Forensics on GPU Coprocessing in Databases – Research Challenges, First Experiments, and Countermeasures

Sebastian Breß, Stefan Kiltz, Martin Schäler

Faculty of Computer Science
Otto-von-Guericke University Magdeburg, Germany
{bress,kiltz,schaeler}@iti.cs.uni-magdeburg.de

Abstract: Recently, using GPUs for coprocessing in database systems has been shown to be beneficial. However, information systems processing confidential data cannot benefit from GPU acceleration yet because knowledge of security issues and forensic-examinations on GPUs are still fragmentary. In this paper, we point out key challenges and research questions related to forensics and anti-forensics on GPUs. Our results and discussion are based on analogies from similar computation environments, and experiences. Initial experimental studies indicate that data in GPU RAM is retrievable by other processes. This can be done by creating a memory dump of device memory. Hence, application data is accessible by users without access permissions, by bypassing the access control system of the database management system. Finally, we discuss approaches, how our results can be used in forensic and anti-forensic scenarios.

1 Motivation

Nowadays, information systems process an increasing amount of personal data, which has to be protected for unauthorized access. The core parts of information systems typically consist of one or more database systems, because they (1) store the application data and (2) handle efficient query processing on the data. Since database systems need to fulfill application's performance requirements, a lot of work focuses on increasing performance of database systems. Recent research focus on accelerating query processing of database systems using *Graphics Processing Units* (GPUs) as coprocessor [BS10, HLY+09, HYF+08, WWN+10]. GPUs are used to accelerate query processing [AW10, BS10, DWL+12, GGKM06, HLY+09, HYF+08, HY11, KLMV12, Pir12, PMK11, PSMK12, LDKA10] as well as query optimization [HM12].

Though GPU acceleration provides significant speedups [HLY+09], some systems only benefit from GPU coprocessing if and only if the confidentiality of data is ensured, e.g., in a scenario of bank accounts or fingerprint databases. GPUs have highly parallel computing capabilities with their own private memory and specialized programming interfaces (e.g., the proprietary CUDA [NVI12a]). A user *Mal* may access data of user *Alice* without permission, because multiple programs of different users can utilize the GPU at the same time. To verify this hypotheses, we investigate whether security protocols of a database system could be bypassed if no additional security measures are taken. In this paper, we take a

first step to allow forensic examination on GPUs as well as preventing users/programs to access data, which they must not access.

Beebe identified the need for forensic methods for "Non-Standard Computing Environments" such as mobile devices, cloud computing, and virtualization [Bee09]. A GPU is a non-standard environment for computations. However, GPU acceleration is not only database specific. Most of the research on General Purpose Computing using Graphics Processing Units (GPGPU) is done for different scenarios. For instance, see the survey of Owens et al. on GPGPU [OLG+07].

We expect that data with access restrictions will be processed on GPUs in the near future. This data may be the purpose of forensic investigation, or shall be removed from the GPU for privacy reasons in a forensically secure way. Additionally, direct access to GPUs may inflict additional security problems, e.g., because of programming or design errors. Approaches already exist, which utilize the GPU for accelerating forensic analysis, e.g., [JB06, MRR07]. However, to the best of our knowledge, there is no noticeable work investigating forensic analysis on GPUs. One of the core challenges is the closed nature of APIs that allow the direct access to the GPU. Hence, we cannot analyze the code statically to prove our assumption that data of previous processes is retrievable. Nevertheless, we can analyze the APIs themselves and derive conclusions. The contributions of the paper are as follows:

1. Raise attention and initiate discussion on forensic examinations of GPUs.
2. State a list of relevant research questions, underlying hypotheses, and respective experimental set-up and methodologies to prove or reject these hypotheses.
3. Perform first experiments and report lessons learned that are valuable for future research.
4. Furthermore, we suggest an approach to counter our discovered security threat.

The paper is structured as follows. In Section 2, we discuss preliminary work, such as background on GPUs, requirements for forensic investigations and our forensic model. We introduce challenges for forensic and anti-forensic when an application utilizes GPU coprocessing in Section 3. We conduct experimental evaluation of the most important challenges of forensic and anti-forensic in Section 4. Section 5 discusses the consequences of our experimental results for forensic and anti-forensic. Since we are not aware of any work on forensics of GPUs, we provide a more general overview of approaches that utilize the GPU for forensic examinations in Section 6. The paper closes with future work in Section 7 and a conclusion in Section 8.

2 Preliminary Considerations

In this Section, we discuss preliminary work. First, we provide background over GPUs. Second, we discuss legal and IT-forensic requirements. Finally, we provide a brief summary of our model of the forensic process in main memory investigation.

2.1 Graphics Processing Units

GPUs are specialized processors, which were initially designed to support graphical applications. Due to their parallel nature, they can be also used for general purpose computation using frameworks such as CUDA [NVI12a], OpenCL[1], or AMD APP SDK[2]. The most important properties of GPUs are: (1) They have more cores than CPUs and a significantly higher computing power than CPUs with the same price. (2) They target high throughput by performing parallel execution of tasks using multi-threading with thousands to ten-thousands of threads. (3) They have a dedicated memory, the GPU RAM (4). GPUs are optimized for computation with high arithmetic density. On the downside, a lot of control flow brakes the performance of a GPU kernel [SK10]. The Compute Unified Device Architecture (CUDA) is a framework for GPGPU. It provides a C interface and enables developers to write their parallel application code running on the GPU in CUDA C, an extension of the C programming language [NVI12a]. The GPU stores and processes data, which may contain relevant information for a forensic examination (e.g., screen content, application data). Data processing, especially if personal data is concerned, and forensic methods in particular need to meet a number of requirements. An exemplary chosen selection is outlined in the following.

2.2 Legal and IT-Forensic Requirements

Systems that operate on person-related data have to adhere to data protection legislation in most countries, such as the Data Protection Act 1998[3]. In order to assure data protection, suitable measures that ensure the security aspect of confidentiality [Bis02] and a working rights management have to be in place. This, in turn, means that any new development in DBMS, such as the proposed GPU coprocessing, has to provide means to ensure that data protection requirements are met.
For electronic evidence to be accepted in a court of law, they have to pass a test of admission. In the United States, the judge decides about the admissibility of evidence based on the Daubert challenge. The scientific validity of a technique or method is judged using originally five Daubert factors [DG01]:

- Peer review and publication,
- General acceptance in the relevant expert community,
- Potential for testing or actual testing,
- Known or potential rate of error,
- Existence and maintenance of standards controlling the use of the technique or method.

[1] http://www.khronos.org/opencl/
[2] http://developer.amd.com/tools/hc/appmathlibs/Pages/default.aspx
[3] http://www.legislation.gov.uk/ukpga/1998/29

Furthermore, for electronic evidence to be accepted, a comprehensive documentation (e.g., by using container formats such as [CGS09]) is mandatory. The concept of the Cyber Forensic Assurance (CFA) [Dar10] enforces this further by assembling *information security*, *information assurance*, the *Parkerian Hexad* and *information quality* into four main considerations and eight components [Dar10]. Component I contains Confidentiality and Possession, Component II of Integrity and Authenticity, Component III of Availability and Utility, and Component IV of Completeness and Accuracy.

The new approaches outlined in this article should integrate those considerations from inception to application in the field.

2.3 Models of the Forensic Process and Main Memory Investigation

To ensure a systematic course of action during a forensic investigation and that a maximum of case-relevant data is addressed, we have to use a model for forensic processes. Several models for digital forensics exist, see [Pol07] for a cursory overview. Kiltz et al. proposed a model that is already applied in main memory forensics and consists of three major components [KHDV09]:

1. *Phases* - grouping of investigation steps into 6 categories:
   - *Strategic preparation SP*
   - *Operational preparation OP*
   - *Data gathering DG*
   - *Data investigation DI*
   - *Data analysis DA*
   - *Documentation DO*
2. *Classes of methods* - grouping of forensic methods into 6 categories:
   - *Operating system OS*
   - *File system FS*
   - *Explicit means of intrusion detection EMID*
   - *IT-application ITA*
   - *Scaling of methods for evidence gathering SMG*
   - *Data processing and evaluation DPE*
3. *Data types* - differentiation of forensically relevant content into 8 layers:
   - *Hardware data* DT_1
   - *Raw data* DT_2
   - *Details about data* DT_3

- *Configuration data* DT_4
- *Communication protocol data* DT_5
- *Process data* DT_6
- *Session data* DT_7
- *User data* DT_8

The idea is to capture the forensic process in a comprehensive way. The central concept is that the system to be investigated is seen as a data source containing forensically relevant information in its data. *Methods* from the classes of methods (e.g., data processing and evaluation, DPE) process this data. The data is layered into the *data types*, targeting a particular data type and transforming it (typically in a more abstract layer, e.g., from raw data DT_1 to process data DT_6), depending on the particular *phase* (e.g., data investigation DI). This model also supports finding new means of locating, acquiring, investigating and analyzing forensically relevant data. This is done by providing a structured way of looking at data sources and methods to process them in the context of the forensic workflow. One such new opportunity to gain access to data sources and process them in a forensically sound way is outlined in this article in the case of the GPU-subsystem, which is suggested to act as a coprocessor to DBMS. In this case, access to the video memory stream managed by the GPU is needed for forensic investigations and access protection against IT-security related incidents. When looking for existing forensic tools or create entirely new ones, this categorization helps to identify the sources and where the methods provided reside. For instance, when trying to gather evidence from systems with no strategic preparation (i.e., no additional forensic methods are available, e.g., explicit means of intrusion detection EMID) the forensic expert is left with what the operating system OS, the file system FS and maybe the IT-application ITA may have gathered. Depending on the phase of the examination, not evaluated data is irretrievably lost (e.g., in the case of data gathering DG) or not taken into account (e.g., in data investigation DI or data analysis DA, violating the demand for completeness, see [Dar10]).

3 Challenges for Database Coprocessing and Security

In this Section, we discuss challenges for secure database coprocessing as well as for GPU Forensics.

3.1 Challenges for Secure Coprocessing

If a database systems utilizes coprocessing techniques, then it might be exposed to potential IT-security risks by circumventing access controls of a DBMS as demanded by Codd's rules [Cod82], or violating the IT security aspects (Confidentiality, Integrity, Availability, Authenticity, Non-Repudiation). To ensure secure GPU coprocessing, we identify the following challenges:

Enforcing Security Protocols: Can the access control component of a DBMS be bypassed to access data without the necessary access rights?

Ensuring Data Integrity: Can confidential data be secretly modified without using the interfaces of a database system?

Secure Processing: How can a GPU be utilized as crypto-processor using known techniques?

To enable database systems, which process confidential data, to benefit from GPU acceleration, the identified challenges have to be addressed.

3.2 Challenges for GPU Forensics

New forensic methods are required to acquire, examine, and analyze data processed on the GPU. Therefore, a detailed research of the GPU memory and proprietary APIs is necessary. To illuminate an incident, it is crucial to have knowledge about recoverability of data, e.g., under which conditions can application data still be accessed and examined. If we want to reconstruct the program flow, we have to be able to interpret the memory of the GPU. Therefore, we have to identify the GPU kernels and program stack as well as internal structures of a GPGPU framework API. Using this considerations, we may be able to reconstruct the program flow. Finally, it is desirable to have a similar tool support for GPU RAM forensic as for CPU RAM forensic, e.g., *Memoryze*[4] or the *Volatility Framework*[5]. Furthermore, a GPU could be used as crypto-processor, which leaves no discernible traces in main memory, resulting in a challenge for IT-forensics. In summary, we identify the following challenges to allow forensic investigations on GPUs:

Recoverability of Data: Under which circumstances is application data still dormant in GPU RAM, e.g., after deallocating memory or rebooting the machine?

Memory Interpretation: How can an acquired memory dump be efficiently interpreted?

- **GPU Kernel Identification:** Can we locate the GPU kernel in the GPU RAM?
- **Accessibility of Internal Structures:** Are internal CUDA/OpenCL/. . . structures accessible?

Reconstructibility of Program Flow: Can we reconstruct the program flow of an executed GPU kernel?

[4] `http://www.mandiant.com/resources/download/memoryze`
[5] `https://www.volatilesystems.com/default/volatility`

4 Experimental Studies

In this Section, we describe the experiments we conducted and present our results for two common CPU main memory forensic techniques applied on GPU device memory.

4.1 First Steps Towards the Acquisition and Examination of GPU Memory

In our experiments, we have to ensure a minimally invasive procedure. That means, we have to avoid modification of the object of investigation for reasons of data integrity, in this case the GPUs device memory. To investigate the GPU device memory, we have to use a GPGPU framework API to access the data in GPU RAM. We selected CUDA, because it is currently the most widely used GPGPU framework for database coprocessing, e.g., [BS10, HLY+09, HYF+08, HY11, LDKA10, MHS+11].

Since the CUDA API is closed source, we cannot verify, which changes the calls to the CUDA API cause in the GPU RAM. We can only derive assumptions from the documentation stating that the used API functions do not modify the GPUs device memory. In further research, however, this assumption should be verified. For all experiments, we used the following CUDA functions: **cudaMalloc**, **cudaMemcpy**, **cudaFree**, and **cudaMemGetInfo**.

cudaMalloc allocates a user specified number of bytes in the global GPU RAM [NVI12b]. **cudaMemcpy** copies data from CPU RAM to GPU RAM or vice versa. Note that when data is copied from non page-locked host memory, the data is first copied in page-locked memory in the CPU RAM and is then transferred to GPU RAM[6] [SK10]. The additional copy operation in CPU RAM may alter data relevant for a forensic investigation. **cudaFree** deallocates memory that was allocated by using **cudaMalloc**. **cudaMemGetInfo** provides information about the size of the currently available and total memory of the GPU RAM. The official CUDA documentation provides further details on the used CUDA API functions [NVI12b].

Note that since we are not concerned about forensic investigations in main memory, we omit the respective considerations in this paper. However, in a real world scenario, one has to consider main memory and GPU RAM forensics while also ensuring *integrity* and *authenticity* for the object of investigation in a heterogeneous system together with a comprehensive process accompanying documentation.

To ensure generality of our observations, we conducted the experiments on two machines with NVIDIA GPUs of different generations: (1) on a machine with an NVIDIA GeForce 9600 GT (compute capability 1.0) and Ubuntu 10.04 (32 bit) operating system and (2) on a machine with a NVIDIA GeForce GT 640 (compute capability 2.1) and Ubuntu 12.04 (32 bit) operating system.

[6] Usually, application data is stored in non page-locked host memory.

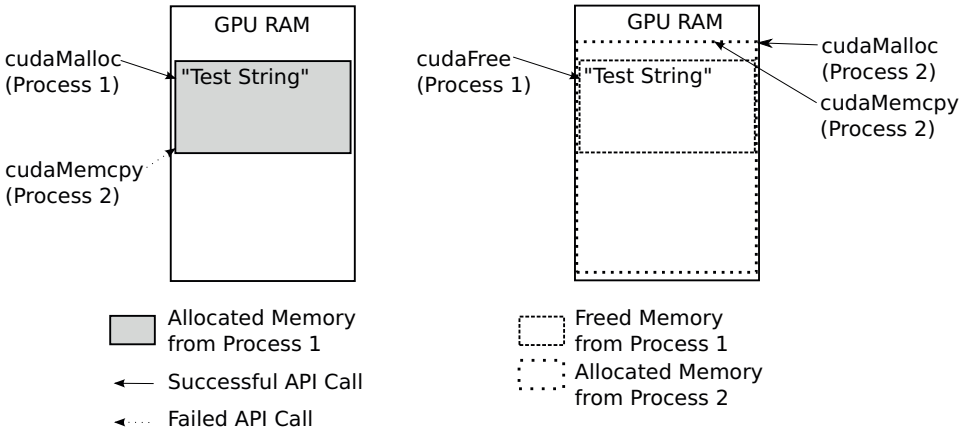

Figure 1: Experimental Results

4.2 Experiments

We now discuss our experiments in detail. We address the main challenges for forensics and anti-forensics. Therefore, we evaluate the challenges **Enforcing Security Protocols** (anti-forensics) from Section 3.1 and **Recoverability of Data** (forensics) from Section 3.2. Note that both challenges complement each other. That means, if security protocols are enforced, recovery of data from GPU RAM is not possible and vice versa. Therefore, our experiments targets the recoverability of data from GPU RAM. In our first experiment, we test whether it is possible to access memory that is not currently allocated by a program. In the second experiment, we examine whether data from terminated programs is accessible by a program. Finally, we apply our forensic model from Section 2.3 to the GPU RAM, because we have to verify to which degree our methods fulfill the legal requirements for forensic workflows.

4.2.1 Accessing non Allocated Memory in GPU RAM

In this experiment we investigate whether it is possible to access GPU RAM, which was previously allocated by **cudaMalloc** and afterwards freed by **cudaFree**.

Our program first allocates memory in the GPU RAM by using **cudaMalloc**. Afterwards, it copies a fixed user defined string in the GPU RAM by using **cudaMemcpy**. To be able to assert that the string was correctly written in GPU RAM, the program copies the data back to the CPU RAM by using **cudaMemcpy** afterwards. Since **cudaFree** sets a device pointer to zero, we copy the device pointer, which contains the address of the user defined string in GPU RAM, and name it *sneaky pointer*. Afterwards, we call delete on the device pointer acquired by the previous call of **cudaMalloc**. We then use *sneaky pointer* as an argument of **cudaMemcpy** to access memory that no longer belongs to the program. As a result, the **cudaMemcpy** function returned with an error code, which states an invalid

parameter values was passed. We repeated the experiment and made the same observations on both test machines. Figure 1 visualizes our results.

Apparently, CUDA has a memory protection mechanism, which does not permit the user to access currently allocated memory from another program. Furthermore, we conclude that it is not possible to access data in non allocated memory.

4.2.2 Creating GPU RAM Dump of Currently Free Memory

In this experiment, we investigate whether it is possible to access data in GPU RAM from previous executed programs, which freed all used GPU RAM correctly by calling **cudaFree**.

There are two users on our test systems, *Alice* and *Mal*. *Alice* executes a program, which writes a fixed user specified string to GPU RAM. After the program terminated, *Mal* executes a program, which computes the available GPU RAM by calling **cudaMemGetInfo** and then allocates the available memory[7]. Afterwards, the allocated memory is copied from GPU RAM to CPU RAM by using **cudaMemcpy**. Then, the program searches in the acquired memory dump for the string previously written by the program that *Alice* executed.

We repeated this experiment 1000 times with strings of different lengths on the test machines. *Mal*'s program found the string written by *Alice*'s program in all cases. We used the secure password generator of openssl[8] to generate the random strings and varied their lengths between 10 and 10000 characters in steps of 10. Note, that the user *Alice* used program A to write a string into GPU RAM, whereas user *Mal* used program B to create a memory dump. Both users had different user accounts, and none of the users had root access. Hence, we conclude that program B can extract data written to GPU RAM by a prior executed program A, even for different user accounts without root access. Therefore, we found a severe security issue for GPU coprocessing in applications in general and, therefore, for GPU accelerated database systems. This observation can be utilized for forensic investigations, as well.

4.3 Application of Forensic Model

With regards to the model of the forensic process introduced in previous work [KHDV09], the GPU RAM represents a subcomponent of the main memory. The GPU RAM shares the main characteristics of CPU RAM as a source of highly volatile data whose content is easily lost (e.g., due to a loss of power) and easily altered (e.g., by the re-use of that memory after deallocation). The described combination of **cudaMemGetInfo** and **cudaMemcpy** represents two phases. The first one is *data gathering DG* for the creation of the dump by creating *raw data* DT_1 using *forensic methods* supplied by the *operating system OS*.

[7] We have to subtract a small amount of memory (around 1MB), because the CUDA runtime does not permit to allocate the complete free space.

[8] `http://www.openssl.org/docs/apps/openssl.html`

The second part of the experiment, the string search, represents the phase of *data investigation DI* with respect to the text string search by using the *forensic method* supplied by the *data presentation and evaluation DPE*, yielding *user data* DT_8. However, those techniques used in the first experiments would not qualify as forensic methods according to the demands placed on forensic examinations (e.g., by the CFA [Dar10], see also Section 2.2) by, e.g., not assuring integrity and authenticity. The first results outline the need for further research, which in turn can be used to create forensic methods that do comply with the requirements of a forensic examination.

5 Discussion

As observed in Section 4, it is possible to create a memory dump of the available space from the GPU RAM. Furthermore, user privileges/access control can be bypassed and data processed by previous executed programs can be recovered. In this Section, we discuss the consequences of our evaluation results from two points of view. First, we treat our results as security threat for database systems, which needs to be addressed to prevent unauthorized data access. Second, we discuss our results from the view of forensic investigations to analyze incidents.

5.1 Secure Coprocessing – Anti-Forensic

Using the results of our experiments, we can conclude that DBMS access control can be bypassed for GPU coprocessing scenarios. This is because data managed by a DBMS can be retrieved by processes from other users without necessary access rights. Therefore, we need a special free function that deletes data forensically secure to address the **Recoverability of Data** challenge from Section 3.1.

We use the information provided by Gutmann [Gut01] to justify our procedures and thus to avoid the extraction of possible confidential data from GPU RAM by other programs. The main memory in general and the memory accessed by the GPU typically consists of a large array of DRAM memory cells with only the latest data fed to an internal latch (cache) of SRAM. Since the cache outputs the requested data, the retention effect of stored data inside DRAM-only cells can be neglected. There is little chance of any piece of data except the last one read DRAM accesses remaining in those latches for more than an instant, because these latches are shared across the entire DRAM [Gut01]. Thus, writing random content or zeros (for easier fabrication) should be sufficient to erase the data previously contained in a given section of main memory in general and GPU memory, in particular[9]. Therefore, we propose a secure **cudaFree** function, which first writes random values or zeros to the memory area, which is to be freed and then calls **cudaFree**. Using this approach, one can effectively avoid data extraction by other programs. Due to the parallel nature of GPUs, the GPU RAM can be concurrently overwritten using thousands or ten-thousands of threads.

[9] Additional measures in a DRAM-only scenario are outlined by Gutmann [Gut01]

As the goal is the destruction of confidential data, a synchronization between threads is not necessary. Hence, data in GPU RAM can be overwritten with lower latency than in CPU RAM.

5.2 Law Enforcement – Forensic Methods

Our results also supports forensic investigations. If an application uses the GPU to accelerate computations, it is likely that the object of investigation can be located in the GPU RAM. Hence, research is necessary as to, which methods can be applied to analyze incidents. As we discovered, one method is simply a creation of a GPU RAM dump. As long as the application of interest has freed its used GPU RAM, it is possible to recover its data processed on the GPU. However, this approach is limited, because we are still unable to investigate memory that is still allocated by the application of interest. Hence, we successfully addressed the **Recoverability of Data** challenge from Section 3.2.

Concerning the experiments regarding the GPU subsystem and in reference to the Cyber Forensic Assurance [Dar10] it can be noted that so far, only the component III (b), "Utility/Relevance - Is it useful/is it the right information?" could partly be shown. In order to become an accepted part of the sets of forensic methods, all the other components need to be addressed, outlining the next steps towards the next necessary research.

One scenario for forensic investigation of GPU RAM arises, when the GPU is utilized by an attacker to intrude in a system, e.g., by using a GPU accelerated password cracker such as John the Ripper[10]. To be able to collect all evidence relevant to an investigation, the GPU RAM has to be considered. Our findings on GPU RAM investigation are a first step to include the GPU in forensic investigations.

6 Related Work

One example for software capable of CPU RAM forensics is the Forensic Analysis Toolkit (FAT-Kit) [PWFA06]. FAT-Kit is designed to support a forensic examination and assumes that the phase of data gathering (i.e., the acquisition of raw data) is already finished and a low-level image of the main memory already exists. FAT-kit, amongst others, aims at identifying structures from the C-programming language and the re-assembly of known address spaces and visualizing the informations that are extracted.

To the best of our knowledge, there is no noticeable previous work on forensic investigations on GPU RAM in the context of database coprocessing on GPUs. Hence, no tool exists, which can fulfill the legal requirements for a fully fledged forensic workflow for GPU RAM.

However, approaches exist for accelerating forensic analysis using GPUs. Jacob and Brodley presented an approach that outsources computations of an intrusion detection system

[10] `http://www.openwall.com/john/`

[JB06]. Marziale et al. investigated GPU accelerating of forensic tools in general and found that GPU coprocessing is beneficial for binary string search, an important operation for forensic investigations [MRR07].

7 Future Work

We only considered the global device memory in this paper. Our investigation has to be extended to the recoverability of data in different GPU memories such as local, constant, texture or shared memory, which are frequently utilized to further improve the performance of GPU algorithms [SK10]. Furthermore, we plan to repeat our evaluation using other GPGPU frameworks such as OpenCL, AMD APP SDK or abstractions of CUDA, such as Thrust[11]. Additionally, we investigate the applicability of other forensic methods for CPU RAM on GPU RAM, e.g., complete and forensically sound data gathering [Sch08], automated interpretation of data structures [PWFA06], and address space reconstruction [PWFA06].

8 Conclusion

In this paper, we addressed the problem of secure data processing for General Purpose Computing using Graphics Processing Units (GPGPU) in the context of database systems. We identified challenges for secure data processing as well as forensic investigations if the GPU is used as coprocessor in database systems. We discovered that application data can be easily accessed by other programs as soon as the allocated GPU memory is freed by an application. However, we did not discover a method to investigate application data in GPU RAM that is still allocated by the application. Furthermore, we discuss how our results can be used from a forensic and an anti-forensic point of view. Note that our results are not limited to database systems, but are applicable to all applications using CUDA C.

Acknowledgements

The work in this paper has been funded by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract No. FKZ: 13N10817 and 13N10818. We thank Ingolf Geist and Siba Mohammad for helpful feedback and discussions as well as the reviewers for their constructive criticism.

[11] `http://thrust.github.com/`

References

[AW10] Witold Andrzejewski and Robert Wrembel. GPU-WAH: Applying GPUs to Compressing Bitmap Indexes with Word Aligned Hybrid. In *DEXA (2)*, pages 315–329, 2010.

[Bee09] Nicole Beebe. Digital Forensic Research: The Good, the Bad and the Unaddressed. In *Advances in Digital Forensics V*, volume 306 of *IFIP Advances in Information and Communication Technology*, pages 17–36. Springer, 2009.

[Bis02] Matthew Bishop. *The Art and Science of Computer Security*. Addison-Wesley, 2002.

[BS10] Peter Bakkum and Kevin Skadron. Accelerating SQL database operations on a GPU with CUDA. In *GPGPU*, pages 94–103. ACM, 2010.

[CGS09] Michael Cohen, Simson Garfinkel, and Bradley Schatz. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. *Digit. Investig.*, 6:S57–S68, 2009.

[Cod82] E. F. Codd. Relational database: a practical foundation for productivity. *Commun. ACM*, 25(2):109–117, 1982.

[Dar10] Glen S. Dardick. Cyber Forensics Assurance. In Andrew Woodward, editor, *Proceedings of the 8th Australian Digital Forensics Conference*, pages 57–64. School of Computer and Information Science, Edith Cowan University, Perth, Western Australia,, 2010.

[DG01] Lloyd Dixon and Brian Gill. *Changes in the Standards for Admitting Expert Evidence in Federal Civil Cases Since the Daubert Decision*. RAND Institute for Civil Justice, 2001. ISBN: 0-8330-3088-4.

[DWL+12] Gregory Diamos, Haicheng Wu, Ashwin Lele, Jin Wang, and Sudhakar Yalamanchili. Efficient Relational Algebra Algorithms and Data Structures for GPU. Technical report, Center for Experimental Research in Computer Systems (CERS), 2012.

[GGKM06] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management. In *SIGMOD*, pages 325–336. ACM, 2006.

[Gut01] Peter Gutmann. Data remanence in semiconductor devices. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 4–4, Berkeley, CA, USA, 2001. USENIX Association.

[HLY+09] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Query Coprocessing on Graphics Processors. *ACM Trans. Database Syst.*, 34:21:1–21:39, 2009.

[HM12] Max Heimel and Volker Markl. A First Step Towards GPU-assisted Query Optimization. In *ADMS*, 2012.

[HY11] Bingsheng He and Jeffrey Xu Yu. High-Throughput Transaction Executions on Graphics Processors. *PVLDB*, 4(5):314–325, 2011.

[HYF+08] Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga Govindaraju, Qiong Luo, and Pedro Sander. Relational Joins on Graphics Processors. In *SIGMOD*, pages 511–524. ACM, 2008.

[JB06] Nigel Jacob and Carla Brodley. Offloading IDS Computation to the GPU. In *ACSAC*, pages 371–380. IEEE, 2006.

[KHDV09] Stefan Kiltz, Tobias Hoppe, Jana Dittmann, and Claus Vielhauer. Video surveillance: A new forensic model for the forensically sound retrival of picture content off a memory dump. In Stefan Fischer, Erik Maehle, and Ruediger Reischuk, editors, *Proceedings of Informatik2009 - Digitale Multimedia-Forensik*, pages 1619–1633, 2009.

[KLMV12] Tim Kaldewey, Guy Lohman, Rene Mueller, and Peter Volk. GPU Join Processing Revisited. In *DaMoN*, pages 55–62. ACM, 2012.

[LDKA10] Tobias Lauer, Amitava Datta, Zurab Khadikov, and Christoffer Anselm. Exploring Graphics Processing Units as Parallel Coprocessors for Online Aggregation. In *DOLAP*, pages 77–84. ACM, 2010.

[MHS+11] Roger Moussalli, Robert Halstead, Mariam Salloum, Walid Najjar, and Vassilis J. Tsotras. Efficient XML Path Filtering Using GPUs. In *ADMS*, 2011.

[MRR07] Lodovico Marziale, Golden G. Richard, III, and Vassil Roussev. Massive threading: Using GPUs to increase the performance of digital forensics tools. *Digit. Investig.*, 4:73–81, 2007.

[NVI12a] NVIDIA. NVIDIA CUDA C Programming Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`, 2012. pp. 30–34, Version 4.0, [Online; accessed 1-May-2012].

[NVI12b] NVIDIA. NVIDIA CUDA Library Documentation. `http://www.clear.rice.edu/comp422/resources/cuda/html/index.html`, 2012. Version 4.0, [Online; accessed 30-October-2012].

[OLG+07] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[Pir12] Holger Pirk. Efficient Cross-Device Query Processing. *Proceedings of the VLDB Endowment*, 2012.

[PMK11] Holger Pirk, Stefan Manegold, and Martin Kersten. Accelerating Foreign-Key Joins using Asymmetric Memory Channels. In *ADMS*, pages 585–597. VLDB Endowment, 2011.

[Pol07] Mark M. Pollitt. An Ad Hoc Review of Digital Forensic Models. In *Systematic Approaches to Digital Forensic Engineering, 2007. SADFE 2007. Second International Workshop on*, pages 43–54, 2007.

[PSMK12] Holger Pirk, Thibault Sellam, Stefan Manegold, and Martin Kersten. X-Device Query Processing by Bitwise Distribution. In *DaMoN*, pages 48–54. ACM, 2012.

[PWFA06] Nick L. Petroni, Jr., Aaron Walters, Timothy Fraser, and William A. Arbaugh. FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digit. Investig.*, 3(4):197–210, December 2006.

[Sch08] Andreas Schuster. The impact of Microsoft Windows pool allocation strategies on memory forensics. *Digital Investigation*, 5, Supplement(0):58–64, 2008. The Proceedings of the Eighth Annual DFRWS Conference.

[SK10] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 1st edition, 2010.

[WWN+10] Slawomir Walkowiak, Konrad Wawruch, Marita Nowotka, Lukasz Ligowski, and Witold Rudnicki. Exploring Utilisation of GPU for Database Applications. *Procedia Computer Science*, 1(1):505–513, 2010.