM. However, the maximum shared memory per *SM* in Pascal GPUs is twice the size of maximum shared memory per *thread block*. In particular, there is 64KB shared memory per SM, but one thread block cannot allocate all 64 KB. To consume the maximum shared memory per SM, we can launch some dummy thread blocks that are concurrent with spy's (or trojan's) thread blocks and use the leftover shared memory on each SM, but do not interfere 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 and remove the noise in the covert channel case.

5.6 Possible Mitigations

Defense mechanisms against microarchitectural covert and side channel attacks on CPUs and GPUs can potentially apply to cross-GPU attacks with some adaptations. One of the well-studied solutions is static or dynamic partitioning of shared resources [13,49,55, 100,104]. For example, Nvidia designed Multi-Instance GPU (MIG) Technology [80] 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. The current design of MIG is not applicable to protect against our cross-GPU attacks. However, we can extend this design to isolate memory accesses from local and remote processes on the single GPU's memory path.

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 [104]). 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 [47] 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.

Chapter 6

Spying on Deep Learning Framework in Multi-GPU systems

Machine learning has gained popularity in recent years. Among may machine learning algorithm, deep learning is one of the most important machine learning algorithm. Currently, deep learning is omnipotent and it's usage can be seen in numerous applications. Deep learning frameworks operates in two phases. The first phase is the learning phase where the framework learns on labelled data and optimizes the network to classify. The second phase is known as the testing phase where unlabelled data is provided to the network and based on the learning phase, the network is capable to classify the unlabelled data. The learning phase determines the effectiveness and it is a data intensive process. Traditional computational devices like CPUs take a long execution time to learn the network. However, the deep neural computations involved matrix operations that is appropriate for the GPU execution.

Recently, we have seen some active interest from the security community towards determining the machine learning model during the training phase by extracting the model parameters and hyperparameters. In [106], Yan *et al.*derived the model parameters and hyperparameters by using cache side channel in CPU. They have used both Prime+Probe and Flush+Reload to derive the ML model structure. NaghibiJouybari *et al.* [69] had similar target to extract the model parameters in discrete GPU environment. In their work, they have used the available GPU performance counters to classify operational intensity and derive the parameters. However, currently the available machine learning framework provides the means to execute the training phase over multiple GPUs. Nvidia introduced DGX boxes constituting of server grade GPUs to accelerate the machine learning model in manifold. Our endeavor is to extract the machine learning model's parameters and hyperparameters in training phase in the modern multi-GPU environment. The rest of the chapter provides the details about our attack methods and findings.

6.1 Introduction

GPUs are one of the major contributor for the advancement in the machine learning research. GPUs are currently one of major accelerators for accelerating the learning phase of the algorithm. Currently all major deep learning frameworks like tensorflow [1], pytorch [84] include the GPU support. Modern GPUs like Nvidia Titan V and Nvidia Tesla V100 runs at 47$\times$ faster for deep learning than a regular server CPU (namely an Intel Xeon E5-2690v4) [95]. Nvidia provides libraries like cuDNN to fine tune the deep neural algorithms on GPUs. Recently, Nvidia released a distributed Multi-GPU systems called DGX targeted towards reducing the training time of the deep learning algorithms over multiple GPUs. With the advancement of deep learning research, there have been an active effort from the security community to perform a deep learning model extraction. Model extraction involves determining the different parameters and hyperparameters by studying the microarchitectural resource behaviour during learning phase. Primarily we observed that the attackers utilized the performance counters to extract the model parameters. Through our work we made a primary effort to extract the model parameters in a multi-GPU environment. We made an effort to extract the model parameters of a deep learning models running on a remote GPU from another GPU. Our first approach is to extract the model parameters using cache based Prime+Probe timing attack to extract the parameters.

6.2 Cache based Prime+Probe multi-GPU model extraction attack

In our Prime+Probe based cache timing multi-GPU attack we have demonstrated a simple model extraction attack. Our threat model have been demonstrated in Fig. 6.1. The Spy is launched on GPU B while the victim is launched on GPU A. Both the processes allocated their memory on GPU A and share the L2 cache. The spy program monitors the L2 cache activity if the victim's process. Here the victim process executes a deep learning algorithm in it's training phase.

We used a multi-layer perceptron (MLP) model as the victim's process on GPU A and the spy process remotely monitors the L2 cache in GPU A from GPU B. The MLP

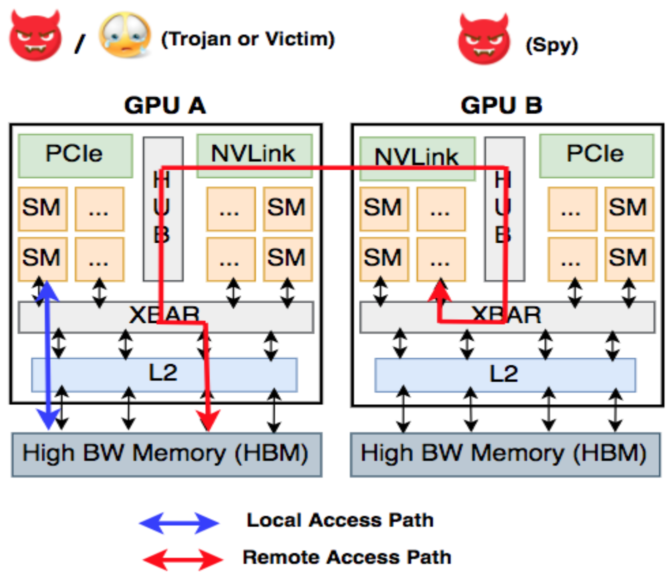


Figure 6.1: Prime+Probe based multi-GPU model extraction attack's threat model

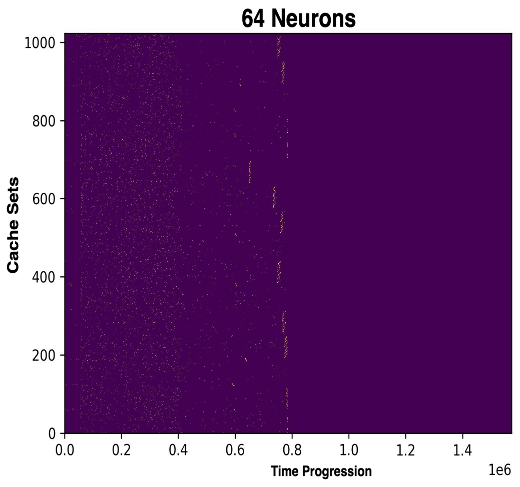
program that we used uses the MNIST dataset [12] which is a hand written digit recognition data-set. The network we used has an input layer, 1 hidden layer and a single output layer. We varied the number of neurons in the model which is one of the parameter that we are trying to extract through our attack. First we determined unique eviction sets required to monitor the cache sets using the reverse engineering in section 5.1.2. We determined 1024 unique cache sets to monitor the cache activities of the deep learning algorithms to balance sampling coverage and the speed of the attack (how often we can sample each set). The memorygram of the monitored cache sets are shown in Fig. 6.2. The X-axis is the timeline and Y-axis are the number of cache sets we monitored. The time buffer we used to record the misses is kept in the local GPU so that we do not record any spurious misses. This allowed us to only record the timing caused due the misses of our determined eviction sets. Fig. 6.2a-6.2d shows the memorygram of the MLP application over different number of

Table 6.1: Average misses over all cache sets

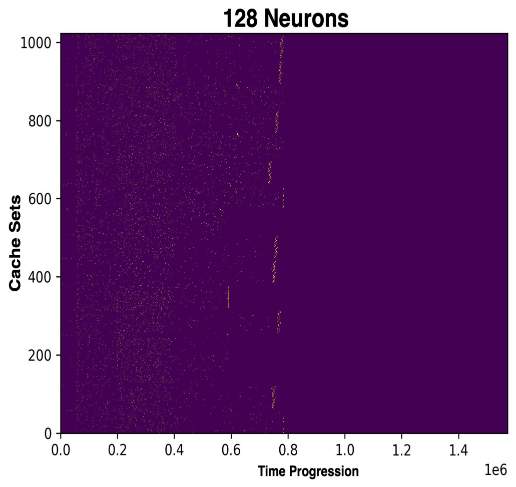
Number of Neurons	**Average Number of Misses**
64	5653
128	6846
256	8744
512	10197

neurons in the hidden layer. An increment in the number of neurons indicates an increased memory intensive operations like matrix multiplication. We observed that as the number of neurons increases the memory footprint increases which shows that there is an increment in the memory operations. To measure the misses over the measured cache sets and to differentiate among the different neuron configurations, we derived the histogram of the misses over the cache sets as shown in Fig. 6.3. The X-axis in the figure is the number of cache sets which is 1024 in our case and Y-axis are the total number of cache misses over a cache set. We have observed from the histogram images that as the number of neurons increases in the hidden layer, the cache miss increases. This allows us to identify the neuron configuration in the MLP application. Table 6.1 shows the average misses over all cache set for a particular neuron configuration. The table confirms our observation from the histogram. The average number of misses increases as the number of neuron configuration also increases.

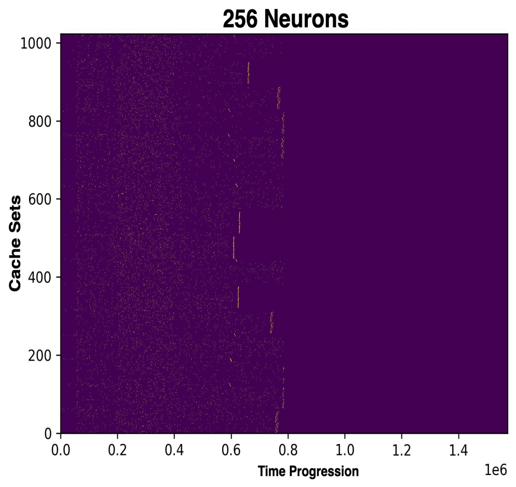
Another important hyperparameter in the machine learning algorithm is an epoch. It specifies the number of epochs or complete passes of the entire training dataset passing through the training or learning process of the algorithm. With each epoch, the dataset's



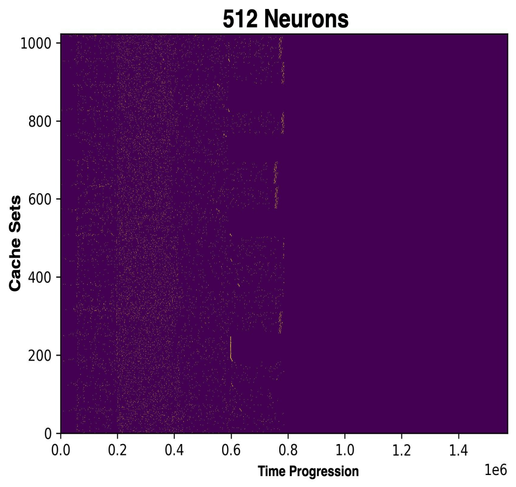
(a) Memorygram of MLP with 64 neurons



(b) Memorygram of MLP with 128 neurons

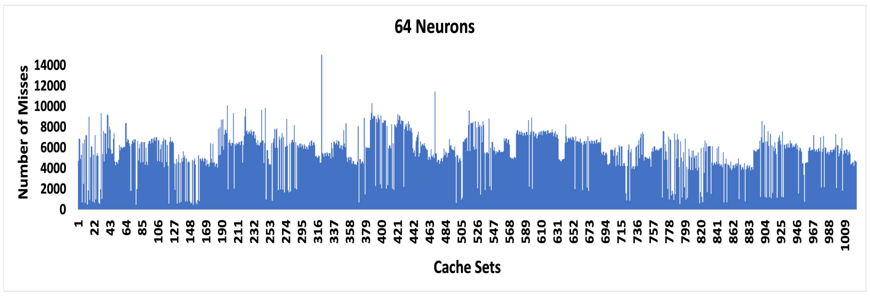


(c) Memorygram of MLP with 256 neurons

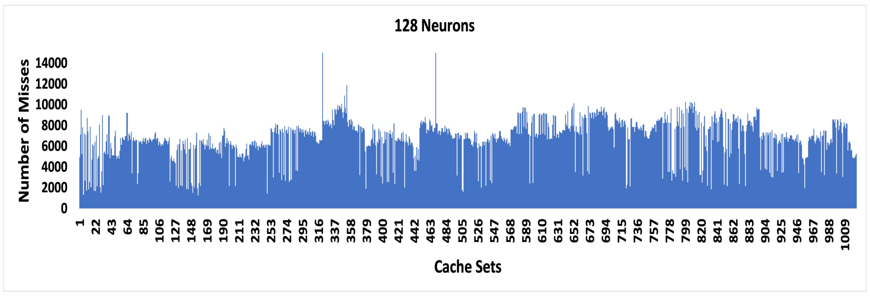


(d) Memorygram of MLP with 512 neurons

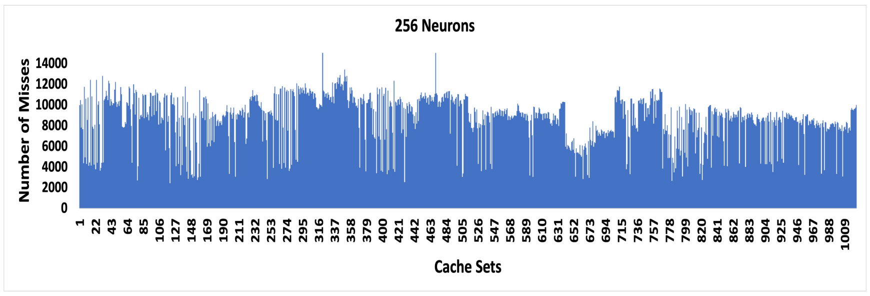
Figure 6.2: Memorygram of the MLP application



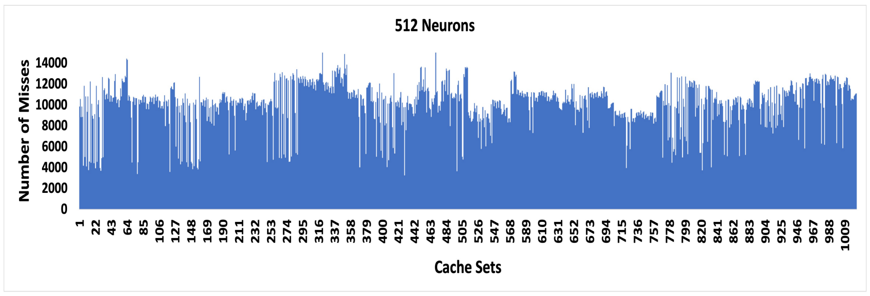
(a) Histogram of cache misses of MLP with 64 neurons



(b) Histogram of cache misses of MLP with 128 neurons



(c) Histogram of cache misses of MLP with 256 neurons



(d) Histogram of cache misses of MLP with 512 neurons

Figure 6.3: Histogram of the cache misses of the MLP application

internal model parameters are updated. Hence, a 1 batch epoch is called the batch gradient descent learning algorithm. We targeted to extract the number of epochs from the MLP through our cache monitoring. We executed the victim program over 2 epochs. We chose 128 number of neurons in the hidden layer with 2 epochs and monitored 1024 cache sets. The memorygram of the execution is shown in Fig. 6.4. We can see from the memorygram that both the epochs are distinctly visible. The first epoch extends from 1x10^5 to 4x10^5 and the second epoch from 7x10^5 to 1x10^6. Some of the cache sets in the second epoch extends beyond the end time. This is due to the fact that the starting time of the second epoch also starts late to the equivalent amount which is due to scheduling of the thread blocks responsible for monitoring that particular cache set. From our Prime+Probe based cache timing attack, we are able to determine different MLP configuration like the number of neurons and the epoch numbers.

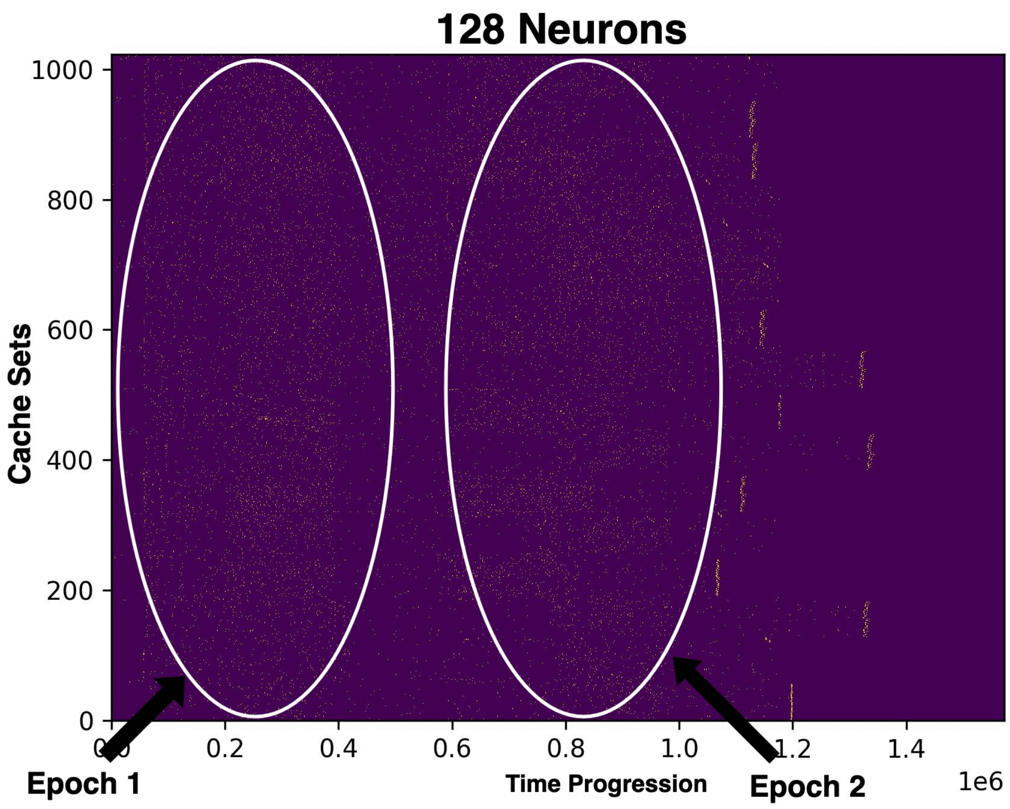


Figure 6.4: Memorygram for a two-epoch experiment

Chapter 7

Concluding Remarks

Through our current work we have been able to demonstrate microarchitectural covert and side channel attack in extremes of GPU architectures. We have demonstrated possible attacks in both integrated and distributed GPU architectures. However, through our research we identified the potential research that we can conduct to extend our understanding about the possibility of further microarchitectural attacks. Our immediate future research endeavor involves detailed study of the probable vulnerabilities in the domain of distributed GPU DGX boxes. Our entire research on the distributed GPUs were carried out in the DGX-1 box which consists of P100 GPUs with proprietary NVLinks. Also we only considered peer-to-peer GPU access within our threat model. In our future research our endeavor would involve expansion of our current understanding and further exploring possible vulnerabilities of the distributed GPU systems.

Our current threat model on the distributed GPUs is only capable of performing a peer-to-peer GPU attack. There are 8 P100 GPUs in DGX-1 that is arranged in a hypercube fashion connected with each other via NVLink. GPUs in one corner is connected to other 3 GPUs on the same plane and another GPU directly across in another plane. The GPUs that are connected via NVLink can access each other's memory directly via Nvidia provided APIs. However, this model is limited to DGX based environment with just a single hop. In our future endeavor we would like to extend our work to a multi-hop GPUs that are not connected via this proprietary links. This would allow us to generalize our current attack model. In our current study, we only vanilla cuda memory allocation policy (*cudaMalloc*). We would also like to study other cuda allocation methods like Unified Virtual Memory (*UVM*) and unified memory. Nvidia gives a unified continuous virtual address space across all host and the cuda device memory present in the system. This would allow us to look into the vulnerabilities that are not connected via the links and connected via PCIe hub which is much widely available.

This distributed GPU servers are targeted towards accelerating the machine learning training phase. Though through our current research we are able to determine certain parameters and hyperparameters, there are missing information (like batch size, number of hidden layers, weights) that is an impediment towards our understanding of the whole network design of the victim process. We would like to explore the possibility of whole model extraction by using the available performance counters that are global across the devices. These performance counters would be related to resources like the L2 cache, RAM and the links. Currently, our threat models are implemented on DGX-1. However, the succeeding DGX boxes like DGX-V uses volta V100 GPUs. The V100 GPUs contain tensor cores within the GPU SM that are intended for accelerating the tensor operations in ML application. We are going to the leakage possibility of the tensor cores during the training phase of the machine learning applications. The DGX-V machines also have different connectivity compared to DGX-1. The GPUs in DGX-V boxes are connected with each other through NVSwitch where all the GPUs are connected with another through a switch based link. This would provide us with a different GPU link architecture that would allow us extend our current threat model.

Recently,Nvidia released ampere GPUs and DGX-A with ampere A100 GPUs. These GPU generation comes with a feature called Multi-Instance GPUs (MIG). This allows a single GPU into smaller GPUs. An application running on these smaller GPU instances would have their own section of SM, L2 cache, cross-bar switches and RAM. So multiple applications running on separate GPU instances would have separate dedicated resources that are not shared. Microarchitectural attacks depends on the sharing of resources. As the GPU instances have separate resources, our microarchitectural attacks become difficult to conduct on the ampere A100 GPUs. Our future research endeavor also include to look into the leakage possibility from the applications executing within this smaller GPU instances. Nvidia is using the A100 GPUs in their data centers. A microarchitectural attack on these smaller GPU instances would be an effective study as the research would help us in conducting an attack in these secure partitions.

Prior microarchitectural attack research were demonstrated on a single device where the view of system is homogeneous. We are the first to demonstrate that the microarchitectural research across devices. Our research is primarily focused on GPU. In our first research, we have explored the vulnerabilities in an integrated CPU-GPU based system. We were able to conduct a covert communication across malicious processes that are located across different devices. We demonstrated two different leakages involving the shared last level cache and the communication bus connecting the devices. We have also demonstrated a proof of concept side channel attack as well where the victim was placed on the CPU and got monitored by a malicious spy process located on GPU. We have also demonstrated a covert channel in the reverse direction from CPU to GPU as well. Though we demonstrated our attack on GPU based accelerators, we believe that our attack principle is valid on similar kind of architectures with other accelerators.

Our second research target explores the vulnerability of GPUs in high performance computing systems, specifically modern distributed GPU servers. We demonstrated a covert channel attack across GPUs by using the shared L2 cache. Our covert channel have high bandwidth of 3.9 MB/s with a low error rate over multiple runs. We also demonstrated a successful side channel on both high performance and machine learning applications. We were able to identify the HPC application executing by studying the memory footprint. We were also able to successfully extract machine learning parameters like the number of neurons in the hidden layer and hyperparameters like the number of epochs from our cache monitoring. Our research demonstrated the possibility of microarchitectural attacks across devices that are not within the similar environment thus extending our understanding in the modern heterogeneous and distributed computing environment.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Jaeguk Ahn, Cheolgyu Jin, Jiho Kim, Minsoo Rhu, Yunsi Fei, David Kaeli, and John 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)*, pages 332–344, 2021.

[3] Jaeguk Ahn, Jiho Kim, Hans Kasan, Leila Delshadtehrani, Wonjun Song, Ajay Joshi, and John Kim. Network-on-chip microarchitecture-based covert channel in gpus. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 565–577, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Amazon. Amazon ec2 p3 instances, 2019.

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

[6] Billy Bob Brumley and Risto M. Hakala. Cache-timing template attacks. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 667–684, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[7] Anupama Chandrasekhar, Gang Chen, Po-Yu Chen, Wei-Yu Chen, Junjie Gu, Peng Guo, Shruthi Hebbur Prasanna Kumar, Guei-Yuan Lueh, Pankaj Mistry, Wei Pan, Thomas Raoux, and Konrad Trifunovic. Igc: The open source intel graphics compiler. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, page 254–265. IEEE Press, 2019.

[8] Jie Chen and Guru Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2014.

[9] Patrick Cozzi. WebGL Overview, Khronos Group, 2018. `https://www.khronos.org/webgl/`.

[10] Erik P. DeBenedictis. It's time to redefine moore's law again. *Computer*, 50(2):72–75, 2017.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

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

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

[14] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 972–984, 2021.

[15] Dag Arne Osvik Eran Tromer and Adi Shamir. Efficient cache attacks on aes, and countermeasures. In *Journal of Cryptology*, pages 667–684, 2009.

[16] Dmitry Evtyushkin and Dmitry Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *CCS*, 2016.

[17] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization*, 13(1):10, 2016.

[18] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: Accelerating microarchitectural attacks with the gpu. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 357–372, 2018.

[19] Yiwen Gao, Hailong Zhang, Wei Cheng, Yongbin Zhou, and Yuchen Cao. Electromagnetic analysis of gpu-based aes implementation. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Thomas R Eisenbarth Gorka Irazoqui and Berk Sunar profile. Cross processor cache attacks. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 353–364, 2016.

[22] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A remote software-induced fault attack in javascript. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 300–321. Springer, 2016.

[23] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 897–912, 2015.

[24] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 897–912, Washington, D.C., August 2015. USENIX Association.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[26] Wenjian HE, Wei Zhang, Sharad Sinha, and Sanjeev Das. Igpu leak: An information leakage vulnerability on intel integrated gpu. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, page 56–61. IEEE Press, 2020.

[27] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan 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*, ASPLOS '20, page 385–399, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Intel. Getting the most from opencl™ 1.2: How to increase performance by minimizing buffer copies on intel® processor graphics, 2014.

[29] Intel. Opencl 2.0 shared virtual memory overview, 2014.

[30] Intel. Intel processor graphics gen9 architecture, 2015.

[31] Intel. Intel graphics compiler, 2019.

[32] Intel. Intel processor graphics gen11 architecture, 2019.

[33] Intel. Intel® open source hd graphics and intel iris™ plus graphics programmer's reference manual: Volume 5: Memory views, 2019.

[34] Intel. Intel® open source hd graphics and intel iris™ plus graphics programmer's reference manual: Volume 7: 3d-media-gpgpu, 2019.

[35] Gorka Irazoqui, Thomas Eisenbarth, and Berk 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*, pages 591–604. IEEE, 2015.

[36] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Systematic reverse engineering of cache slice selection in intel processors. In *2015 Euromicro Conference on Digital System Design*, pages 629–636. IEEE, 2015.

[37] Ravi Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Don Newell, Yan Solihin, Lisa Hsu, and Steve 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*, SIGMETRICS '07, page 25–36, New York, NY, USA, 2007. Association for Computing Machinery.

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

[39] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. Dissecting the nvidia volta gpu architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826*, 2018.

[40] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A complete key recovery timing attack on a gpu. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA'16, pages 394–405, Barcelona Spain, March 2016. IEEE.

[41] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A novel side-channel timing attack on gpus. In *Proceedings of the on Great Lakes Symposium on VLSI*, VLSI'17, pages 167–172, 2017.

[42] Mengxuan Wang Juan Fang and Zelin Wei. A memory scheduling strategy for eliminating memory access interference in heterogeneous system. In *The Journal of Supercomputing*, volume 76, page 3129–3154, 2020.

[43] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, New York, NY, USA, 2016. Association for Computing Machinery.

[44] Mehmet Kayaalp, Khaled N Khasawneh, Hodjat Asghari Esfeden, Jesse Elwell, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. Ric: Relaxed inclusion caches for mitigating llc side-channel attacks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.

[45] 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*, 17(2):201–204, 2018.

[46] khronos group. OpenCL Overview, Khronos Group, 2018. `https://www.khronos.org/opencl/`.

[47] Hyojong Kim, Ramyad Hadidi, Lifeng Nai, Hyesoon Kim, Nuwan Jayasena, Yasuko Eckert, Onur Kayiran, and Gabriel Loh. Coda: Enabling co-location of computation and data for multiple gpu systems. *ACM Trans. Archit. Code Optim.*, 15(3), sep 2018.

[48] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[49] Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, and Huiyang 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.

[50] Esmaeil Mohammadian Koruyeh, Khaled N Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[52] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. Armageddon: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 549–564, Austin, TX, August 2016. USENIX Association.

[54] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 973–990, 2018.

[55] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby 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.

[56] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[57] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy* [56], pages 605–622.

[58] 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)*, pages 156–161, 2019.

[59] Chao Luo, Yunsi Fei, and David Kaeli. Side-channel timing attack of rsa on a gpu. *ACM Transactions on Architecture and Code Optimization*, 16(3), August 2019.

[60] Chao Luo, Yunsi Fei, Pei Luo, Saoni Mukherjee, and David Kaeli. Side-channel power analysis of a gpu aes implementation. In *33rd IEEE International Conference on Computer Design*, ICCD'15, 2015.

[61] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *Proc. International Symposium on Computer Architecture (ISCA)*, pages 118–129, 2012.

[62] Manel Martínez-Ramón, Arjun Gupta, José Luis Rojo-Álvarez, and Christos Christodoulou. *Machine Learning Applications in Electromagnetics and Antenna Array Processing.* Artech House, 2021.

[63] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse engineering intel last-level cache complex addressing using performance counters. In *International Symposium on Recent Advances in Intrusion Detection*, pages 48–65. Springer, 2015.

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

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

[66] microsoft. Microsoft azure, 2019.

[67] Hoda Naghibijouybari, Khaled Khasawneh, and Nael Abu-Ghazaleh. Constructing and characterizing covert channels on gpus. In *Proc. International Symposium on Microarchitecture (MICRO)*, pages 354–366, 2017.

[68] Hoda Naghibijouybari, Khaled N. Khasawneh, and Nael Abu-Ghazaleh. Constructing and characterizing covert channels on gpgpus. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 354–366, 2017.

[69] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2139–2153, New York, NY, USA, 2018. Association for Computing Machinery.

[70] Ajay Nayak, Pratheek B., Vinod Ganapathy, and Arkaprava Basu. (mis)managed: A novel tlb-based covert channel on gpus. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 872–885, New York, NY, USA, 2021. Association for Computing Machinery.

[71] nvidia. Nvidia nvswitch the world's highest-bandwidth on-node switch. `https://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf`.

[72] NVIDIA. Nvidia tesla p100 the most advanced datacenter accelerator ever built featuring pascal gp100, the world's fastest gpu. `https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf`.

[73] NVIDIA. Nvidia® nvlink tm high-speed interconnect: Application performance. `http://info.nvidianews.com/rs/nvidia/images/NVIDIA%20NVLink%20High-Speed%20Interconnect%20Application%20Performance%20Brief.pdf`.

[74] nvidia. Sli best practices. `http://developer.download.nvidia.com/whitepapers/2011/SLI_Best_Practices_2011_Feb.pdf`.

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

[76] Nvidia. Cuda c++ programming guide, 2019.

[77] NVIDIA. 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`.

[78] Nvidia. Parallel thread execution isa version 6.5, 2019.

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

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

[81] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos 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*, CCS '15, page 1406–1418, New York, NY, USA, 2015. Association for Computing Machinery.

[82] Riccardo Paccagnella, Licheng Luo, and Christopher 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, August 2021.

[83] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

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

[85] Colin Percival. Cache missing for fun and profit. In *BSDCan*, 2005.

[86] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

[88] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding information leakage in the memory controller with fixed service policies. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Dec. 2015.

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

[90] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[91] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018.

[92] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 213–226, 2018.

[93] Top-500. Top-500 supercomputer list, 2018.

[94] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689, 2016.

[95] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.

[96] Pepe Vila, Boris Köpf, and José F Morales. Theory and practice of finding eviction sets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 39–54. IEEE, 2019.

[97] Xin Wang and Wei Zhang. An efficient profiling-based side-channel attack on graphics processing units. In Kim-Kwang Raymond Choo, Thomas H. Morris, and Gilbert L. Peterson, editors, *National Cyber Summit (NCS) Research Track*, pages 126–139, Cham, 2020. Springer International Publishing.

[98] Yao Wang and G. Edward Suh. Timing channel protection for a shared memory controller. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2014.

[99] Zhenghong Wang and Ruby B Lee. Covert and side channels due to processor architecture. In *Computer Security Applications Conference (ACSAC)*, 2006.

[100] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2007.

[101] Junyi Wei, Yicheng Zhangy, Zhe Zhou, Zhou Liy, and Mohammad 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.

[102] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms. In *arXiv:1912.11523*, 2020.

[103] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th {USENIX} security symposium ({USENIX} security 16)*, pages 19–35, 2016.

[104] Qiumin Xu, Hoda Naghibijouybari, Shibo Wang, Nael Abu-Ghazaleh, and Murali Annavaram. Gpuguard: Mitigating contention based side and covert channel attacks on gpus. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, pages 497–509, New York, NY, USA, 2019. ACM.

[105] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[106] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020. USENIX Association, August 2020.

[107] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2018.

[108] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 719–732, 2014.

[109] Yuval Yarom and Katrina Falkner. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, August 2014. USENIX Association.

[110] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B Lee, and Gernot Heiser. Mapping the intel last-level cache. *IACR Cryptology ePrint Archive*, 2015:905, 2015.

[111] Michael K. Reiter Yinqian Zhang, Ari Juels and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 990–1003, 2014.

[112] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018.

[113] P. Zou, A. Li, K. Barker, and R. Ge. Fingerprinting anomalous computation with rnn for gpu-accelerated hpc machines. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 253–256, 2019.