Covert Channels on GPGPUs

Hoda Naghibijouybari and Nael Abu-Ghazaleh
Computer Science and Engineering Department
University of California, Riverside
{hnagh001, naelag}@ucr.edu

Abstract—GPUs are increasingly used to accelerate the performance of not only graphics workloads, but also data intensive applications. In this paper, we explore the feasibility of covert channels in General Purpose Graphics Processing Units (GPGPUs). We consider the possibility of two colluding malicious applications using the GPGPU as a covert channel to communicate, in the absence of a direct channel between them. Such a situation may arise in cloud environments, or in environments employing containment mechanisms such as dynamic information flow tracking. We reverse engineer the block placement algorithm to understand co-residency of blocks from different applications on the same Streaming Multiprocessor (SM) core, or on different SMs concurrently. In either mode, we identify the shared resources that may be used to create contention. We demonstrate the bandwidth of two example channels: one that uses the L1 constant memory cache to enable communication on the same SM, and another that uses the L2 constant memory caches to enable communication between different SMs. We also examine the possibility of increasing the bandwidth of the channel by using the available parallelism on the GPU, achieving a bandwidth of over 400Kbps. This study demonstrates that GPGPUs are a feasible medium for covert communication.

Index Terms—Security, Covert Channel, GPGPU.

1 Introduction

General Purpose Graphical Processing Units (GPGPUs) are employed to accelerate a range of applications including security, computer vision, computational finance, medical diagnostics, and bio-informatics [8]. Despite their improving performance and increasing range of applications, the security vulnerabilities of GPGPUs have not been well studied; only a few research papers have examined security vulnerabilities of GPGPUs [11], [13], [16], [17], [19].

Due to their computational power, GPGPUs are used to accelerate security sensitive computations [1], [4], [14]. Any security vulnerabilities in these accelerators can leak critical information. Moreover, the possibility of side channels within the GPU or between the GPU and CPU can compromise sensitive data of regular applications, or enable covert channels to exfiltrate sensitive data when there is no direct communication channel available for that purpose. In particular, since the primary purpose of GPUs is often to manage the display, information leaked about the display (especially for smart devices where the display is also an input device), can expose the user activity and sensitive data input [2]. Thus, it is important to understand the vulnerabilities to enable future hardware designs and software defenses that mitigate them.

Covert channels are secret communication channels that exploit unintended side-channels for covert communication [3]. In the context of microarchitectural channels, the side-channels consist of shared microarchitectural resources. In a typical scenario, a *trojan* process communicates to a *spy* process either by creating contention on a shared resource or by affecting its state in a way that can be detected by the spy. For example, the spy may interpret contention on a shared resource (observed through longer latency of operations) as a communication of 1 and the absence of contention as the communication of a 0.

In this paper, we explore the feasibility of covert communication through the GPGPU. We review the general architecture of GPGPUs in Section 2. We investigate whether cooperative thread arrays (CTAs, also known as blocks) belonging to different programs can be colocated on the same streaming multiprocessor (SM) or across different SMs concurrently. For both cases, we explore what shared resources can be used as a covert channel using contention (in Section 3). We verify the usability of these channels in Section 4 by building a covert channel on a real GPGPU through constant memory at the L1 cache level when the CTAs are colocated on the same SM and at the L2 cache level when they are colocated on different SMs.

To best of our knowledge, this is the first work that considers covert channel vulnerability of GPGPUs. We review related work in Section 5, and present our concluding remarks in Section 6.

2 Background and Threat Model

A GPGPU consists of some several streaming multiprocessors (SM, or SMX) that access a shared device memory. The device memory is off-chip memory and is partitioned into global memory, constant memory and texture memory. These memories are shared among all streaming multiprocessors (SMs). They are also used to transfer data between the CPU and GPU. There is also a shared L2 cache to provide faster access to memory. Each SM has its own on-chip shared memory, as well as several L1 caches for the instruction, global data, constant data and texture data. Figure 1 presents an architecture overview of GPGPUs.

For example, one of our target devices, the Nvidia Tesla K40C, includes 15 SMs, each featuring 192 single-precision CUDA cores. Each SM uses four warp schedulers and eight instruction dispatch units [15]. The size of global memory, L2 cache, constant memory and shared memory are 12 GB, 1.5 MB, 64 KB and 48 KB respectively.

Our threat model consists of a standard covert communication scenario with a trojan and spy kernels from two different applications that co-exist on the same GPU. They wish to communicate covertly. We assume that no other applications share the GPU at the same time; this may be accomplished by

Manuscript submitted: 20-Apr-2016. Manuscript accepted: 08-Jun-2016. Final manuscript received: 12-Jun-2016

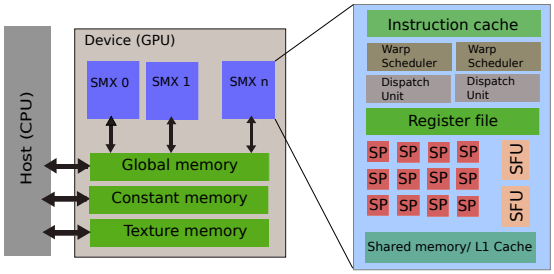

Fig. 1. GPGPU architecture overview

scheduling sufficient blocks of each kernel to take up the full GPU resources. In the future, we will consider the impact of noise from co-located applications on the covert channel.

3 Colocating Applications on GPGPU

Our goal is to create covert channels through contention on shared hardware resources in the GPGPU. As a first step, we need to control the placement of concurrent applications (in our case, the trojan which sends sensitive information, and the spy which receives it) on the GPGPU. The placement defines what resources the applications share and therefore what covert channels they may use.

We note that in the current generation of GPGPUs, different CPU applications may not launch multiple GPU kernels at the same time. However, a single CPU application may use multiple streams to launch multiple GPU kernels. Thus, one may think of a single scheduler process on the CPU that receives GPU requests and issues them concurrently. Moreover, general multiprogramming for GPUs will likely arrive in the near future [21], [23].

There is no information in the NVidia documentation about thread block assignments to SMs and co-location of two concurrent kernels executing on the same GPU; thus, it is necessary to reverse engineer the placement algorithm. First, we explore whether blocks belonging to two kernels can be co-located on the same SM. We launch two kernels on different streams. In each kernel, we read register (smid) for each block to determine the SM ID. In addition, we use the clock() function to measure the start time and stop time of each block. By using this information, and repeating the experiment for different numbers and configurations of blocks, we reverse engineered the placement algorithm.

We found that the block scheduler uses a leftover policy. In particular, the blocks for the first kernel are assigned to different SMs in a mostly round-robin manner. If there are SMs that are idle, or that have leftover capacity, they are used for blocks of the second kernel. Otherwise, the blocks of the second kernel are queued until at least one SM is released. Therefore, if each kernel is launched with a number of blocks more than number of SMs on the device, such that each block does not exhaust the resources of the SM, they achieve co-residency within an SM. In this case we can utilize shared resources on the same SM as a target of contention for covert channels.

We also consider a case where the two kernels cannot be co-located on the same SM, due to a small number of blocks leaving some SMs idle for the second kernel, or in cases where there is no leftover capacity on the SMs used by the first kernel. In this case, covert communication is still possible through contention on resources that are shared between all SMs such as the L2 cache.

4 Covert Channels on GPGPUs

In this section, we demonstrate a covert channel through contention on constant memory in GPUs. Although we can utilize other shared resources such as shared memory, we selected constant memory because the size of both the L1 and L2 caches is small allowing us to create contention easily. The same attack applies to other parts of the memory hierarchy. In addition, we could also use contention on other shared structures such as the the processing cores and the register file.

The attack proceeds in two steps. First, an offline step uses the microbenchmarking approach introduced by Wong et al. [22] to deduce constant memory and cache characteristics at each level of the hierarchy. The second step is the communication step where we use contention to create the covert channel. We present results from experiments on a Kepler GPU (Tesla K40C). However, we implemented the experiment also on a Fermi (Tesla C2075) and a Maxwell (Quadro M4000) GPU with similar results.

4.1 Attack Step I: Offline Characterization of Constant Memory Hierarchy

The parameters of the constant memory hierarchy and each level of the cache can be extracted from latency measurement of loading different size array from constant memory using a strided access pattern. The cache is first warmed by reading the array, then the array is read again while timing the accesses [22]. As long as the array fits in the cache, latency remains constant. As the size of the array is increased, we observe when the latency changes, due to capacity related cache line misses, which provides us with the information about the cache structure as described below.

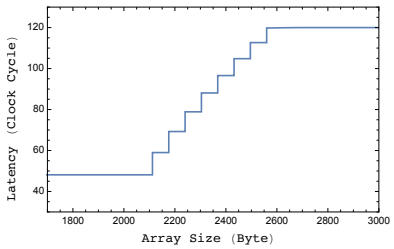

Fig. 2. L1 constant memory cache, stride value is 64 bytes.

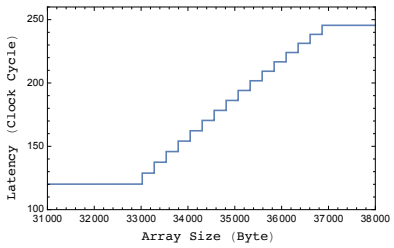

Fig. 3. L2 constant memory cache, stride value is 256 bytes.

Figure 2 and Figure 3 show the latency measurements for the L1 cache and L2 cache respectively. Each point on the figure represents an experiment with the array size shown on the x-axis. The number of steps in the figure is equal to the number of cache sets. The cache line size corresponds to the width of each

step. From the cache size, number of cache sets and cache line size, we can calculate the cache associativity. The configuration of the Kepler GPU constant memory caches is derived to be:

- L1 cache size is 2 kB organized as 8-set 4-way set associative cache, with each cache line size is 64 byte cache lines.
- L2 is a 32 kB cache organized as a 16 sets 8-way set associtative cache, with 256 byte line size.

4.2 Attack Step II: Intra SM covert channel through L1

We set up an experiment with two concurrent kernels using different streams on the GPU. On the Kepler K40C device, there are 15 SMs, so we consider 15 thread blocks (or more) for each kernel to be sure that they can have co-residency on the same SM.

The first kernel, which is the trojan, communicates by either creating contention or doing nothing to encode either 1 or 0 respectively. The second kernel, which is the spy, measures the latency of load operations from constant memory to distinguish what the trojan process is communicating: a high latency indicates 1, and a low latency indicates a 0.

More specifically, the covert channel communication operates as follows:

- The trojan creates contention or does nothing to send 1 or 0 to the first process respectively. To create contention on one set of the L1 cache, the trojan loads an array with the size of L1 cache (2 KByte, accessed with a stride of 512 to make the accesses hash into the same set) from constant memory.
- On the receiving side, the spy also loads a 2KByte array with the same stride as the trojan while timing the access.

Due to the concurrent execution of both kernels on the same SM, the access from the trojan process evicts the spy data and leads to cache misses. The spy can then detect communication by measuring the difference of latency (cache hit for 0, vs. cache miss/L2 access time for a 1). Note that we create contention over only a single set of the cache, rather than over the whole cache, reducing the memory traffic and accelerating the attack.

Our experiment results show that in the case of contention (i.e. sending 1), the measured latency by the spy is about 112 clock cycles, but without contention (i.e. sending 0), the latency is 49 clock cycles. This difference allows the spy to easily determine the bit being transmitted.

To communicate multiple bits, the trojan and the spy have to stay synchronized. To simplify this problem, we launch two kernels to communicate each bit of the message. Moreover, to ensure overlap between spy and trojan processes, we need to iterate sending each bit a sufficient number of times (20 times in our experiments). Clearly, this incurs some overhead to launch the kernels and to repeat the accesses, but it simplifies synchronization by leveraging the stream operations, resulting in error free communication with a bandwidth of around 30Kbps. In the future, we intend to explore synchronization through covert communication (on different sets of the cache). We believe that this will allow us to significantly improve the channel bandwidth.

We can exploit the parallelism available on the GPU to further increase the bandwidth of the covert channel. In particular, if we manage to colocate the trojan and the spy on multiple SMs, each of these instances can communicate independently using the approach described above. With 15 SMs available in Tesla K40C device, the trojan is able to send 15 bits simultaneously and the bandwidth of communication is increased 15 times. The measured bandwidth is about 430Kbps.

TABLE 1
Covert channel communication bandwidth on different GPU architectures

GPU	**L1** (one set)	**L1** (Parallel)	**L2** (one set)
Tesla C2075 (Fermi)	25 Kbps	330 Kbps	20 Kbps
Tesla K40C (Kepler)	30 Kbps	430 Kbps	23 Kbps
Quadro M4000 (Maxwell)	30 Kbps	380 Kbps	22 Kbps

4.3 Alternative Attack Step II: Inter SM covert channel through L2

When two concurrent kernels cannot be co-located on the same SM, they can still communicate through L2 constant cache that is shared between all SMs. The process of creating a covert channel is the same as the L1 channel. However, the parameters of the L2 cache are different: we consider array size of 32 kB and stride value of 4096 bytes (16 sets $\times$ 256 bytes) to fill just one cache set. The measured latency with contention on L2 is 246 clock cycles, while it is 122 clock cycles in the case of no contention. The measured bandwidth in this scenario is about 23Kbps.

We can create contention in parallel, such that each thread block loads a part of the array (corresponding to just one way) to fill the L2 cache collaboratively. This increases bandwidth about 8 times. Another possible scenario to exploit parallelism is using each thread block to communicate through one particular L2 cache set, enabling trojan to send 16 bits (number of L2 cache sets) simultaneously. However, cache traffic and bank collisions, as limiting factors should be considered and overcome, since L2 is a shared cache between all SMs.

Table 1 shows the covert channel communication bandwidth for different GPU architectures and the different scenarios of covert channel attacks that we described. To compare, we considered the same number of iterations for communicating each bit in different architectures. Clearly, high bandwidth covert channels are feasible on GPGPUs.

5 Related Work

Lee et al. [11] utilize information leakage that occurs due to not clearing newly-allocated memory in the GPU to extract rearranged webpage textures of Chromium and Firefox web browsers from NVIDIA and AMD GPUs. When multiple users share the same GPU, there is information leakage between concurrently executing processes or from recently terminated processes.

This vulnerability can be closed by clearing memory before it gets reallocated to a different application.

Mukherjee et al [13] utilize a power-based side-channel attack on a GPU to gain secret information from an AES implementation being accelerated by the GPU. The derived information includes the plain text and encryption key which are obtained through correlation between intermediate results produced by a cryptographic algorithm and power consumption values. Jiang et al. [9] conduct a timing side-channel attack. Due to inherent SIMT and memory coalescing behavior present on GPUs, they identify correlation between execution time and unique cache line requests. Observing the encryption

time, a small set of potential keys that result in this time are identified, allowing the correct key to be guessed by trying all the keys in this set. Although it is possible to use these side channels for covert communication, the bandwidth is likely to be significantly smaller than our attacks because in both, the leakage is measured outside of the chip.

Covert channels have been explored by many studies in the context of CPUs [6]. For an overview, an excellent recent study characterized the bandwidth of several CPU contention-based covert channels including cache based side channel attacks [7]. Chen et a. [3] proposed a framework to detect and mitigate timing covert channel on shared hardware resources on a CPU by monitoring conflicts; it is possible that similar techniques can be used to mitigate covert channels on the GPU. We believe that our paper is the first study to characterize GPU based covert channels.

Side-channel vulnerabilities are a dual of covert channel vulnerabilities where the leakage through the side channel is exploited by a spy to derive sensitive information from an executing victim; many CPU side channel attacks have been explored, including attacks on caches (e.g., [10], [12], [18]). Several defenses and mitigations for these attacks have been proposed (e.g., [5], [20]). Side channel attacks are dangerous but are more challenging to carry out than covert channel communication because the spy and the victim do not collude. Rather, the spy seeks to exploit any leakage from a victim to discover its sensitive data such as cryptographic keys. Our future work will explore side-channel attacks on GPGPUs.

Olson et al [16] developed a taxonomy of vulnerabilities and security threats for accelerator based on threat types (Confidentiality, Integrity and Availability) and risk categories (what part of accelerator they affect). In other work [17], they propose sandboxing accelerators to protect against vulnerabilities when the CPU and the accelerators share the same address space (e.g., under a Heterogeneous System Architecture configuration).

6 Conclusion

This paper demonstrates the feasibility of covert channel communication on GPGPUs. We explored how the communicating kernels can leverage the block scheduler to achieve co-residency. If they are able to achieve co-residency on the same SM, the SM local resources can be used as a side channel. If co-residency is only possible across SMs, then the inter-SM shared resources can be used for contention. We demonstrated two attacks on the L1 and the L2 constant caches on three generations of NViDia GPUs and took advantage of the GPU parallelism to increase the channel bandwidth.

We explored how to increase the bandwidth using the available parallelism on the GPGPU. Our future work will explore other strategies to further improve bandwidth of covert channels by improving the synchronization between the trojan and the spy and also consider communication in the presence of noise from other applications. We believe that other shared resources, such as shared memory, register file and data caches, or shared function units, can be utilized as covert channel or side channel to communicate secretly or extract secret information from critical workloads running on GPGPUs. Our future works will also include exploring the possibility of side channel attacks on GPGPUs, as well as mitigation techniques against covert- and side-channels.

Acknowledgment

This work is partially supported by US National Science Foundation grant CNS-1422401.

References

[1] A. Di Biagio, A. Barenghi, G. Agosta and G. Pelosi. *"Design of a Parallel AES for Graphic Hardware using the CUDA framework."* Proc. IEEE IPDPS 2009.

[2] Q. A. Chen, Z. Qian and Z. M. Mao. *"Peeking into your app without actually seeing it: Ui state inference and novel android attacks."* 23rd USENIX Security Symposium, 2014.

[3] J. Chen and G. Venkataramani. *"CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware."* Proc. MICRO 2014.

[4] R. Detomini, R. Lobato, R. Spolon and M. Cavenaghi. *"Using GPU to exploit parallelism on cryptography."* Information Systems and Technologies (CISTI), pp. 1-6, 2011.

[5] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side-channel attacks," in *ACM Transactions on Architecture and Code Optimization*, Jan. 2012.

[6] D. Evtyushkin, D Ponomarev and N. Abu-Ghazaleh. *"Understanding and mitigating covert channels through branch predictors"*. ACM Trans. on Architecture and Code Optimization, Jan. 2016.

[7] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath and M. Tiwari. *"Understanding contention-based channels and using them for defense."* International Symposium on High Performance Computer Architecture (HPCA), pp. 639-650, 2015.

[8] W. W. Hwu. *"GPU Computing Gems Emerald Edition."* Elsevier, 2011.

[9] Z. H. Jiang, Y. Fei and D. Kaeli. *"A complete key recovery timing attack on a GPU."* HPCA, 2016.

[10] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev and A. Jaleel. *"A High Resolution Side Channel Attack on Lower Level Caches."* Proc. ACM Design Automation Conference (DAC 2016).

[11] S. Lee, Y. Kim, J. Kim and J. Kim. *"Stealing Webpage Rendered on your Browser by Exploiting GPU Vulnerabilities."* Proc. of IEEE Symposium on Security and Privacy, pp. 19-33, 2014.

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

[13] S. Mukherjee, C. Luo, C. Finnegan, Y. Fei and D. Kaeli. *"Side Channel Attacks on GPUs."* BARC, 2015.

[14] N. Nishikawa, K. Iwai and T. Kurokawa. *"High-performance symmetric block ciphers on CUDA."* Networking and Computing (ICNC), pp. 221-227, 2011.

[15] *"NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110."* NVIDIA Whitepaper, 2012.

[16] L. Olson and E. Lena. *"Security Implication of Third-Party Accelerators."* Computer Architecture Letters, 2015.

[17] L. E. Olson, J. Power, M. D. Hill and D. A. Wood. *"Border Control: Sandboxing Accelerators."* Proc. MICRO, 2015.

[18] C. Percival. *"Cache missing for fun and profit."* 2005. Available from: http://www.daemonology.net/papers/htt.pdf.

[19] R. Di Pietro, F. Lombardi and A. Villani. *"CUDA leaks: Information Leakage in GPU Architecture."* arXiv:1305.7383v, 2013.

[20] Z. Wang and R. Lee, "A novel cache architecture with enhanced performance and security," in *Proc. International Symposium on Microarchitecture (MICRO)*, Dec. 2008.

[21] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang and M. Guo. *"Simultaneous Multikernel: Fine-grained Sharing of GPGPUs."* IEEE Computer Architecture Letters, 2015.

[22] H. Wong, M. M. Papadopoulou, M. Sadooghi and A. Moshovos. *"Demystifying GPU microarchitecture through microbenchmarking."* ISPASS, pp. 235-246, 2010.

[23] Q. Xu, H. Jeon, K. Kim, W. W. Ro and M. Annavaram. *"Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming."* The 43rd Internaltional Symposium on Computer Architecture (ISCA), 2016.