CloudNet Anti-Malware Engine: GPU-Accelerated Network Monitoring for Cloud Services

George Hatzivasilis[1[*]], Konstantinos Fysarakis[2], Ioannis Askoxylakis[1] and Alexander Bilanakos[3]

[1] Institute of Computer Science, Foundation for Research and Technology – Hellas (FORTH), N. Plastira 100, 70013 Heraklion, Crete, Greece
[2] Sphynx Technology Solutions, Gubelstr 12, 6300 Zuk, Switzerland
[3] Dept. of Computer Science, University of Crete, Voutes University Campus, 70013 Heraklion, Crete, Greece
hatzivas@ics.forth.gr

Abstract. In the modern applications for Internet-of-Things (IoT) and Cyber-Physical Systems (CPSs) heterogeneous embedded devices exchange high volumes of data. Interconnection with cloud services is becoming popular. Thus, enhanced security is imperative but network monitoring is computational intensive. Parallel programming utilizing Graphics Processing Units (GPUs) is a well-tried practice for drastically reducing the computation time in computation intensive domains. This paper presents CloudNet – a lightweight and efficient GPU-accelerated anti-malware engine, utilizing the CUDA General Purpose GPU (GPGPU). The core of the system computes the digests of files using a CUDA-optimized SHA-3 hashing mechanism. Malware digests are stored in a data structure so that detection checks take place as network traffic is processed. Work includes a comparative analysis for three types of data structures (hash table, tree, and array) to identify the most appropriate for this specific field. We develop several versions of two basic variations of applications, including performance comparisons of GPU-accelerated implementation to the reference and optimized CPU implementations. The CloudNet is developed in order to protect CPSs that communicate information to the industrial cloud. A trace of an industrial wind park traffic is utilized for the evaluation of CloudNet, achieving two times faster network monitoring than typical CPU solutions.

Keywords: cloud, industrial cloud, network monitoring, anti-malware, parallel computing, GPU, CUDA, SHA-3, CPS, IoT, IIoT.

1 Introduction

Most users nowadays will not neglect to safeguard their computers and laptops from all types of malicious software. However, the same approach is not always adopted for the devices in the domains of Internet of Things (IoT) and Cyber-Physical Systems (CPSs) which exchange information with the cloud [1], [2]. Thus, malicious entities can find a new opportunity in attacking these less protected components and

infiltrate in the whole network [3]. Drawbacks are mainly occurred due to [3], [4]: i) poor or non-existent security at the device-end, ii) poor network segmentation where the devices are directly exposed to the Internet, iii) unneeded functionality that is left in based on generic development processes, and iv) the use of the default credentials, which are often hard coded. So, the network monitoring and the inspection of traffic form the device to the cloud is becoming imperative along with the development of new efficient and scalable solutions which can tackle the processing of high volumes of data that are produced by such systems.

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to process data rapidly on a graphics card. At first, GPUs were deployed for processing images in high volume but later their highly parallel computing capabilities were exploited by a plethora of applications where computing intensive operations had to be performed.

To this effect, NVIDIA [5] offers the Compute Unified Device Architecture (CUDA, [6]) – a parallel computing platform and programming model. Through the use of CUDA, a graphics card can be utilized as a general purpose processing unit; a concept known as General Purpose GPU (GPGPU) computing. Thus, any application can potentially exploit the processing capabilities and highly parallel nature of modern graphics card to achieve significant speed-up factors compared to their CPU-only versions.

Several security sensitive applications require to process data at real-time and in high volumes. Such applications include anti-malware systems that monitor the network traffic or scan the hard disk of a PC to find out malicious data.

In this paper, an efficient and lightweight GPU accelerator for hashing and hash lookup is presented, called CloudNet. The proposed system utilizes an optimized implementation of SHA-3 [7] in GPUs, acting as the core hash function for calculating the digest of examined malwares. Then, we develop and compare several data structures for maintaining the processed data and select the most appropriate of them. We apply the proposed solution in security-related applications for anti-malware detection and real-time malware intrusion detection in network traffic.

The rest paper is organized as follows: Section 2 discusses the related studies in the domain. Section 3 presents CloudNet. Sections 4 includes the implementation details of the system and provides a comparative analysis of the different reference and optimized implementations of the underlying hash function and the data structures. Finally, Section 5 concludes and refers future work.

2 Related Work

The proposed effort is focused in adapting known security solutions in graphics cards. When it comes to applying anti-malware solutions on GPUs, the performance evaluation targets to improve either the general operation of the system or specific inner core elements.

Antivirus engines in GPUs are presented in [8], [9]. GrAVity [8], a massively parallel antivirus engine, modifies ClamAV (an open-source antivirus software, [10]) and

exploits the computing power of modern graphic processors to achieve 100 times higher performance than the CPU-only CalmAV. In [9], the authors propose a highly-efficient memory-compression approach for GPU-accelerated virus signature matching. They implement Aho-Corasick and Commentz-Walter automatons for performing GPU-accelerated virus scanning in conjunction with a set of virus signatures from CalmAV. Their antivirus engine is appropriate for real-world software and hardware systems.

Intrusion detection mechanisms for network security that exploit the computation capabilities of GPUs are reported in [11] and [12]. Gnort [11], which is based on the Snort open-source NIDS, is an intrusion detection system that is applied in GPUs to offload the costly pattern matching operation form the CPU. It increases the overall throughput and can be applied either for processing synthetic network traces or monitoring real traffic via an Ethernet interface. In [12], the authors propose a regular expression matching mechanism for intrusion detection on GPUs. Their system achieves a 48 times speedup over traditional CPU implementations.

The main method to create malware signatures is hash functions. To increase the performance of an anti-malware system, one can speed up the process of calculating the digest of the processing data. To take advantage of the parallel computing features in CPUs and, later, GPUs, there was an effort to either create hash functions that can be parallelized or design general mechanisms that can be used to parallelize any hash function. The SHA-3 contest was held to define a new hash function standard. However, Keccak [16], the hash function that won the contest, lacks of an inherent parallel nature. In order to create a parallel implementation that will exploit the potential of GPUs, one has to resort to alternative mechanisms. This can be achieved by introducing a Merkle tree [11] construction; a technique that can be used to parallelize any hash function and the one that was chosen for the presented system as well, as will be detailed later in this work.

CloudNet is a novel anti-malware engine that efficiently utilizes the state-of-the-art SHA-3 function in order to detect malware traffic. It is designed having in mind the intrinsic features and performance requirements of the modern IoT and cloud computing applications (e.g. [13], [14]). The system protects a real industrial setting, where IoT-enabled smart devices and wind turbines exchange information to the industrial cloud [15]. In regards to GrAVity and Gnort, it can be utilized in order to further increase the parallelization of the internal scanning procedure.

3 CloudNet

This implementation involves the development of a lightweight and efficient GPU-accelerated malware identification/hash lookup mechanism based on the SHA-3 [16] hash function. CloudNet exploits the CUDA toolkit and the parallelization capabilities of modern GPUs to improve the performance of the digest computation, i.e. the inner calculations of the chosen cryptographic hash function that will be run on the GPU.

The proposed solution implements two main operations. At the initialization phase, the system takes as input a directory with malware files, computes the digests of each

item, and stores them in a data structure. At the processing phase, the system takes network traffic as input, computes the digests of the relevant data segments, and compares them against the information that is maintain in the data structure. **Fig. 1** illustrates the malware identification phases.

For antimalware engines, computing the hash on an entire file from the start is a waste of resources, whether it's done on CPU or GPU. What you can do instead, is to compute the hash only on a small portion of the file (like the first 2KB, for instance), then, if it is found in a preliminary hash table, compute the hash value for the rest of the file. For most scanned files, the first hash will not be found (we assume that most files are clean) and the hash computation for the entire file is avoided. Nevertheless, in the rest paper we mention the results for hashing the entire file as the overall setting could be also utilized in other related domains that require full processing, i.e. blockchaining [17].

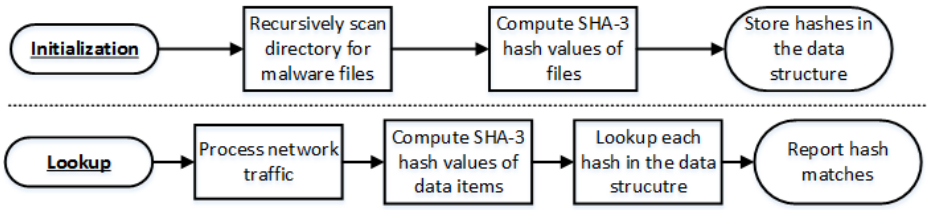

Fig. 1. Functionality sketch of GPU-accelerated mechanism.

The utility could scan the whole hash table looking for matches for the specific hash value it just calculated. This way, it could report back if the specific data exists in the initial hash table and what that item was. Such an identification mechanism could be useful for malware detection on network monitoring or local disks (e.g. to detect malicious or flagged files on the network).

The system maintains a hash-table with the hash values (signatures) of known malwares, which is created during the Initialization phase. Upon execution of the Lookup phase, it computes the digest of input traces using the same hash function and look for matches in the pre-computed values residing on the hash table.

Main operations include:

- Compute a digest
- Compare two digests
- Insert a new digest in the hash table
- Search for a digest in the hash table
- Report malware identification
- Delete or Move detected malicious traffic (optional)

For improving the performance of the GPU accelerated identification and verification solution, we focus on the optimization of the inner digest calculation and the data structure that maintains the processed data. We utilize the CUDA-optimized implementation of SHA-3 [12] to improve the inner operation, where we need to compute

the digest of inputted data. In the following, we analyze the performance of different implementations of SHA-3 in CPU and GPU in the different application settings. Moreover, we conduct a comparative analysis along three types of data structures (hash table, binary tree, and array) that can maintain the processed data.

4 Performance Evaluation

This section presents the implementation details of CloudNet. The proposed system was developed and evaluated on a test-bed featuring an Intel Core i7 Processor (6MB Cache, 2.1 GHz), 8GB of RAM, an NVIDIA GeForce GTX 1050 GPU (640 cores, 2GB buffer, 6Gbps memory speed, 1.4 GHz clock) [19], and the Ubuntu 17.10 operating system.

An initial CUDA implementation of the SHA-3 hashing mechanism was compared to the reference serial implementation as well as an optimized serial implementation, both of which are available on the official website [7].

We deploy the proposed solution on an IoT setting, where CloudNet monitors the network traffic between CPSs and cloud services. **Fig. 2** illustrates the application setting.

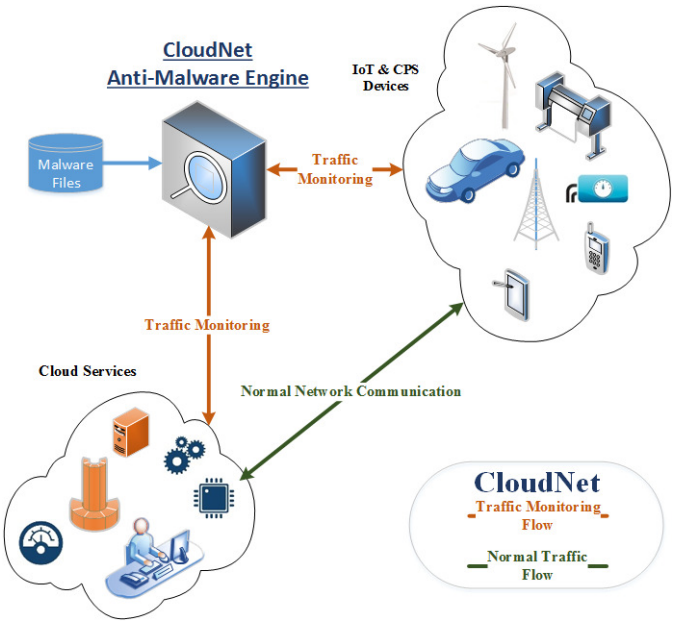

Fig. 2. The CloudNet setting.

As a benchmark, we utilize a trace of a real industrial wind park in Brande, Denmark. The examined Software-Defined Networking (SDN) and Network Function Virtualization (NFV) -enabled wind park is a characteristic use case of the next gen-

eration networks operating over 5G communication infrastructures. Several sensing devices and wind turbines transmit data to the backend Supervisory Control and Data Acquisition (SCADA) servers that monitors and/or react to environmental or other operational parameters. The trace focuses on traffic to/from the SCADA servers and was captured for around 1000 seconds of operation. Totally, more than 20.000 connections with low data rates are included. The TCP connections exhibit 100ms, 250ms, and 500ms end-to-end delay based on the service. The UDP connections have more stringent communication requirements of 10ms. These numbers are utilized as performance thresholds for the applicability of CloudNet. **Fig. 3** depicts the Industrial IoT (IIoT) wind park installation.

4.1 Choosing Data Structure

Before proceeding with the GPU implementation, the data structure to be used had to be finalized. To that end, the performance of alternative structures was examined and compared. In specific, array, binary tree, and hash table structures were investigated, focusing on the insertion time (i.e. how long it takes to insert a new entry into the data structure during initialization) and search time (i.e. how long it takes to lookup a specific entry in the data structure). The results of said comparison can be seen in **Fig. 4**, where it is evident that the hash table has an advantage over alternative structures.

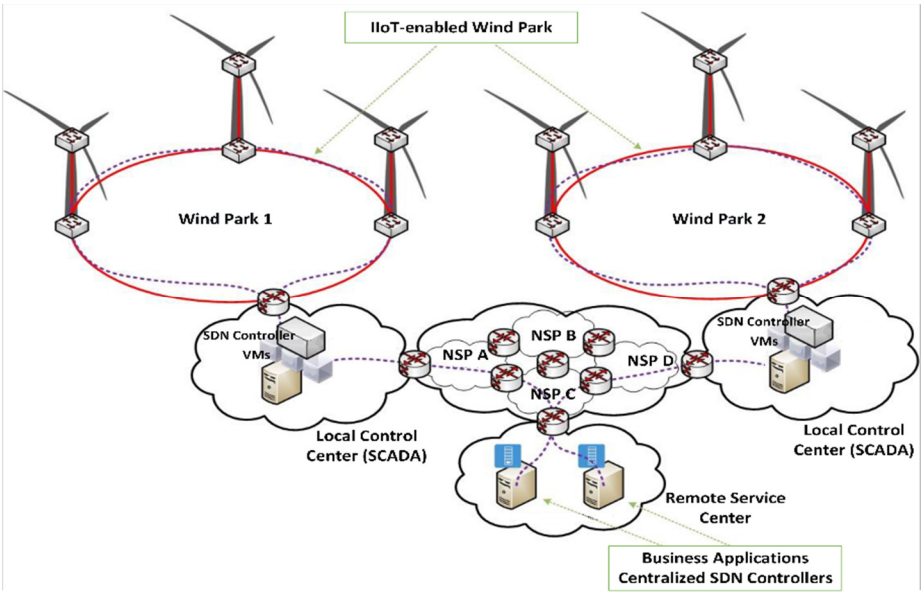

Fig. 3. The IIoT wind part installation.

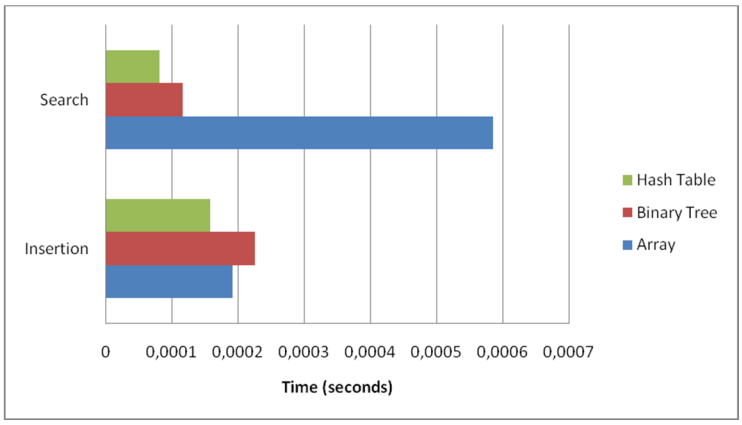

Fig. 4. Data structure performance comparison for CUDA-accelerated implementations.

4.2 GPU Results

Having finalized the data structure to be used, efforts could focus on the development and optimization of the CUDA hashing mechanism.

One way to parallelize any hash function is to look for parallelism in the inner functions, but this is not always an option as hash functions, by nature, are often complex and serialized. The SHA-3 CUDA implementation was based on a technique which can be used to parallelize any hash function, i.e. using a tree hash mode, also known as Merkle tree [20]. With this method, parallelism is realized outside the hash function, by running instances (tree leaves) concurrently and then gathering the results of each instance at an upper level of the tree. More details on the techniques typically used can be found in [21].

In addition, the program checks if the machine features a CUDA-enabled GPU and, depending on the result, then runs the Optimized CPU code or the CUDA code of the cryptographic hash function. On CUDA-enabled devices, the utility also checks the hardware features (CUDA version supported) of the GPU, in order to choose between the overlap-enabled and the basic tree hashing modes.

Some preliminary results of the simple hashing tree mechanism appear in **Fig. 5**. The results were obtained on a test system featuring an NVIDIA GeForce GTX 1050 GPU [19]. As is evident from the graph, the GPU-accelerated version of the hashing mechanism has a significant advantage in terms of the actual hashing performance, but also suffers extra I/O overhead because of the data transfer from host to GPU.

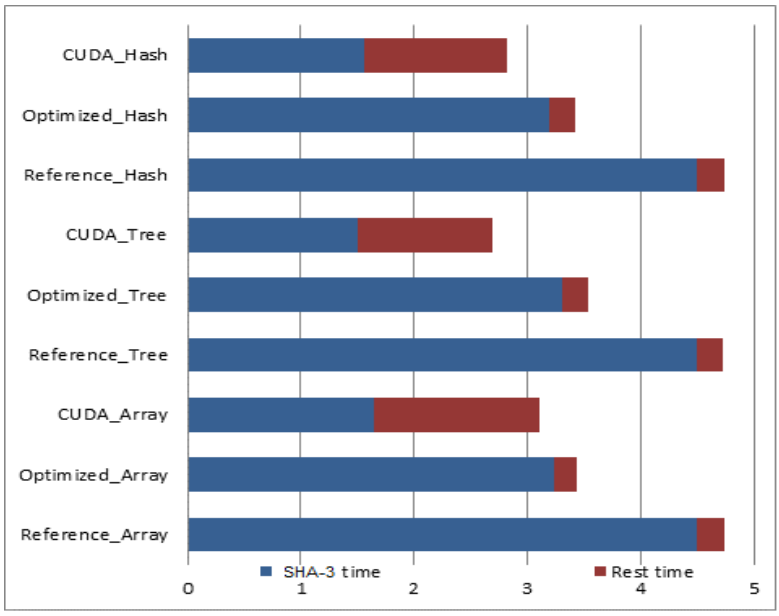

Fig. 5. Comparison of execution and I/O time of SHA-3 implementations.

4.3 Further Optimizations

The execution of the hashing mechanism was successfully improved by exploiting features present on newer generations of GPUs. Overlapping GPU and CPU computations, i.e. having the CPU compute the tree's top node of previous GPU computations, while the GPU processes other kernels, can yield significant improvements. Furthermore, if there is hardware support, data transfers and computations on the GPU can be overlapped as well, yielding further benefits. To accomplish this, page-locked memory and CUDA streams must be used, to allow the succession of transfers and computations in each stream, as described in the CUDA programming guide [18].

In order to assess the performance benefits from using the GPU to speed up the hashing mechanism of the developed application, a test folder was created and was used as a benchmark of the application's execution time. The folder contained 200 relatively small malware files (minimum size 3MB, maximum size 50MB, average size 20MB) and 100 bigger files (minimum size 70MB, maximum size 170MB, average size 123MB) and was used as an input at the application's initialization phase. Therefore the application recursively hashed all files and stored the results in the hash table.

The performance gains from employing the improved GPU implementation, namely overlapping GPU and CPU (top node) computations and the extra performance boost achieved by combining that with GPU transfer and computation overlaps, are evident in **Fig. 6.A** and **Fig. 6.B**. In addition to significantly improving the hashing speed itself, the overlap techniques allowed the minimization of the performance

penalties incurred by host-to-GPU and GPU-to-host transfers that are unavoidable in GPU implementations, as can be seen in **Fig. 6.C**.

The total execution time of the application when initializing for the given benchmark folder can be seen in **Fig. 7**. Noticeably, the time consumed on the hashing mechanism itself was successfully minimized by exploiting the GPU, to the point where hashing time is insignificant and the execution time is dominated by input/output and other processes. The latter significantly limits the total speedup achieved in the specific application, as can be seen in **Fig. 8.A**.

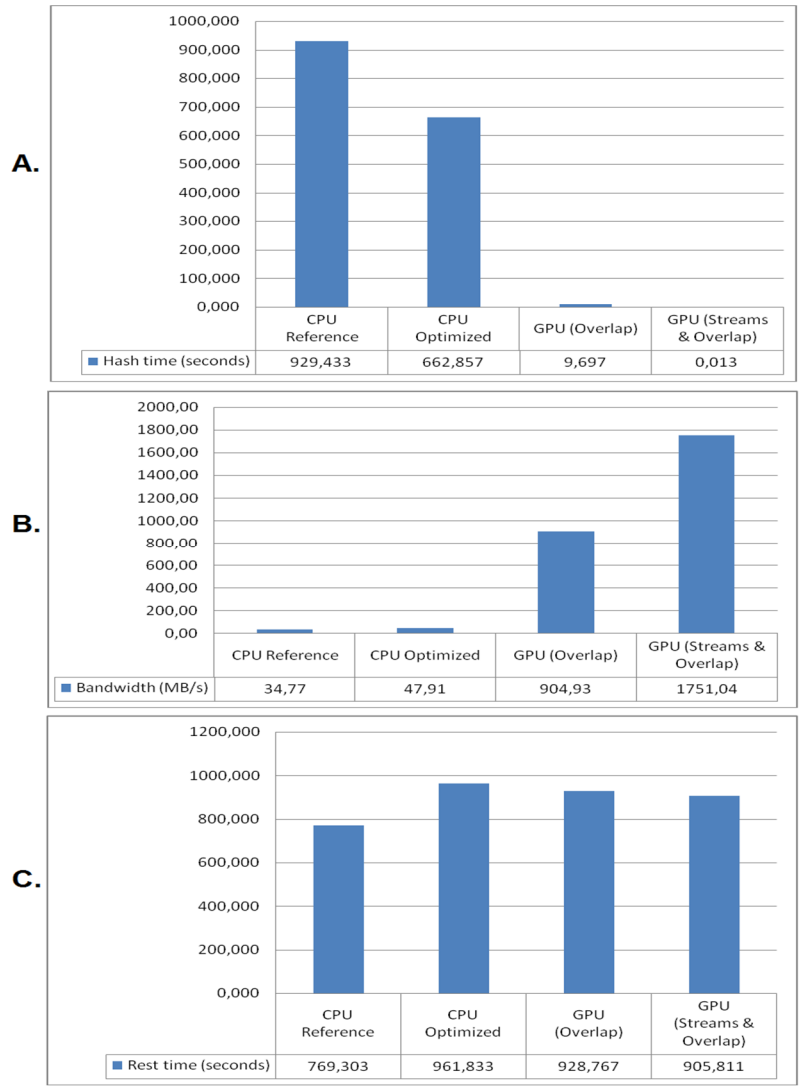

Fig. 6. Test folder's files processing: A. Time consumed on hashing, B. Speed when hashing, and C. Time consumed in non-hashing functions (I/O etc.).

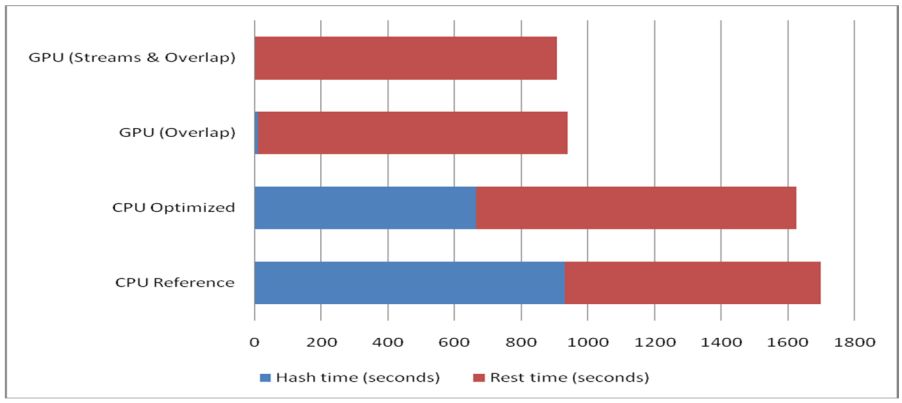

Fig. 7. Total execution time comparison.

Regarding the lookup phase, the GPU implementation's performance is a bit slower than the CPU alternatives, as can be seen in **Fig. 8.B**. This is to be expected as there is the extra overhead of transferring files to and from the GPU. This could be addressed by parallelizing the hash table search, but such an endeavor would only yield noticeable benefits in cases of large tables (i.e. tables storing values of many files). In its current form, the developed application would not benefit in a noticeable way, as the time spent on the lookup phase is not significant compared to other, more time consuming parts of the execution, like the initialization and the hashing itself (especially when run on the CPU).

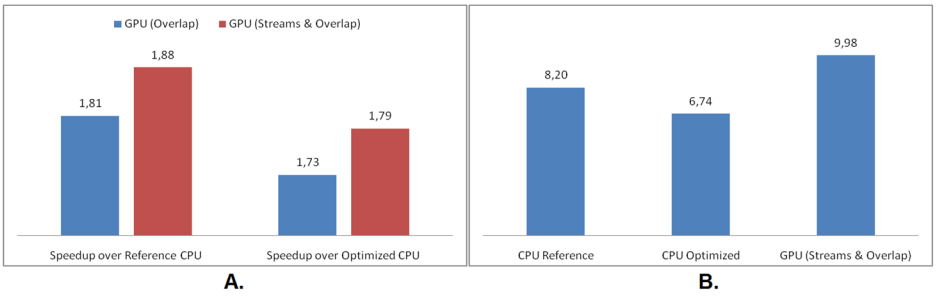

Fig. 8. A. Total speedup and B. Hash lookup times.

5 Conclusion

A lightweight and efficient GPU accelerating hashing and hash lookup mechanism utilizing the CUDA GPGPU toolkit was presented in this paper. We apply the proposed system in an anti-malware intrusion detection application, called CloudNet. Reference and optimized versions in CPU and GPU of the underlying primitives are implemented and a comparative analysis is conducted for selecting the most efficient ones. The system is evaluated under the traffic trace of a real wind park, featuring the

properties of an Industrial IoT setting. The most efficient GPU version is effectively almost 2 times faster than the reference implementation in CPU, essentially eliminating the overhead of hash calculations and having the bulk of the execution time being consumed by input/output operations.

Any attempts to further speed up the execution time of the application should, mainly, be focused on the initialization phase, aiming to alleviate the abovementioned bottleneck in hard disk I/O. Moreover, a combined architecture, which will utilize parallel computing in both CPU and GPU can be considered, where CPU cores would process distinct network traces and pass them to the proposed GPU accelerator. In any case, the benefits of using GPU acceleration when hashing files with the SHA-3 algorithm are substantial and beneficial for various applications, especially in cases where they are not performance-bound by input/output operations.

Acknowledgment

This work has received funding from the European Union Horizon's 2020 research and innovation programme under grant agreement No. 780315 (SEMIoTICS). The authors would also like to thank the network engineers maintaining the subject wind park in Brande, Denmark for their valuable input in interpreting the network traces.

References

1. Zhu, C., Shu, L., Leung, V. C. M., Guo, S., Zhang, Y., Yang, L. T.: Secure multimedia Big Data in trust-assisted sensor-cloud for smart city, IEEE Communications Magazine, IEEE, vol. 55, issue 12, pp. 24-30 (2017).
2. Zhu, C., Zhou, H., Leung, V. C. M., Wang, K., Zhang, Y., Yang, L. T.: Toward Big Data in green city, IEEE Communications Magazine, IEEE, vol. 55, issue 11, pp. 14-18 (2017).
3. Antonakakis, M., et al.: Understanding the Mirai botnet, 26th Usenix Security Symposium (SS), Vancouver, BC, Canada, August 16-18, pp. 1093-1110 (2017).
4. Lu, Z., Wang W., Wang, C.: On the evolution and impact of mobile botnets in wireless networks, IEEE Transactions on Mobile Computing, IEEE, vol. 15, issue 9, pp. 2304-2316 (2016).
5. NVIDIA Corporation, Santa Clara, California, USA, http://www.nvidia.com/ .
6. Compute Unified Device Architecture (CUDA), http://www.nvidia.com/object/cuda_home_new.html .
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak sponge function family, http://keccak.noekeon.org/.
8. Vasiliadis, G., Ioannidis, S.: GrAVity: a massively parallel antivirus engine, Recent Advances in Intrusion Detection (RAID), Springer, LNCS, vol. 6307, pp. 79-96 (2010).
9. Pungila, C., Negru, V.: A highly-efficient memory-compression approach for GPU-accelerated virus signature matching, Information Security (ISC), Springer, LNCS, vol. 7483, pp. 354-369 (2012).
10. Clam AntiVirus, Open Source (GPL) antivirus engine, http://www.clamav.net .
11. Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P., Ioannidis, S.: Gnort: high performance network intrusion detection using graphics processors, Recent Advances in Intrusion Detection (RAID), Springer, LNCS, vol. 5230, pp. 116-134 (2008).

12. Vasiliadis, G., Polychronakis, M., Antonatos, S., Markatos, E. P., Ioannidis, S.: Regular expression matching on graphics hardware for intrusion detection, Recent Advances in Intrusion Detection (RAID), Springer, LNCS, vol. 5758, pp. 265-283 (2009).
13. Hatzivasilis, G., Papaefstathiou, I., Manifavas, C.: SCOTRES: Secure Routing for IoT and CPS, IEEE Internet of Things Journal (IoT), IEEE, vol. 4, issue 6, pp. 2129-2141 (2017).
14. Hatzivasilis, G., Papaefstathiou, I., Manifavas, C.: Real-time management of railway CPS, 5th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS 2017), IEEE, Bar, Montenegro, 11-15 June (2017).
15. Hatzivasilis, G., Fysarakis, K., Soultatos, O., Askoxylakis, I., Papaefstathiou, I., Demetriou, G.: The Industrial Internet of Things as an enabler for a Circular Economy Hy-LP: A novel IIoT Protocol, evaluated on a Wind Park's SDN/NFV-enabled 5G Industrial Network, Computer Communications – Special Issue on Energy-aware Design for Sustainable 5G Networks, Elsevier, vol. 119, pp. 127-137 (2018).
16. National Institute of Standards & Technology (NIST): SHA-3 Winner Announcement, http://csrc.nist.gov/groups/ST/hash/sha-3/winner_sha-3.html .
17. Alexandris, G., Alexaki, S., Katos, V., Hatzivasilis, G.: Blockchains as Enablers for Auditing Cooperative Circular Economy Networks, 23rd IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2018), IEEE, Barcelona, Spain, 17-19 September, pp. 1-7 (2018).
18. NVIDIA Jetson TK1, http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html .
19. NVIDIA: GeForce GTX 1050, https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1050/ .
20. Merkle, R.: A certified digital signature, Advances in Cryptology (CRYPTO), Springer, pp. 218-238 (1990).
21. Sevestre, G.: Keccak Tree hashing on GPU, using NVIDIA Cuda API, https://sites.google.com/site/keccaktreegpu/ .