GPUGuard: Mitigating Contention Based Side and Covert Channel Attacks on GPUs

Qiumin Xu^{*†}
University of Southern California
Google Research
qiuminxu@google.com

Hoda Naghibijouybari^{*}
University of California, Riverside
hnagh001@ucr.edu

Shibo Wang
Google Research
shibow@google.com

Nael Abu-Ghazaleh
University of California, Riverside
nael@ucr.edu

Murali Annavaram
University of Southern California
annavara@usc.edu

ABSTRACT

Graphics processing units (GPUs) are moving towards supporting concurrent kernel execution where multiple kernels may be co-executed on the same GPU and even on the same streaming multiprocessor (SM) core. While concurrent kernel execution improves hardware resource utilization, it opens up vulnerabilities to covert-channel and side-channel attacks. These attacks exploit information leakage across kernels that results from contention on shared resources; they have been shown to be a dangerous threat on CPUs, and are starting to be demonstrated on GPUs. The unique micro-architectural features of GPUs, such as specialized cache structures and massive parallel thread support, create opportunities for GPU-specific channels to be formed. In this paper, we propose GPUGuard, a decision tree based detection and a hierarchical defense framework that can reliably close the covert channels. Our results show that GPUGuard can detect contention with 100% sensitivity and a small (8.5%) false positive rate. The timing channels are mitigated through Tangram, a GPU-specific contention channel elimination scheme, with only 8% to 23% overhead when there is an attack and zero performance overhead when no attacks are detected. Compared to temporal partitioning, GPUGuard is 69%-96% faster in various architectures even when active, showing that it is possible to gain substantial performance from executing concurrent kernels on a single SM while securing GPUs against these attacks.

CCS CONCEPTS

• **Computer systems organization** → **Single instruction, multiple data**; • **Security and privacy** → *Artificial immune systems; Side-channel analysis and countermeasures.*

^{*}Both authors contributed equally to this research.

[†]This work reported upon here was performed while she was a Ph.D student at University of Southern California.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6079-1/19/06...\$15.00

<https://doi.org/10.1145/3330345.3330389>

KEYWORDS

GPU, security, covert and side channel, machine learning, attack detection and mitigation

ACM Reference Format:

Qiumin Xu, Hoda Naghibijouybari, Shibo Wang, Nael Abu-Ghazaleh, and Murali Annavaram. 2019. GPUGuard: Mitigating Contention Based Side and Covert Channel Attacks on GPUs. In *2019 International Conference on Supercomputing (ICS '19)*, June 26–28, 2019, Phoenix, AZ, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3330345.3330389>

1 INTRODUCTION

Microarchitectural covert and side channels are a dangerous threat vector that allows leakage of sensitive information from otherwise secure implementations of applications and systems. Many CPU microarchitectural structures such as caches [28, 39, 53] and branch predictor units [16, 17] have been shown to be viable vectors for such attacks. Speculation based attacks such as the recent Melt-down [36] and Spectre [30] also rely critically on the presence of covert channels. These attacks continue to be a substantial threat, and an active area of research on CPUs. A number of defenses and mitigations have been investigated to protect CPU structures against these attacks [15, 27, 37, 38, 50, 65].

Recent work has started to demonstrate that GPUs are also vulnerable to microarchitectural covert and side-channel attacks. GPU manufacturers are offering increasing support for multiprogramming on GPUs to fully utilize the growing resource availability in GPU – wider data paths and more streaming multiprocessors (SMs). GPUs are now offered as a resource in cloud computing systems. A malicious VM can now spy on other applications that share a GPU (a side channel attack) or collude with another to covertly communicate sensitive information to bypass information isolation boundaries. Jiang et al. [24] have demonstrated timing attack on GPUs that relies on data dependent kernel execution time variations that occur due to memory coalescing behavior. Naghibijouybari et al. demonstrated the presence of high bandwidth covert channels [45], as well as side channels on GPUs [46].

In this work, our goal is to provide a comprehensive solution to mitigate contention based covert- and side-channel attacks between two kernels co-executing on a GPU. At the core of these attacks an adversary/spy exploits contention in hardware resources to indirectly infer information about a victim kernel in the case of side channel attacks, or a colluding trojan kernel in the case of

covert channel attacks. While solutions for such attacks in CPUs have been proposed, GPUs have a substantially different execution model with massive parallelism, internal hardware schedulers that impact colocation and contention, as well as several unique (micro)architectural structures such as constant cache, which provide a varied range of paths for contention based channel formation.

Our attack detection relies on the increased resource contention that is exhibited when a GPU is facing a covert or side-channel attack. In the context of CPUs, Hunger et al. have already shown that resource contention is one of the most quantifiable impact of an attack [20]. As such, GPUGuard non-intrusively monitors resource contention across kernels through a set of well defined features and resource utilization metrics. It then uses these features and metrics to classify kernel interaction behaviors, and to identify covert and side-channel formation. Once an attack is identified, the second component of our solution separates contending kernels into separate security domains, uniquely possible in GPUs due to the inherent spatial parallelism available, to close the identified contention channels. We use security domains at different hierarchy levels to maximize sharing (and performance) when it is safe, but to close contention based channels when there is a possibility for the existence of such a channel.

One simple solution to mitigate any information leakage through shared resources is to temporally partition the resources. But it has been shown in many recent studies that GPUs benefit greatly from fine-grain sharing, including intra-SM sharing [13, 51, 66, 67, 70]. As we show in Section 7, temporal partitioning alone results in nearly 2X performance penalty compared to our proposed scheme.

To summarize, this paper makes the following contributions:

- We propose GPUGuard a new framework for defending GPUs from contention based covert and side channels.
- We evaluate GPUGuard on 250 covert-channel attacks running on their own or in combination with benign GPU workloads. We also reimplement one of the recent GPU side channel attacks, proposed by Naghibijouybari et al. [46] and show that we are able to detect it and mitigate it. To the best of our knowledge, our mitigation framework is the first proposal for defending against contention-based covert and side channel attacks on GPUs.
- GPUGuard proposes a novel spatial partitioning of resources to guard against covert and side channels formed by contention on intra-SM shared resources, such as execution units, L1 cache, warp scheduler, and instruction fetch units. For attacks formed through L2 cache and atomic memory accesses GPUGuard proposes to fall back on temporal partitioning of resources. GPUGuard in its current form does not protect against channels formed through the following resources: shared memory, SIMT stacks, and register files.
- Our solution is able to detect these channel attacks with 93% accuracy in our benchmarks. The defense scheme reliably closes contention channels by isolating kernels through a hierarchy of resource partitioning approaches.
- Compared to an insecure baseline that simply ignores such channels, GPUGuard pays 8%-28% performance penalty *only* when actively eliminating any identified channels. On the other hand, when there are no active channels GPUGuard

has little performance overhead, as the cost of monitoring contention and classifying the kernel behaviors is negligible. Compared to a baseline that uses temporal partitioning that prohibits concurrent kernel execution to eliminate channels, GPUGuard has 1.69X-1.96X better performance.

2 WHY CONSIDER GPU CHANNELS

It is not surprising that to date most of the covert and side channel attacks have been demonstrated on CPUs. GPUs have mostly been spared from such attacks primarily because until recently they had limited (or even no) support for concurrent kernel execution which leads to resource contention. But that limitation is rapidly being relaxed. For example, AMD multiuser GPUs [3] and NVIDIA vGPUs [2] both enable up to 16 concurrent clients to share a GPU. NVIDIA GPUs support asynchronous compute to concurrently run graphics and general purpose workloads since Maxwell [1]. The Volta multi-process service [5] features spatially sharing a GPU among multiple applications. Intra-SM concurrent kernel execution [13, 51, 66, 67, 70] further enables finer-grain sharing within a single SM to improve overall GPU utilization. Furthermore, GPUs are now being deployed in a virtualized cloud environment such as the Google Cloud [18], Amazon AWS cloud [7] and Microsoft Azure [42]. Both covert [45] and side channel [46] attacks between unprivileged applications based on contention have recently been demonstrated on GPUs. In this environment it is only a matter of time before GPU execution model will be compromised through covert and side-channel exploitation. Hence, we believe it is vital to tackle this challenge now and GPUGuard serves that purpose.

2.1 Baseline GPU Architecture

Each GPU kernel contains several cooperative thread arrays (CTA) or thread blocks, with each CTA sub-divided into groups of threads called a warp in the NVIDIA terminology or a wavefront in the AMD terminology. For simplicity, we use NVIDIA terminology throughout the paper. Figure 1 shows our baseline GPU consisting of several SMs, connected to global memory partitions through interconnection networks. Each SM comprises vast set of execution resources such as ALUs, registers, caches (both specialized caches such as constant caches and texture caches and regular data caches), and shared memory. L2 cache and global memory are accessible to multiple kernels running on different SMs. Modern GPUs [5, 19] allow multiple kernels to share an SM. Hence even intra-SM resources such as L1 caches, integer and floating point execution units, special function units, load/store units, instruction fetch units, and warp schedulers are shared across kernels within each SM.

While multiple recent studies focused on the effectiveness of sharing resources across concurrent kernels [6, 67, 70], related GPU security concerns are not well studied. Concurrent kernels running on the same SM, possibly belonging to different processes, share GPU resources at a fine granularity. Thus, it becomes possible to exploit indirect information flow through the shared microarchitecture structures to conduct covert or side channel attacks.

2.2 Threat Model: Covert and Side Channel Attacks on GPUs

We consider two threat models: covert channel and side channel attacks. For a covert channel attack we assume two colluding kernels

Figure 1: Baseline GPU: protected SM resources are shaded.

that concurrently share the same GPU and desire to communicate sensitive data across protection boundaries. Contention channels exploit differences in observed behavior caused by the presence or absence of contention on microarchitectural resources. They are able to measure contention by timing their operations, or by inspecting the hardware performance counters which are available in user mode in the current generation of GPU drivers [46]. In a side channel attack context, one kernel (the attacker) is observing contention to infer secret information about a victim if its resource access pattern is dependent on sensitive data.

As an example covert channel scenario, a Trojan application can create contention on shared resource by replacing the contents of a cache set to encode '1' and leave the resource idle to encode '0'. The Spy application, on the other side accesses the cache and measures its access time to decode the transferred bit. Similarly, a Trojan application can create contention by excessively using execution units, warp scheduler, and instruction fetch units to encode '1' and leave those resource idle to encode '0', which spy can then decode.

3 GPUGUARD KEY IDEA: HIERARCHICAL SECURITY DOMAINS

We propose GPUGuard, a holistic protection framework for GPUs to detect and defend against contention-based attacks. Figure 3a presents an illustrative example of GPUGuard. In this example, we assume that there are four applications concurrently running on the GPU, including two regular applications, a Trojan application, and a Spy application. Each application launches kernels to the shared GPU, which may be assigned to execute on the same SM. A GPUGuard classifier is designed to detect collusion between two kernels. Our defense mechanism will reschedule the suspected kernels into isolated security domains. For example, in Figure 3a, the GPUGuard identified that Kernel 3 and 4 are suspicious. The GPU now creates three isolated security domains (SD1-3) and issues Kernel 1 and 2 to SD1, Kernel 3 to SD2, and Kernel 4 to SD3. In this way, the timing channel between Kernel 3 and 4 is closed.

3.1 Security Domains

It is critical to define the scope of a security domain to minimize the performance overheads. Rather than using a one-size-fits-all approach GPUGuard uses a hierarchy of security domains with varying scopes. Depending on the type of attack detected GPUGuard employs a security domain that encompasses only those resources that are being used in channel formation.

Figure 2: Existing partitioning techniques on GPUs

Coarse-grain security domain using temporal partitioning. At a coarse grain level, GPUGuard uses temporal partitioning which is essentially time division multiplexing. As shown in Figure 2a, using temporal partitioning each kernel can only execute in its assigned time slots. Note that in this example the Spy kernel K1 (blue) can only execute in the odd time slots, while the Trojan kernel K2 (yellow) can only execute in the even time slots. In this case, the execution of the kernels are completely isolated. As shown, the kernel waiting time Δt_1 and Δt_2 are independent of the execution time of the other kernel.

GPUs rely on kernel level preemption (essentially context switches) to enforce temporal partitioning: when K1 reaches the end of its assigned slot, the GPU needs to save the kernel context, preempt the kernel and then schedule the next kernel K2 to run on the GPU. After K2 uses up its time slot, K1 must reload its context and then resume execution. Context switching on GPUs is more expensive than on CPUs [52, 66]. On the Pascal architecture which supports optimized preemption, kernel preemption takes 100 micro seconds – a 100K cycle penalty even when using a 1GHz GPU. Thus temporal partitioning alone is an expensive solution for providing isolation.

Finer-Grain security domains using spatial partitioning. The next level of isolation can be achieved through a hierarchy of spatial partitioning approaches. Adriaens et al. proposed spatial partitioning at the granularity of SM to partition GPU resources across multiple kernels, primarily to improve resource utilization [6]. This technique can be easily adapted to create multiple security domains on the same GPU. Figure 2b shows an example with 16 SMs and with spatial partitioning the Trojan kernel K1 occupies SMs 0-7 while the Spy kernel K2 occupies SMs 8-15. Because the kernels are separated on different SMs, no contention can be established through intra-SM resources, such as execution units or L1 caches. However, spatial partitioning at the granularity of an SM does not protect against attacks through globally shared resources such as the L2 cache, memory channels, and interconnection network.

Spatial partitioning is a heavy handed solution: the entire resources are strictly partitioned leading to significant performance hit when there are no attacks. A finer grain isolation can be provided using intra-SM partitioning. It has been demonstrated that sharing a single SM (intra-SM sharing) across multiple kernels can provide higher system throughput and better utilization of GPU resources compared to spatial partitioning [13, 66, 67, 70]. Hence, we argue in this work that the benefits of intra-SM sharing must be delivered to the end user, without compromising the potential security risks.

Putting it all together. Figure 3b summarizes how GPUGuard may use both temporal and spatial partitioning at various granularities to achieve the required isolation with minimum performance

Figure 3: An illustrative example of a GPU system with GPUGuard (a) monitor and detect timing attacks; (b) select a security domain level based on specific attack type (our contributions are highlighted)

overhead. As shown in Figure 3b, we can partition the four SMs into two security domain SD1 and SD2, each containing two SMs. GPUGuard may also use intra-SM partitioning of parallel execution lanes or utilizing other underutilized resources inside an SM to create multiple security domains. For example, assuming that there are four execution lanes in the special functional units inside an SM, GPUGuard can assign the first two lanes to SD1 and the remaining two lanes to SD2. This partitioning can be achieved through security aware warp folding which we will introduce shortly. Note that many GPU workloads have shown significant warp level divergence [22, 69], and lane level partitioning in many cases improves the resource utilization. Through this hierarchy GPUGuard activates the right amount of isolation for preventing collusion while still maximizing the benefits of fine-grain intra-SM resource sharing across kernels.

4 ATTACK TYPES

4.1 Description of Typical Attacks

We assume a conventional covert communication scenario with a Trojan and Spy kernels from two different applications that are co-executing on the same GPU and wish to communicate covertly. The attack benchmarks we used in this paper are intra-SM and inter-SM microarchitectural covert channels on GPUs, categorized into five groups modeled on recently published attacks [45] and summarized in Table 1. These five types of attacks have been demonstrated to occur in current generation GPUs and hence we chose to tackle these five attacks. Based on current generation GPU microarchitectural details that are publicly known, we believe these five attacks cover a wide range of information leakage through shared resources.

In our experiments, we couldn’t reliably measure timing variance through shared memory bank conflicts to construct a covert communication, hence, we do not consider such attacks in this work. This is also justified by Naghibijouybari et al. [45], who observed “little measurable effect in the timing of a competing kernel”. We surmise it is because of the uncertainty in shared memory bank conflicts. It is possible with careful reverse engineering that such an attack can be constructed, but we will leave it for future work. We also do not directly address L2 cache attacks in this paper, where the L2 cache may be shared across multiple SMs. In this case we believe that GPUGuard can simply switch to coarse grain temporal partitioning to mitigate L2 cache attacks. Finally, we believe that our decision tree classification approach is general enough to tackle new intra-SM attacks by including such attacks into the training set to retrain the classifier and reprogram the detection units.

Table 1: Description of attacks in our dataset. Different variations of each attack type are present [45].

L1 Cache Attack: Trojan accesses one or multiple cache set(s) to send "1" and Spy accesses the same set(s) and measures the access time. Attacks can target L1 constant, instruction, data or texture caches.
Execution Unit Attack: Trojan threads do a number of double or single precision ops to create contention on INT/FP units to send "1". Spy threads do the same ops and measure the execution time.
SFU Attack: Trojan threads do a number of special function operations (like <code>__sinf</code>) to create contention on SFUs to send "1". Spy threads do the same operations and measure the execution time.
Scheduler Attack: These are timing channels created as a side effect of a primary EU and SFU attack, typically leaking information by observing warp scheduler contention.
Atomic Attack: Trojan threads do atomic ops on global memory addresses (one particular address or strided addresses to achieve coalesced or uncoalesced accesses) to send "1". Spy accesses the same pattern and measures the access time.
*"In all attack scenarios, high measured latency by the Spy is decoded as "1" and low latency is decoded as "0".

4.2 Attack Detection

To detect such covert channel attacks, GPUGuard continuously monitors the activation and resource usages of running kernels. GPUGuard employs a decision tree classifier that reads readily available performance counter metrics to track kernel behaviours and identify suspicious contentions. We elected to use decision tree, a machine learning model, as it is robust to noise that can fool deterministic threshold based detectors.

The attack detector continuously monitors the execution status, resource utilization and various other performance counters (e.g. cache miss rates) for different active kernels. A selection of features is extracted periodically (once every 1000 cycles in our setup) from the collected performance counter statistics and are used by a machine learning classifier to detect whether there are suspect timing channels between any two concurrent kernels. The output of the classifier is a label we assigned to different attacks and normal application. We develop a multi-class classifier that not only detects suspected timing channel presence, but also determines the target shared resources that are used to communicate covertly.

Decision Tree Classifier. Without loss of generality, we use a decision tree based classification algorithm to classify the attack type. Decision trees are a supervised learning algorithm in which the classification model is built by breaking down a dataset into progressively smaller subsets based on a feature value at every decision point. This classification may be viewed as a tree structure where each level progressively refines the classification of an input. The two most important advantages of decision trees over other classification models are: (1) small hardware overhead (see Section 7.4); and (2) direct isolation of relevant feature elements

Table 2: All collected features to create dataset.

Instruction features: # of SP-INT , SP-FP , SFU and LD/ST issued # of decoded ALU , SFU , ALU-SFU , INT , FP , Load & Store ops, # of decoded Branch, Barrier, Memory barrier, Call, Ret, Atomic ops , # of decoded INT MUL/DIV, FP MUL/DIV, FP sqrt/log/sin/exp , SP-FP , INT/FP/SFU/MEM instructions processed in decode stage) # of decoded Tex/non-Tex ops, some cache related instruction opcodes such as LD_OP, LDU_OP, Total stall: # of cycles warp scheduler issues no warp to execute Total issue: # of cycles warp scheduler issues a warp to execute
FU features: SP-INT, SP-FP , SFU and LDST util. at execute stage
Mem features: # of cycles warp scheduler stalled by long memory latency L1D accesses/misses/evictions, L1C accesses/misses/evictions L1I accesses/misses/evictions , L1T accesses/misses/evictions L2 accesses/misses/evictions , L1C, L1D, L1I and L1T accesses per set L1D read hit/miss per set, L1D write hit/miss per set L2 read hit/miss per bank, L2 write hit/miss per bank

through an estimate of information gain. We use the ID3 algorithm to build the tree [54], which uses entropy and information gain to identify appropriate decision points, and employs a top-down greedy search through the space of possible branches with no back-tracking. Once the tree is constructed based on the training data, a new instance is classified by starting at the root node of the tree, testing the attributes (or feature elements) specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated to reach a leaf node [43].

As with any classification algorithm there are two issues that we must address. First, the classifier may have both false-positive and false-negative classifications. We show later in our results that 0% of the malicious kernels were classified as benign and only 8% of benign kernels were classified as malign. Hence, we believe misclassification is not a concern. The second challenge is that an attacker may design Trojan/Spy pairs that continuously shifts between benign behavior and malicious behavior that may cause the decision tree to lower its threshold. But to alter such a behavior the attacker must first observe the information leakage to adapt, but such covert channel bandwidth is going to be drastically reduced in the first place using GPUGuard. Hence, the time to adapt will be extended significantly which we believe is the primary deterrent in covert and side-channel attacks.

Feature Selection. The decision tree model is built using a training input set consisting a large collection of features that correspond to various resource utilization indicators, covering different types of covert channels. Table 2 lists all the features that were collected as inputs for the decision tree model.

The feature vectors were divided based on whether they came from the training or testing data set benchmarks. The data in each of the sets was obtained from running the benchmarks and attacks that belong to that set only. The decision tree model is trained using the training input set. Once the training is complete, we obtain the decision tree model parameters which include the weight of each of the features in determining the attack type. We then optimize the tree by pruning unimportant features. To identify the most important feature elements, we use the decision tree model for multiclass classification to compute the importance factor of each feature element based on the training subset. Through this selection, we were able to reduce the 234 features to a set of 24 that are identified as the most important features. As a result our trained decision tree

classifier takes as input a select set of microarchitecture features collected at runtime, which are listed in bold font in Table 2.

Pruning the feature set allows us to reduce the complexity of the classifier, without sacrificing detection accuracy. The identified important features are related to the resources that are used by our attack benchmarks to create contention; intuitively, the decision tree checks the utilization of each resource to identify the presence and type of contention. We then classify the test set based on our decision tree model using two-fold cross validation. Section 7 evaluates the accuracy of our online detection, based on classification results for each instance in the test set.

Feature Collection. The 24 selected features are sampled for each active kernel periodically (every 1000 cycles) and then fed to the classifier. An effective online detector needs to filter out occasional false classifications and quickly signal true malicious behavior. Ozsoy et al. [49] adopted an Exponentially Weighted Moving Average (EWMA) approach for their binary classification. For the same reason, we design a voting algorithm that first considers classification results from the decision tree classifier (making the time-series consist of 0's for benign application and 1-5's for five attack types.) We then use a window of 10 of these decisions and pick the majority decision as the correct answer and output whether an attack is in progress and, if so, which of the five attack types is being used. Synthesis results show that the extra hardware overhead of the classifier is not high (shown in Section 7.4).

Detector Adaptation. Although we believe the attacks in our training set cover exploitable shared resources, if in the future new contention channels are encountered due to microarchitectural enhancements to SMs, they can be addressed by retraining the model with new training data. Since classification is implemented in hardware we need to provide the ability to adapt the detector at runtime if new attack vectors are identified during in-field operation. To make this adjustment feasible we expect the boot loader to essentially re-program the node weights in the tree, which are implemented using registers.

5 TANGRAM: ATTACK MITIGATION

The second component of GPUGuard is a defense against covert channel attacks once an attack is detected. We refer to the GPUGuard's defense mechanism alone as the Tangram (shown in Figure 4). Tangram uses a hierarchy of resource slicing to prevent the attack types that are detected. The approach to partition resources is similar to a dissection puzzle called *Tangram*. Tangram uses hierarchical security domains to separate colluding kernels, starting with fine grain intra-SM slicing and then gradually moving towards coarser-grain inter-SM slicing, depending on the attack type. In particular, GPUGuard uses intra-SM spatial partitioning of resources to mitigate L1 cache attack, execution unit, and SFU attacks. For Atomic attack, GPUGuard uses temporal partitioning. The only partitioning that is not used in GPUGuard is partitioning of SMs into clusters of security domains, since all attacks can be mitigated with either intra-SM spatial partitioning or temporal partitioning.

As shown in Figure 4, and described in more details below, security domain are created using sliced data pipelines (1), controlled memory request traffic within memory units (2), rate limited scheduling in the warp scheduler (3), and L1 fetch arbiter (4).

The maximum number of security domains for each resource is set as four. Hence, at most four suspicious kernels can be isolated in a given SM. This is a reasonable limitation since more than four concurrent kernels inside the same SM have diminishing returns in performance benefits [66].

Mitigating Execution Units and SFU Attack: Datapath Slicing. As in our baseline GPU, we assume 32 execution lanes within a single SM. Datapath slicing splits the 32 execution lanes into four slices, each with eight lanes. Note that GPUs already treat the 32 execution lanes as a collection of clustered lanes that are operated semi-independently of others. For instance, AMD’s subwarp execution model allows multiple warps to share the same set of datapaths. No matter if they are from different kernels or protection domains subwarp execution requires the same control logic for each of the sub datapaths.

GPUGuard relies on the subwarp execution model where each slice is allocated to a single kernel thereby preventing one kernel from observing the SFU and execution unit usage of another kernel. Note that datapath slicing does not change the number of threads inside a warp or the number of register file banks in the SM. Datapath slicing folds the 32 threads in a warp into four quarter-warps, which are then issued in succession. The threads in a warp are shifted in successive cycles to align with the slice in a linear fashion: threads 0-7 are mapped to lane 0-7 in the first cycle, threads 8-15 are mapped to lane 0-7 in the second cycle, and so on.

Figure 4: Tangram overview (black units are modifications)

Datapath slicing allows for concurrent warps from different security domains to be executed simultaneously on isolated datapath slices. With four slices each warp executing on a slice needs to be executed in four consecutive cycles, incurring a three cycle delay for a given warp. While a sliced pipeline delays the execution of each warp, the total throughput of the GPU is similar to, or in some cases even better than, a unified 32-lane pipeline. When there are control divergence, a sliced pipeline can help fill out the idle resources more effectively and improves the performance.

Tangram relies on some additional hardware support for executing multiple sub-warps concurrently. Tangram adds a set of

32 registers immediately before and after each data pipeline slice (shown in Figure 4), so that the registers can be consumed and updated in multiple successive cycles. However, the proposed design does not require any modification to the interconnection network between the register banks and the data pipeline. The interconnection still forwards the registers into the same original 32-wide registers of the pipeline. Then a 1-to-4 channel de-multiplexer is added to shift the register to the add-on 32-wide register of a particular slice. The write-back process follows a similar path: the results of a slice will be stored locally in the add-on registers and shifted out to the original 32-wide write-back register through the 4-to-1 channel multiplexer. After each sub-warp finishes, the caching register shifts left by 8 words to feed into the next sub-warp.

Cache partitioning is a reasonable mechanism in CPUs to create separation and remove contention channels. Since GPUs have multiple cache types, we use a novel cache redirection approach instead of partitioning. To mitigate covert channels from constant cache accesses, Tangram dynamically re-routes traffic from constant cache to use the L1-D cache. For this purpose, Tangram relies on the decision tree classifier to categorize the attack type as the constant data cache attack, and in which case the constant values accessed by one malicious kernel are moved to the L1-D cache. Once an attack is detected, Tangram monitors constant data load operations from one of the malicious kernel. The load address is first looked up in the constant cache. If there is a hit in the constant cache, Tangram marks it as a miss and evicts the data from constant cache and places a miss fill request to bring the data back into the L1-D cache. For this purpose, Tangram uses the constant data address to lookup the L1D cache to find a victim cache line. The victim cache line is evicted and constant data is then stored in that line. Thus, the channel through the constant cache is eliminated. If the attacker detects the protection and then changes to use L1 data cache, Tangram will eliminate the covert channel formed through L1 data cache using cache bypassing. Previous studies show that the GPU L1D cache miss rate is so high so that the performance is not harmed when the GPU L1-D cache is bypassed [11, 23, 34, 35, 61, 68]. Therefore, Tangram selectively bypasses the L1-D cache requests if the attacks are detected on the L1D cache instead. Since it is not possible to re-purpose the read-only constant cache for potentially read/write operations from a regular L1-D cache, our approach simply picks either the Spy or Trojan kernel and mark all its load/store operations as non-cacheable.

Mitigating Scheduler Attacks. When a kernel modulates Execution Units/SFU/Cache accesses, in addition to the contention on one specific unit, the attack often creates weaker side attack contention on other shared resources, including shared warp scheduler and instruction fetch units. Thus we add the following techniques to mitigate those side attack channels.

Rate Limiting Warp Scheduler. The GPU warp scheduler selects which warps will be issued to execute in the next cycle from a pool of all the active warps in the SM. The warps from multiple security domains can compete for the scheduling bandwidth and issue timing attacks. Our baseline warp scheduler selects the next available warps to issue based on the last issued first and then the oldest order. When all the warps from a kernel are stalled, all the scheduling cycles will be given to the next kernel. On the other hand, if the warps from a kernel are always ready to execute, it

will consume all the scheduling bandwidth and starve the other kernel. This interference in scheduling can be manipulated for timing attacks. Therefore, we enhance the warp scheduler with a rate limiter, so that scheduling cycles will be fairly distributed. For example, if one warp scheduler can issue up to two warps in each cycle, and there are two security domains, we will ensure that only one warp from each security domain can be issued. In the case of four security domains, one warp from each security domain can only be issued every other cycle.

Instruction Fetch Arbitration. Tangram prevents contention on the instruction cache using instruction fetch arbitration. A malicious kernel may intentionally saturate the instruction fetch bandwidth. Tangram alters the control unit in the L1 fetch arbiter so that it will successively fetch from different security domains in a round-robin manner. Therefore, each security domain gets fair access while simultaneously preventing resource hogging by a single kernel.

Figure 5: Logical View of Tangram Security Unit

Mitigating Atomic Attack: Temporal Partitioning Global memory attacks are primarily carried through atomic operations to measure contention in memory channel. When such an attack is detected by our classifier we fall back on temporal partitioning of the SM. In particular, we context switch the two malicious kernels (without perturbing the normal kernels). Since context switching is an expensive operation, we only use this option for tackling covert channels formed through atomic operations. The associated performance penalty is primarily paid by the colluding kernels, and only in very rare cases by regular kernels if they were misclassified as colluding kernels.

5.1 Security Unit Implementation.

The various schemes described above defend against different types of attacks. Based on the attack classification, the coordination across various schemes is handled using the *Tangram Security Unit* (TSU), shown in Figure 5. When all the kernels are executing normally, TSU keeps all kernels in a single security domain and none of the resource partitioning schemes described above are activated. However, when the classification algorithm detects an attack, the warp ids of the two colluding warps are sent to the TSU. TSU then activates the resource splitting across security domains. Each kernel is assigned a security domain id and all warps in that kernel execute within that security domain.

TSU maps warp IDs to security domain IDs using the *Security Domain Table* (SDT). The obtained security domain ID is used as the index of *Tangram Table*, which tracks the resources assigned

to each security domain. In this way, TSU guarantees kernels are executed in isolated security domains.

Each Tangram table entry consists of a 3-bit instruction fetch token, a 3-bit warp scheduling token, a 4-bit datapath slice mask, and a 16-bit cache utilization mode indicator. The instruction fetch token and warp scheduling token are used to determine a warp’s scheduling slots out of the total scheduling cycles during a given observation window. For example, if the warp scheduling token for SD1, SD2, and SD3 are one, one, and two, respectively, and the warp scheduler can issue two warps in each cycle, then in a two cycles window, the number of warps can be issued by each of them is one warp for SD1, one warp for SD2, and two warps for SD3. To ensure fair access for different security domains, all the tokens will be initially set to one.

The datapath slice mask has 4 bits, each corresponding to a datapath slice (Recall that we have four datapath slices). If bit 1 of the datapath slice is cleared (set to 0) for a warp then that warp cannot be issued to datapath slice 1. Thus each warp is restricted to execute only on those slices whose corresponding bits in the datapath slice number are set to 1.

The last field in the Tangram entry is the 16-bit cache redirection mode, which is used to provide fine grained security protection while accessing caches. In our baseline GPU there are four caches (shared memory, L1 D-Cache, constant cache, and texture cache). Each of the above four caches has a corresponding 4-bit cache redirection mask (so a total of 16-bits). When an incoming memory request is bound for a given cache type, Tangram looks up the 4-bit mask associated with that cache to determine if the request need to be redirected to another cache type. For instance, if a constant cache access request from a security domain is isolated to use L1 D-cache then all requests to the constant cache will use the corresponding 4-bit mask in Tangram to initiate that redirection. Similarly if all the four bits associated with a given cache type are zero, the request traffic control logic will redirect all requests from that cache type to go to the global memory (in response to a detected atomic attack). In this way, the access to caches are always going to be re-directed to the other under-utilized resources or the global memory to guarantee the execution isolation.

The TSU access latency is smaller than a clock cycle, and is off the critical path: the instruction fetch token is obtained one instruction in-advance; the warp scheduling token is retrieved in parallel with accessing the SIMT stack; the datapath slice mask and cache access mode are collected by the operand collector with other operands. The average power and area overhead of TSU are negligible compared to the entire system. Detailed analysis is presented in Section 7.4.

6 EXPERIMENTAL SETUP

In this section, we discuss the experimental methodology.

6.1 Architecture

We use GPGPU-Sim v3.2.2 [8], a cycle accurate timing simulator, in our evaluation. Our configuration parameters are described in Table 3. For Volta architecture, the parameters are set based on NVIDIA’s white paper [5], and HBM2 timing is set based on previous work [48]. The simulator was extended to run multiple applications concurrently, and we abide by the GPGPU-Sim model

to assume that all the data fed into a kernel fits in the GPU device memory. We obtained the same set of attack benchmarks from the authors in [44, 45]. Those attacks are fully validated in GPGPU-Sim against real GPU hardware, and hence our results using the GPUGPU-Sim simulation infrastructure accurately model the attacks observed in hardware.

We evaluated the performance impact of multiple defense schemes: temporal partitioning (labeled as TP in all our results), spatial partitioning through clustered SMs (labeled as SP in all our results), and GPUGuard against having Trojan and Spy kernels insecurely sharing an SM without protection. In temporal partitioning, each kernel is assigned an execution window of 50K cycles in a round robin manner. At the end of the 50K cycle window, we will preempt the current kernel and switch to the kernels in the next security domain. In clustered SM approach SMs are evenly allocated to Trojan and Spy kernels. In GPUGuard, datapath slicing, fair warp scheduling and instruction fetch are turned on for all benchmarks and for Atomic attacks GPUGuard falls back on temporal partitioning as we described earlier.

6.2 Workloads

We selected 40 readily available applications as benign samples from a collection of benchmark suites [8, 9, 47, 59]. We then extended the GPU covert channel attack applications based on attacks obtained from the authors of [44, 45] and hand-coded 250 different pairs (Spy and Trojan) of malicious applications, which cover atomic operation attacks (Atomic), constant cache attacks (Cache_A and Cache_B), attacks on execution units (ADD and MUL), and special functional units (SFU). These different attacks create orthogonal types of channels between Trojan and Spy by using different resources. They also differ with respect to implemented optimizations (e.g., Synchronization via handshaking through different cache sets [45] and Multi-bit communication), as well as the communication rate and the communicated data. Thus, the attack variants exhibit substantially different contention behavior. We also implemented prime-and-probe style side-channel attacks on constant cache which is run with different normal programs.

We split both the benign applications and attacks into separate training and testing sets, so that 60% of the benchmarks are used as training set and other benchmarks as testing set. The benchmarks are run on the GPGPU-Sim simulator [8] to collect a 24-entry feature vector for each kernel at each sampling window. Nvidia nvprof report GPU performance counter values only after each kernel termination, that is too coarse grain for our scheme. We empirically set the default window size to 1000 cycles, while providing a comparison of window sizes in Section 7.1. These feature vectors are the input to the decision tree classifier. The output is a label we assign: (0) normal application, (1) L1 cache attacks, (2) global memory attacks with atomics, (3) execution units attacks, and (4) SFU attacks.

To reduce the simulation time, a subset of the covert channel attacks is used for system evaluation: four versions for each type of attack, with various inputs, programming styles, and implemented optimization. For constant cache attacks, we evaluated both the base attacks that contend using one fixed set (Cache_A) and the improved attacks that continues to probe all the cache sets to communicate (Cache_B). To quantify the performance impact of control

divergence, half of the execution unit benchmarks have little divergence, while the other half have 25% - 50% of control divergence. We further run the defense schemes on 34 randomly selected, normal application pairs from the detection sets to study the performance penalty of FP predictions. The application parameters follow the benchmark sets used in [70].

6.3 Synthesis

The decision tree based classifier and control logic of GPUGuard were designed and verified in Verilog RTL, and synthesized with the FreePDK 45nm library [58] using the Synopsys Design Compiler [60]. FabMem [12] is used to model the security domain table and the Tangram table within the Tangram Security Unit, and also register buffers. The latency, energy, and area overheads were all taken into account.

7 EVALUATION

In this section, we evaluate and discuss the accuracy of the attack classifier, and analyze the performance and energy impact of the proposed defense scheme to the entire system. We also report the latency, area, and power overheads of the components in the proposed GPUGuard technique.

7.1 Detection Accuracy

Figure 6a shows the confusion matrix. The first column indicates the actual attack type and the last row shows the predicted attack type; attack types are numbered to match description of attacks shown in Table 1. As shown by the strong diagonal matrix the predicted and actual attacks are close in most cases. The classification accuracy is measured to be 93.8% with window size of 1000 cycles. The detection accuracy with window size of 5000 cycles was even higher at 97% for this experiment, but we selected 1000 cycles for faster detection.

The accuracy is also measured using true positive (TP), false negative (FN), true negative (TN), and false positive (FP). In our results, 8.5% of regular applications were misclassified as malicious applications (FP), but 0% malicious applications were misclassified as regular applications (FN). FP cases cause the system to react unnecessarily (performance penalty) while FN evades detection which represents a security concern. Note that the TP rate of the classification is 91.5% and TN rate is 100%, indicating that our detection can reliably signal a malicious behavior, and if not, the running applications are truly benign.

Covert channel attacks rely on contention on shared resources to communicate encoded messages, and the decision tree classifier takes into account many features that are related to such contention, including resource utilization, cache misses and many others. The model trains a decision tree predictor by optimally setting the thresholds of the features to detect the contention level. The structure of the decision tree model fits very well with the problem we are solving, and therefore, yields a high accuracy. It must also be noted that the data set we used to build the decision tree is *disjoint* from the data set of benchmarks used for evaluation.

Multiple channel attack. Training and evaluation described above are performed using applications communicating over a single channel. Since we monitor performance counters that capture the contention for all cache, memory, and execution units, if the

Table 3: Configurations of Fermi, Kepler and Volta architecture.

	Fermi (GTX580)	Kepler (K40)	Volta (Titan V)
Compute Units (SIMD width 32)	16x2 SP, 16x1 SFU, 700 MHz	16x6 SP, 16x1 SFU, 745 MHz	84x2 SP, 84x1 SFU, 600 MHz
Regs / Shmem / Max # CTA	32768 / 48 KB / 8	32768 / 48 KB / 8	262144 / 96 KB / 32
Warp Schedulers (1 per SM)	max issue 2 warps per cycle, default gto	max issue 4 warps per cycle, default gto	
L1 Cache	16KB 4-way L1D, 4KB 4-way L1C		16KB 4-way L1D, 8KB 4-way L1C
L2 Cache	768 KB	768KB	4608KB
Memory Model	6MCs, GDDR5: $t_{CL}=12$, $t_{RP}=12$, $t_{RC}=40$, $t_{RAS}=28$, $t_{RCD}=12$, $t_{RRD}=6$		64MCs, HBM2: $t_{CL}=12$, $t_{RP}=12$, $t_{RC}=40$, $t_{RAS}=28$, $t_{RCD}=12$, $t_{RRD}=6$

attacker changes its behavior to communicate over a different hardware resource or attempts a multiple channel attack, the contention is also detected. To get more accurate classification, we need to add those samples to our training set with the correct label. We hand crafted a multiple channel attack that combines all four attacks listed in Table 1. The existing scheme successfully classified the application as an attack. To support concurrent multiple channels on different resources, one straightforward solution is to enable temporal partitioning, once a multiple channel attack is detected.

Comparison to neural network model detection. We implemented a multi-layer perceptron (MLP) artificial neural network model to compare the classification results to our decision tree based detection. In our MLP implementation the input layer contains neurons equal to the number of features (24 important features in our case), and the output layer contains neurons equal to the number of classes (five in our case). The data is fed to the input layer of the network, and after the feed-forward propagation, the output layer of the network contains a vector of values. The neuron containing maximum value determines the class of the data. The error in prediction is calculated and using this error the weights of the network are modified by gradient descent algorithm. The MLP based classification accuracy is measured at 87%. Figure 6b shows the total confusion matrix which visualizes the performance of MLP classification. Decision tree based classification outperforms MLP in our dataset.

Robustness. We also evaluated the detector accuracy when attack kernels (Spy and Trojan) are running with normal kernels at the same time. In this situation, it is harder for the detector to accurately classify attacks due to the contention noise introduced by normal kernels. Our results show that the classification accuracy for decision tree based and MLP based detection are 91.5% (95% with a window size of 5000) and 85.1% respectively. Figure 7 shows the total confusion matrix for these two detection schemes.

Figure 6: Confusion matrix for decision tree and MLP

To further evaluate the detection robustness, we consider the attack benchmarks that are intentionally designed to avoid detection by lowering the communication bandwidth. Specifically, we change the Spy and Trojan codes by adding extra delay between communicating two consecutive bits, to reduce the channel bandwidth by 2x, 10x up to 10^5 x. 10^5 x slow down reduces the absolute

Figure 7: Confusion matrix for decision tree and MLP based classification under higher contention noise.

BW from 30kbps to 0.3bps for constant cache attacks. Based on our experiments, our detector accurately detects the contention when Trojan tries to send a '1' to Spy (with the same accuracy of the attacks without slowdown). On the other hand, the classifier will not detect applications as attacks in the longer idle periods (no communication), since there is no contention.

7.2 Mitigating Side Channel Attacks

Similar to covert channel scenario, in contention based side channel, a malicious application (Spy) accesses to different hardware resources and either measures the access time or its own performance counters to extract some information about concurrent workloads (victim) running on GPU. Due to the large number of active threads, and the relatively small cache structures, it is hard to achieve high-precision prime-probe or similar timing attacks on GPUs. In a recent work, Naghibijouybari et al. [46] demonstrated that GPU side channels are feasible by aggregate measures of contention through available GPU performance counters. To the best of our knowledge, this work is the only proposed side channel between two concurrent applications on GPU. In this subsection, we intend to evaluate our defense on this contention-based side channel.

We re-implemented the CUDA-CUDA side channel attack [46] on GPGPU-Sim. In this attack, a Spy application runs concurrently with a back-propagation workload from Rodinia benchmark and extracts the number of neurons in the input layer of neural network through side channel. We collected runtime per kernel performance counters for Spy application when it is concurrently running with back-propagation algorithm with input layer size varying in the range between 64 and 65536 neurons. We trained a Random Forest classifier with 10-fold cross validation and achieved accuracy of about 70% recovering the input layer size. The performance counter set available on real GPUs through NVIDIA tools are a bit different than those collected on GPGPU-sim during runtime, leading to lower side channel accuracy using the simulator. Since the Spy accesses different hardware resources to create contention, similar to the Spy and Trojan in the detection benchmarks, can be classified correctly as attack by our threshold-based classifier. Once the Spy has been detected as an attack, Tangram will be enabled promptly to separate the malicious Spy from other normal concurrent applications into different security domains. By our intra-SM

Figure 8: Performance impact of GPUGuard on (a) Fermi; (b) Kepler; (c) Volta; (d) Benign Apps on Volta, grouped by M_M (two memory bound apps), M_C (one memory + one compute bound app) and C_C (two compute bound apps). Measured by normalizing the execution time over intra-SM sharing without protection. (SP: Spatial Partitioning, TP: Temporal Partitioning)

isolation between two concurrent applications, we observed that the accuracy of attack significantly decreased obtaining essentially a random guess accuracy: Tangram was able to mitigate the attack by isolating contention between victim and spy.

7.3 Performance Impact

Figures 8a to 8c show the performance of all the defense schemes compared to intra-SM slicing without protection on NVIDIA Fermi, Kepler, and Volta architecture. For constant cache attacks, temporal partitioning (labeled TP) alone slows down program execution by at least 2.1x, while GPUGuard has significantly lower overhead. Clustered SM partitioning alone (labeled as SP) improves performance over temporal partitioning by 42% on average across three architectures, since it avoids kernel preemption. GPUGuard further reduces the overhead by 18% and 30% for ADD and SFU attacks over spatial partitioning. In our baseline Fermi configuration, the initiation interval of MUL application is as long as 16 cycles, and the longer latency caused by warp folding leads to some performance penalty. However, recent generations of GPUs greatly improved the latency of matrix multiply operation, which is likely to amortize this performance penalty. Overall, GPUGuard provides robust defense across multiple attacks and incurs less than 15% overhead, only when actively defending against an ongoing attack. Considering the mitigation techniques in GPUGuard almost only turned on when there is an attack detected, the performance overhead is much smaller than simply clustering SMs all the time.

In the attack on Atomic primitives, the Trojan kernel chooses to perform Atomic operations or not to encode '1's and '0's. The Spy kernel, on the other hand, will always issue Atomic operations. The back-pressure in memory system leaks whether the Trojan kernel is sending the '1' or '0'. The Spy kernel measures the Atomic instruction latency to decode this fluctuation. While concurrently executing a Trojan and Spy kernel in the same GPU amortizes the memory pressure, sharing the same SM can further reduce the congestion in load and store units. However, GPUGuard and spatial partitioning cannot close global memory channels. Since GPUGuard falls back on temporal partitioning for global memory

attacks, GPUGuard's performance is the same as temporal partitioning in Figures 8a to 8c for Atomic attacks.

Overall, the geometric mean of GPUGuard's performance is 94%, 96%, and 69% faster than of temporal partitioning, and 14%, 22%, and 10% faster than spatial partitioning in Fermi, Kepler, and Volta architecture, showing that it is possible to benefit from multiprogramming while maintaining protection against covert-channel attacks. Volta architecture has much higher HBM bandwidth, that reduces the number of warps stalled by long latency memory accesses. As we have more ready warps to schedule to execute, the protection schemes see less performance overhead. It's worth to note that, GPUGuard only incurs 8% of performance overhead against a baseline with no protection in Volta. Thus, another key benefit of intra-SM protection is system robustness. A mitigation technique with high performance penalty provides opportunity for Denial of Service attacks. By minimizing the performance slowdown, GPUGuard also minimizes such potential security risks. Moreover both temporal partitioning and spatial partitioning require preempting the running Trojan or Spy kernel. GPUGuard partitions the resources within a core, and does not incur preemption overhead. Finally, GPUGuard triggers defense only when attack is detected, thus has no overhead when there are no attacks detected.

Impact of False Positives. FP cases are rare. Nonetheless, we further study the performance impact of those cases when normal kernels are inaccurately classified as malicious in Volta. As shown in Figure 8d, temporal partitioning incurs 77% slowdown, spatial partitioning incurs 42% slowdown, while GPUGuard reduces that to only 15%. GPUGuard benefits the most for kernels that are memory + compute case, when complimentary sharing inside one SM is favored, and the least for compute + compute case, when datapath slicing reduced the opportunity for complimentary compute operation to share pipeline cycles.

7.4 Hardware Overhead

GPUGuard requires 24 performance counters for sampling the selected features for threat detection (each 10 bits), once every 1000 cycles. Among those, 10 counters, including the $SP - INT$, $SP - FP$, SFU issued counters, the $SP - FP$, SFU , and LD/ST utilization counters, L1C and L2 accesses and misses counters, are already provided

in the modern GPUs [4]. The V100 clock rate is 1.53GHz and the required sampling bandwidth is 45.9MB/s, negligible compared to the 900GB/s off-chip memory bandwidth.

The data collected by the counters are fed into the decision tree classifier. Synthesis results show that the classifier consumes 0.21mW per detection with 0.62ns latency. Since the classification is not on the performance critical path, the one cycle latency does not affect the overall system performance. The area overhead of the classifier is 0.001mm², which is small compared to the die area.

Table 4: Extra hardware overhead per SM at 45nm.

	Latency (ns)	Power (mW)	Area (mm ²)
RegBuffers	0.21	95.5	0.09
DEMUXs	0.06	0.28	0.002
MUXs	0.06	0.45	0.004
TangramUnit	0.57	1.08	0.001

Tangram security units require 22.5B RAM for keeping track of the security domain IDs and scheduling information. As described in Section 5, our datapath slicing design simply folds a warp and has low hardware overhead. Different from a full-blown variable sized warp architecture, proposed by Rogers et al. [56], our design does not require any modification to the interconnection network between the register banks and the data pipeline. To support our datapath slicing, 128B register buffers and four multiplexers/demultiplexers (32-bit width) are required per SM. The latency, power, and area overheads are broken down in Table 4. The area overhead is 0.1mm² per SM. Based on the activity factors collected by the timing simulator, the average power of the added hardware is 8.3mW per SM. We extract area and average power of 16 SMs from GPU-Wattch [33], which are 704mm² and 73W, respectively. GPUGuard results in 0.2% area overhead for GPUs with 16 SMs. The total power consumption was 0.18% of the overall system power.

8 RELATED WORK

Microarchitectural covert-channel and side-channel attacks have been widely studied on different resources on CPUs [20, 28, 32, 39, 41, 53, 63, 72]. There are many defense proposals to close side channel attacks on the CPUs which mostly focus on caches and memory controllers. These proposals include: (1) Static or dynamic partitioning of resources like L1 cache [15, 21, 50, 55] that can introduce unacceptable performance overhead and can only support a limited number of partitions with reasonable overhead and mitigation mechanisms like locking the critical cache lines with the support of OS and compiler [31, 64]. Liu et al. [37] proposes partitioning the LLC into secure and non-secure partitions and line locking the secure partition for defeating side channels. (2) Randomizing memory-to-cache mapping, including randomization in the replacement of the cache lines in the entire cache [65] and in the cache fill strategy [38]. (3) Adding noise to timing by manipulating time measurement structure of processor [40]. (4) Traffic control in memory controllers [57, 62]. Such defenses do not transit directly to GPUs or to covert channels. Our solution uses GPU-appropriate forms of partitioning triggered by detection of covert communication. We note that covert channels are more difficult to defend against than side-channels, so we expect that our defense will also defend

against concurrent kernel execution based side-channel attacks if those are possible.

Online detection of contention based covert communication is an alternative that is useful for closing covert channels. Chen et al. [10] present a framework to detect timing covert channel on shared hardware resources on a CPU by dynamically monitoring conflict patterns between processes. However, their framework is designed to detect alternating pattern of cache conflicts between Spy and Trojan and not to detect any variations of the attack—those channels that access to other shared resources concurrently. Yan et al. [71] propose a record and deterministic replay framework. It detects timing attacks by replaying execution on a different cache configuration to detect contention only on caches. In contrast, our solution monitors contention on all known resources as it occurs. There are also a number of online detection schemes based on hardware performance counters [14, 29, 49]. Unlike these prior solutions that only focus on malware detection, GPUGuard also provides mitigation solutions for preventing information leakage through covert channels.

Jiang et al. [24] present an architectural timing attack from the CPU to the GPU. The attack triggers an AES computation on the GPU, and times it showing that there exists correlation the measured time (which varies due to key dependent memory coalescing behavior) and the last round key in AES encryption. Recently, Kadam et al. [26] propose sub-warp randomization techniques to alleviate such correlation-based timing attacks by making the relationship between execution time and coalesced memory accesses less predictable. Jiang et al. present another timing side channel [25] based on correlation between execution time of one table lookup of a warp and a number of bank conflicts generated by threads within the warp. These attack are substantially different from our attacks with a different threat model. The attacks we protect against construct contention based covert channels between two malicious applications running concurrently. The defense proposed by Kadam et al. (which does not affect our attacks), can be combined with ours for protection against both types of attacks.

9 CONCLUSIONS

In this paper, we propose GPUGuard, a dynamic detection and defense mechanism against covert channel attacks. The detection uses a decision tree based design that is able to accurately detect covert- and side-channel attacks (100% sensitivity in our experiments). The detection algorithm feeds the classification results to Tangram, a GPU-specific covert channel elimination scheme. Tangram uses a combination of warp folding, pipeline slicing, and cache remapping mechanisms to close the channels with 8%-23% performance overhead when there are active attacks, and 15% for normal benchmarks categorized as attacks with only a small (8.5%) false positive rate. In all other cases GPUGuard pays nearly zero performance overhead.

ACKNOWLEDGMENTS

We would like to thank our shepherd, Sally A. McKee, and the anonymous reviewers for their valuable suggestions on how to improve this paper. This work was supported by DARPA-PERFECT-HR0011-12-2-0020, NSF CAREER-0954211, and NSF CNS-1619450.

REFERENCES

- [1] 2014. NVIDIA GeForce GTX980. Technical Report. NVIDIA.
- [2] 2016. *GRID VIRTUAL GPU*. Technical Report. Nvidia.
- [3] 2016. *Whitepaper: AMD multiuser GPU: hardware-enabled GPU virtualization for a true workstation experience*. Technical Report. AMD.
- [4] 2017. NVIDIA Profiler User's Guide. <http://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference>.
- [5] 2017. *NVIDIA Tesla V100 GPU Architecture*. Technical Report. NVIDIA.
- [6] Jacob T. Adriaens, Katherine Compton, Nam Sung Kim, and Michael J. Schulte. 2012. The Case for GPGPU Spatial Multitasking. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [7] Amazon AWS. 2019. Amazon Elastic Graphics. <https://aws.amazon.com/ec2/Elastic-GPUs/>.
- [8] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*.
- [9] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*.
- [10] Jie Chen and Guru Venkataramani. 2014. CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [11] Xuhao Chen, Li-Wen Chang, Christopher I. Rodrigues, Jie Lv, Zhiying Wang, and Wen-Mei Hwu. 2014. Adaptive cache management for energy-efficient GPU computing. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [12] Niket K. Choudhary, Salil V. Wadhavkar, Tanmay A. Shah, Hiran Mayukh, Jayneel Gandhi, Brandon H. Dwiell, Sandeep Navada, Hashem H. Najaf-abadi, and Eric Rotenberg. 2011. FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [13] Hongwen Dai, Zhen Lin, Chao Li, Chen Zhao, Fei Wang, Nanning Zheng, and Huiyang Zhou. 2018. Accelerate GPU Concurrent Kernel Execution by Mitigating Memory Pipeline Stalls. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [14] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the Feasibility of Online Malware Detection with Performance Counters. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [15] Leonid Domnitsier, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side-Channel Attacks. In *ACM Transactions on Architecture and Code Optimization, Special Issue on High Performance and Embedded Architectures and Compilers*.
- [16] Dmitry Evtvushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump over ASLR: Attacking Branch Predictors to Bypass ASLR. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [17] Dmitry Evtvushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
- [18] Google Cloud Platform. 2019. Cloud GPUs. <https://cloud.google.com/gpu/>.
- [19] HSA Foundation. 2014. Heterogeneous System Architecture (HSA): Architecture and Algorithms. In *Proceedings of the International Symposium on Computer Architecture tutorial (ISCA)*.
- [20] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [21] Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer. 2010. Achieving Non-Inclusive Cache Performance with Inclusive Caches - Temporal Locality Aware (TLA) Cache Management Policies. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [22] Hyeran Jeon and M. Annavaram. 2012. Warped-DMR: Light-weight Error Detection for GPGPU. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [23] Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi. 2014. MRPB: Memory request prioritization for massively parallel processors. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [24] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2016. A complete key recovery timing attack on a GPU. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [25] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2017. A Novel Side-Channel Timing Attack on GPUs. In *Proceedings of the Great Lakes Symposium on VLSI*. 167–172.
- [26] Gurunath Kadam, Danfeng Zhang, and Adwait Jog. 2018. RCoal: Mitigating GPU Timing Attack via Subwarp-Based Randomized Coalescing Techniques. *Architecture (HPCA)*.
- [27] Mehmet Kayaalp, Khaled N Khasawneh, Hodjat Asghari Esfeden, Jesse Ellwell, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. 2017. RIC: Relaxed Inclusion Caches for Mitigating LLC Side-Channel Attacks. In *Proceedings of the 54th Annual Design Automation Conference (DAC)*.
- [28] Mehmet Kayaalp, Dmitry Ponomarev, Nael Abu-Ghazaleh, and Aamer Jaleel. 2016. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53th Annual Design Automation Conference*.
- [29] Mikhail Kazdagli, Vijay Janapa Reddi, and Mohit Tiwari. 2016. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [30] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *arXiv preprint arXiv:1801.01203* (2018).
- [31] Jingfei Kong, Onur Acicmez, Jean-Pierre Seifert, and Huiyang Zhou. 2009. Hardware-Software Integrated Approaches to Defend Against Software Cache-based Side Channel Attacks. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [32] Esmail Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*.
- [33] Jingwen Leng, Taylor Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [34] Ang Li, Gert-Jan van den Braak, Akash Kumar, and Henk Corporaal. 2015. Adaptive and transparent cache bypassing for GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.
- [35] Chao Li, Shuaiwen Leon Song, Hongwen Dai, Albert Sidelnik, Siva Kumar Sastry Hari, and Huiyang Zhou. 2015. Locality-Driven Dynamic GPU Cache Bypassing. In *Proceedings of the International Conference on Supercomputing (ICS)*.
- [36] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).
- [37] Fangfei Liu, Qian Ge, Yuval Yarom, Frank McKeen, Carlos Rozas, Gernot Heiser, and Ruby Lee. 2016. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [38] Fangfei Liu and Ruby B. Lee. 2014. Random Fill Cache Architecture. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [39] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *Security and Privacy (SP)*.
- [40] Robert Martin, John Demme, and Simha Sethumadhavan. 2012. TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-Channel Attacks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [41] Clmentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and K. Rumer. 2017. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Network and Distributed System Security Symposium (NDSS)*.
- [42] Microsoft Azure. 2019. GPU-Accelerated Microsoft Azure. <http://www.nvidia.com/object/gpu-accelerated-microsoft-azure.html>.
- [43] Tom M. Mitchell. 1997. Machine Learning. In *McGraw-Hill Science/Engineering/Math*.
- [44] Hoda Naghibiyouybari and Nael Abu-Ghazaleh. 2016. Covert Channels on GPGPUs. In *IEEE Computer Architecture Letters*.
- [45] Hoda Naghibiyouybari, Khaled Khasawneh, and Nael Abu-Ghazaleh. 2017. Constructing and Characterizing Covert Channels on GPGPUs. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [46] Hoda Naghibiyouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered Insecure: GPU Side Channel Attacks are Practical. In *Conference on Computer and Communications Security (CCS)*. 2139–2153.
- [47] Nvidia. 2009. CUDA SDK 2.3. <https://developer.nvidia.com/cuda-toolkit-23-downloads>.
- [48] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W. Keckler, and William J. Dally. 2017. Fine-grained DRAM: Energy-efficient DRAM for Extreme Bandwidth Systems. In *Proceedings of the International Symposium on Microarchitecture*. 41–54.
- [49] Meltem Ozsoy, Caleb Donovan, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2015. Malware-aware processors: A framework for efficient on-line malware detection. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [50] D. Page. 2005. Partitioned Cache Architecture as a Side-Channel Defense Mechanism. In *Crypt. ePrint Arch.*

- [51] Sreepathi Pai, Matthew J. Thazhuthaveetil, and R. Govindarajan. 2013. Improving GPGPU Concurrency with Elastic Kernels. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
- [52] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. 2015. Chimera: Collaborative Preemption for Multitasking on a Shared GPU. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
- [53] Colin Percival. 2005. Cache missing for fun and profit. In *BSDCan*.
- [54] J.R. Quinlan. 1986. Induction of decision trees. In *Machine Learning*.
- [55] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [56] Timothy G. Rogers, Daniel R. Johnson, Mike O'Connor, and Stephen W. Keckler. 2015. A Variable Warp Size Architecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [57] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramanian, and Mohit Tiwari. 2015. Avoiding Information Leakage in the Memory Controller with Fixed Service Policies. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [58] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, Julie Oh, and R. Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *Proceedings of the International Conference on Microelectronic Systems Education*.
- [59] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniek Liu, and Wen Mei Hwu. 2012. *Parboil: A revised benchmark suite for scientific and commercial throughput computing*. Technical Report.
- [60] Synopsys. 2010. Synopsys Design Compiler. <http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx>.
- [61] Yingying Tian, Sooraj Puthoor, Joseph L. Greathouse, Bradford M. Beckmann, and Daniel A. Jiménez. 2015. Adaptive GPU Cache Bypassing. In *Proceedings of the 8th Workshop on General Purpose Processing Using GPUs*.
- [62] Yao Wang and G. Edward Suh. 2014. Timing channel protection for a shared memory controller. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [63] Zhenghong Wang and Ruby B. Lee. 2006. Covert and Side Channels Due to Processor Architecture.. In *Computer Security Applications Conference (ACSAC)*.
- [64] Zhenghong Wang and Ruby B. Lee. 2007. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [65] Zhenghong Wang and Ruby B. Lee. 2008. A Novel Cache Architecture with Enhanced Performance and Security. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [66] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous Multikernel GPU: Multi-tasking throughput processors via fine-grained sharing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [67] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2017. Quality of Service Support for Fine-Grained Sharing on GPUs. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [68] Xiaolong Xie, Yun Liang, Yu Wang, Guangyu Sun, and Tao Wang. 2015. Coordinated static and dynamic cache bypassing for GPUs. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.
- [69] Qiumin Xu and Murali Annavaram. 2014. PATS: Pattern Aware Scheduling and Power Gating for GPGPUs. In *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*.
- [70] Qiumin Xu, Hyeran Jeon, Keunsoo Kim, Won Woo Ro, and Murali Annavaram. 2016. Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
- [71] Mengjia Yan, Yasser Shalabi, and Josep Torrellas. 2016. ReplayConfusion: Detecting Cache-based Covert Channel Attacks Using Record and Replay. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.
- [72] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. 2018. Are Coherence Protocol States Vulnerable to Information Leakage?. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.