CUDACS: Securing the Cloud with CUDA-Enabled Secure Virtualization

Conference Paper · December 2010

DOI: 10.1007/978-3-642-17650-0_8 · Source: DBLP

2 authors:

Flavio Lombardi
Università Degli Studi Roma Tre
42 PUBLICATIONS **352** CITATIONS
SEE PROFILE

Roberto Di Pietro
Nokia Bell Labs
206 PUBLICATIONS **3,809** CITATIONS
SEE PROFILE

CUDACS: Securing the Cloud with CUDA-Enabled Secure Virtualization

Flavio Lombardi[1] and Roberto Di Pietro[2,3]

[1] Consiglio Nazionale delle Ricerche, DCSPI Sistemi Informativi,
Piazzale Aldo Moro 7, 00185 - Roma, Italy
flavio.lombardi@cnr.it
[2] Università di Roma Tre, Dipartimento di Matematica,
L.go S. Leonardo Murialdo, 1 00149 - Roma, Italy
dipietro@mat.uniroma3.it
[3] Consiglio Nazionale delle Ricerche, IIT,
Via Giuseppe Moruzzi 1, 56124 - Pisa, Italy
dipietro@iit.cnr.it

Abstract. While on the one hand unresolved security issues pose a barrier to the widespread adoption of cloud computing technologies, on the other hand the computing capabilities of even commodity HW are boosting, in particular thanks to the adoption of *-core technologies. For instance, the Nvidia Compute Unified Device Architecture (CUDA) technology is increasingly available on a large part of commodity hardware. In this paper, we show that it is possible to effectively use such a technology to guarantee an increased level of security to cloud hosts, services, and finally to the user. Secure virtualization is the key enabling factor. It can protect such resources from attacks. In particular, secure virtualization can provide a framework enabling effective management of the security of possibly large, heterogeneous, CUDA-enabled computing infrastructures (e.g. clusters, server farms, and clouds). The contributions of this paper are twofold: first, to investigate the characteristics and security requirements of CUDA-enabled cloud computing nodes; and, second, to provide an architecture for leveraging CUDA hardware resources in a secure virtualization environment, to improve cloud security without sacrificing CPU performance. A prototype implementation of our proposal and related results support the viability of our proposal.

Keywords: Cloud computing security, CUDA, virtualization, trusted platforms and trustworthy systems.

1 Introduction

A barrier to the widespread adoption of cloud computing technologies is the number of unresolved security issues. Recent improvements in Graphics Processing Units (GPU) provide the Operating System (OS) with additional computing resources that can be used for tasks that are not strictly related with graphics [15]. In particular, commodity hardware such as Ati Stream and Nvidia CUDA [3] feature manycore GPUs capable of efficiently executing most parallel tasks [1].

M. Soriano, S. Qing, and J. López (Eds.): ICICS 2010, LNCS 6476, pp. 92–106, 2010.

Recent proposals aim to expose graphics acceleration primitives to virtual machines (VM or guest) by giving the guest VM mediated access to GPU resources [5]. However, performance results are somewhat disappointing [10]. Furthermore, most often cloud nodes are not used for graphics-intensive tasks and workloads at all, and so their GPUs usually remain idle. Finally, most available server software is not yet capable of efficiently using GPUs. Hence, these underutilized computing resources could be used to increase the security of all CUDA-enabled nodes in the cloud. By CUDA-enabled we refer to a hardware/software combination that can host CUDA libraries and run related code. We focus on CUDA since the main competitor, Ati Stream technology, has not yet gained wide acceptance in the research community. However, considerations given in this paper will also be true for other manycore architectures.

In this paper, we show that secure virtualization (i.e. virtualization with advanced monitoring and protection capabilities [12]) is the key enabling factor to deploy CUDA technology in order to guarantee an increased level of security to cloud nodes, services, and finally to the user. Secure virtualization protects such additional computing resources from attacks and allows to leverage them without sacrificing CPU performance. In particular, we propose CUDA Cloud Security (CUDACS), i.e. a security system for the cloud that makes use of GPU computing resources to transparently monitor cloud components in order to protect them from malware. The proposed system integrates novel execution models in the security architecture, making the implemented mechanism further hidden and transparent.

Even though GPUs are already leveraged for security purposes [16], to the best of our knowledge, our work is the first effort that makes use of GPUs for guaranteeing increased security to cloud computing nodes and services. Secure virtualization is also used to protect access to CUDA resources. In particular, secure virtualization can provide the additional boundaries and tools that allow effective security management of possibly large CUDA-enabled computing infrastructures (e.g. clusters, server farms, and clouds).

Contributions. The contributions of this paper are twofold: (1) to investigate the characteristics and security requirements of actual CUDA-enabled cloud computing nodes; (2) to provide a novel architecture leveraging CUDA hardware resources in a secure virtualization environment, with the goal to improve cloud security without sacrificing performance. In particular, in the following we show how CUDACS can leverage GPU computing power through full virtualization to provide increased protection to an actually deployed cloud system such as Eucalyptus [17]. A prototype implementation is presented and first performance results are discussed. Results indicate that such a CUDA-enabled protection system greatly reduces the impact of the security system on guest VM performance.

Roadmap. The remainder of this document is organized as follows: next section surveys related work. Section 3 provides some background on CUDA and a taxonomy of CUDA-enabled cloud security issues and requirements. Section 4 describes

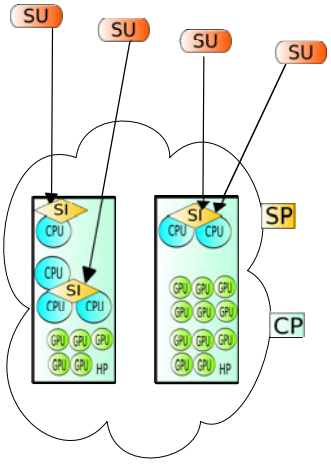

Fig. 1. CUDA cloud service model components: Cloud Provider (CP), Hosting Platform (HP), GPU core (G), CPU core (C), Service Provider (SP), Service Instance (SI), Service User (SU)

CUDACS architecture, and provides implementation details. Performance are discussed in Section 5 whereas in Section 6 concluding remarks are reported.

2 Related Work

Cloud security has been the objective of many research efforts in the last few years. Especially worth mentioning are [2], a complete cloud security risk assessment report and the work by Ristenpart [20]. In particular, the latter one shows that it is possible to achieve VM co-residence and, as a consequence, extract information from a target machine. In ACPS [13] the main requirements of a cloud protection system are identified and a framework for protecting cloud nodes and services via virtualization is presented. ACPS requirements are similar to those of present work. In particular, CUDACS and ACPS share the requirement that cloud security monitoring solutions have to be as transparent as possible to guests.

Virtualization-aware monitoring systems such as *SecVisor* [21] leverage Virtual Machine Managers (a.k.a. hypervisors) to allow some form of external monitoring (see [14]). Such proposals have limitations with respect to the cloud computing scenarios (e.g. *SecVisor* only supports one guest per each host). Most importantly, they can cause important performance degradation in the guest VMs that: (1) renders the adoption of such security systems expensive; and, (2) causes those systems to be detectable by timing analysis [24].

Present solutions do not leverage co-processor hardware to perform security checks or, if they do, additional proprietary hardware is usually required [19]. However, introducing proprietary hardware is usually expensive, or—from a system perspective—easily detectable. Instead, CUDA-capable GPUs are nowadays

common inside most server and desktop x86 hardware. They allow execution of most general purpose tasks apart from plain graphics processing.

Recent proposals allow the guest VM to access GPU resources to expose graphics acceleration to guest software [6]. Particularly interesting is vCUDA [11], allowing High Performance Computing applications to be executed within virtual machines—access to hardware acceleration is achieved by using an advanced API call interception and redirection technique. CUDA has already been used for security purposes: on the one hand to help breaking security of keys using a brute-force attack [9]; on the other hand Tumeo [22] presented an efficient GPU implementation of a pattern matching algorithm that can be useful when employed for intrusion detection purposes.

3 Background

Cloud services are available at different layers: **dSaaS** The data Storage as a Service providing storage capability over the network; **IaaS** The Infrastructure as a Service layer offering virtual hardware with no software stack; **PaaS** The Platform as a Service layer providing resources such as virtualized servers, OSes, and applications; **SaaS** The Software as a Service layer providing access to software as a service over the Internet. We have focused on the "lowest" computational layer (i.e. **IaaS**) where we can provide effective protection from threats to the other layers.

CUDA hardware and programming model are focused on data parallelism, and provide programming abstractions that allow programmers to express such parallelism in terms of threads of execution (kernel). Tens of thousands of such threads can run on a single GPU device. Threads can be grouped in a CUDA block, composed of multiple threads that cooperate and share the same set of resources. The CUDA toolkit is the most widely used and mature GPU programming toolkit available, providing APIs for performing host-GPU I/O and launching parallel tasks. It includes a compiler for development of GPU kernels in an extended dialect of C and C++.

3.1 Cloud Security

Figure 1 illustrates the CUDA cloud computing scenario we are interested in. A service provider (SP) runs one or more service instance (SI) on the cloud, which can be accessed by a group of final service users (SU). For this purpose, the SP hires resources from the cloud provider (CP). Such hosting platforms (HP) often feature multiple GPU cores (GPU), but the service workload is usually run exclusively on CPU core (CPU) resources, thus leaving GPU cores free to run other tasks. Only the CP has physical control over cloud machine resources.

In our model, and coherently with the literature, we rely on host integrity, since we assume the host to be part of the Trusted Computing Base (TCB) [8]. Indeed, guests can be the target of any possible kinds of cyber attack and intrusion such as viruses, code injection, and buffer overflow to cite a few. In case

the guest image is provided by the user, (e.g. in IaaS), the CP does not know the details of the service it is hosting, and the property of having a trustful VM cannot be guaranteed even at deploy time. As such, guest activity has to be continuously monitored for possibly malicious activity.

3.2 Requirements

We identified the core set of requirements to be met by a CUDA-capable security monitoring system for clouds as follows (see also [13]):

REQ1. Effectiveness and Precision: the system should be able to detect most kinds of attacks and integrity violations while avoiding false-positives;
REQ2. Transparency: the system should minimize visibility from VMs; that is: SP, SU, and potential intruders should not be able to detect whether the VM is monitored or not.
REQ3. Nonsubvertability: it should not be possible to disable, circumvent or alter the security system itself.
REQ4. Compatibility: the system should be deployable on the vast majority of CUDA-enabled cloud middleware and HW/SW configurations.
REQ5. Dynamic Reaction: the system should detect an intrusion attempt over a cloud component and it should take appropriate actions against such an attempt.
REQ6. Performance: the system should allow cloud physical nodes to securely host VMs and applications without sacrificing performance.
REQ7. Continuity of Service: the security system should survive a crash in the CUDA hardware and libraries and fall back to using the CPU.
REQ8. GPU Resource Protection: access to CUDA hardware resources should be controlled and monitored by the security system.

4 CUDA Cloud Security

We propose the CUDA Cloud Security system (CUDACS) to protect the integrity of cloud virtual machines and that of cloud middleware by allowing the host to monitor guest virtual machines and infrastructure components. Our approach is, to the best of our knowledge, the first effort to leverage CUDA hardware to efficiently and undetectably perform security and integrity monitoring of virtualized guests—as well as infrastructure components— of current cloud computing systems. CUDACS leverages virtual introspection to observe guest behavior [7]. It is an entirely host side architecture. This allows to deploy unmodified guest virtual machines, without requiring any coding or wrapping intervention.

In order to guarantee system security, CUDACS monitors code and data integrity for those components that are especially exposed to attacks, in particular core guest kernel and host middleware components. Furthermore, in order to monitor cloud entry points, we check the integrity of cloud components via

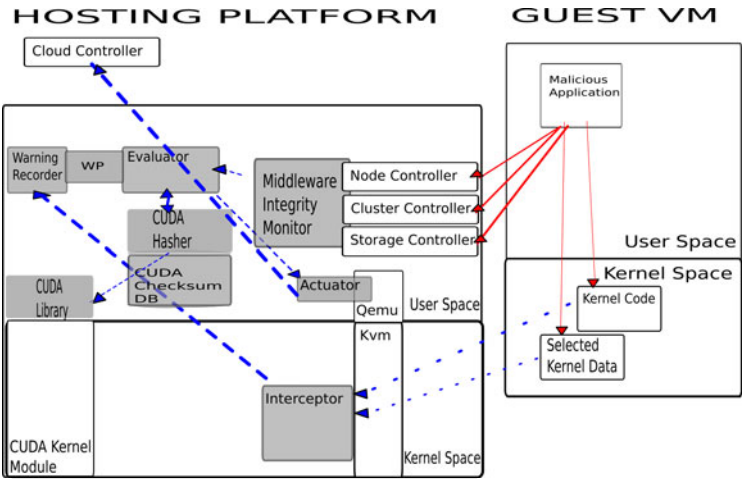

Fig. 2. Eucalyptus with CUDACS components (in gray): Interceptor, Warning Recorder, Warning Pool (WP), Middleware Integrity Monitor (MIM), Evaluator, CUDA Hasher, Actuator

Table 1. Warning Object core structure

Field Name	Type	Semantics
WId	integer	monotonically increasing value, the higher, the most recent
Dlv	integer	ranges from 1 to 4, the higher, the most dangerous
WTy	short	ranges from 0 to 254
SPt	pointer	start of considered memory area
Len	long	length in bytes of the memory area
PEv	pointer	previous event sequence list
NEv	pointer	next event sequence list

periodic checksum verification of executable files and libraries. We also need to ensure that an attacker-run application cannot detect that an intrusion detection system is active. This allows to also use our protection system as a honeypot.

The architecture of an Eucalyptus cloud with deployed CUDACS components is shown in Figure 2. Potentially dangerous data flows are depicted in continuous lines, while monitoring data flows are depicted in dashed lines. Guest activity (e.g. system_call invocation) is detected by the *Interceptor* and submitted by the *Warning Recorder* as a *Warning Object* to the *Warning Pool*. The potential threat is then evaluated by the *CUDA Evaluator*. The *Interceptor* in non-blocking, i.e. it cannot block or deny any system call. This helps preventing the monitoring system from being detected. Further, in order to guarantee increased invisibility (see REQ2), the *Warning Pool* allows to cache *Warning Objects* and to set priorities with respect to their order of evaluation. A large *Warning Object* incoming data rate might delay decision and reaction by the *Actuator*. This is the reason why some operations (e.g. *sidt* instruction execution) are given higher

Table 2. CUDACS: core set of functionalities

Functionality	**Semantics**
middleware_integrity_check	executable and library change monitoring
guest_kernel_data_check	kernel data structure content change monitoring
guest_kernel_code_integrity_check	kernel code implementation change monitoring
guest_service_check	service executable and library change monitoring

priority than other ones (e.g. file operations). In fact, every *Warning Object* (see Table 1) contains a danger level (Dlv) field set by the *Interceptor* according to the desired policy (see also [23]). In our implementation CUDACS policies are not expressed using a standard policy specification language; they are expressed as commands to be executed when different sets of conditions are met. Adopting state of the art solutions (see [4]) for policy specification to CUDACS is part of future work.

Our predefined policy sets decreasing danger level values with respect to the risk associated to possibly affected component, in particular:

4 alteration of middleware components;
3 changes in core guest kernel data and code (e.g. sidt);
2 modification of userspace libraries and code;
1 modification of guest system files.

The strong decoupling between attack detection and evaluation, that is performed in CUDA hardware, neutralizes the timing attack. In fact, the data rate of *Warning Objects* sent to the *CUDA Evaluator* and to the *CUDA Hasher* is kept constant. This way, such components are active even when the *Warning Pool* has no new data to process (see Figure 3). The *Warning Pool* manages *Warning Objects* composed of (see Table 1): Warning Identifier (WId): a unique increasing Id; Danger level value (Dlv): a measure of the dangerousness; a warning type (WTy) (e.g. guest_instruction; guest_crossviewanomaly; incoming_message; outgoing_message; middleware_component;...); Start Pointer (SPt): a pointer to the memory area start address; Lenght (Len): a value indicating memory area length; Previous Event (PEv): a pointer to events occurring before this one; Next Event (NEv): a pointer to events following present one.

The *CUDA Hasher* is in charge of performing fast checksumming of all the data chunks that are transferred to the GPU memory for inspection. Such chunks can be originated from memory footprints regarding code and data, and from files on (virtual) disks. A change in checksum value, with respect to the value computed and stored in the CUDA Checksum DB, is reported to the *Evaluator* and countermeasures are taken by the *Actuator* according to the implemented policy. As an example, a change in kernel code could trigger a snapshot and in-depth recording of low-level guest kernel behavior. Alternatively, such change might start a migration and/or restart of services on a different cloud guest. A non-blocking approach, such as the one presented here, might allow the attacker

to perform some—limited in time—tampering with the target system. However, the computing capacity offered by CUDA allows performing security checks in a very short time. Moreover, the Warning Pool mechanism allows to examine the most dangerous actions first (see Figure 4 and Figure 5), and so the attacker has a very limited time frame to take control of the system before being discovered and neutralized. Choking the security system is not possible given the *Warning Pool* priority-based policy. Furthermore, the Warning Pool can launch an alert and start dropping low-priority items in case the number of waiting warnings exceeds a given threshold. This is not really a problem given CUDA computing capabilities. The runtime *Middleware Integrity Monitor* (MIM) is a host userspace daemon that checks cloud middleware code and library integrity via the *Evaluator*.

In CUDACS, the *Checksum DB* database is copied inside GPU memory and contains computed checksums for critical host infrastructure and guest kernel code, data, and files. The runtime *Warning Recorder* sends warnings towards the *CUDA Evaluator*. The *CUDA Evaluator* examines such warnings and evaluates (based on its DB, see REQ1) whether the security of the system is endangered. In such a case the *Actuator* daemon is invoked to act according to a specified security policy (REQ5). At present, the security policy can be set up by configuring some simple conditions that, when met by warning objects, trigger the execution of the stored command string. The right reaction choice is obtained by bit-masking the warning object fields. As such, the reaction to an attack can be the replacement of the compromised service(s) or virtual machine(s), that can be resumed from their latest secure state snapshot. However, if the attack pattern is repeatedly exploited, external intervention is required and as such an alert is issued towards administrator and cloud controller. The *CUDA Evaluator* leverages a simple security engine that, being CUDACS a purely host-contained system, has to evaluate threats based on memory footprint and attack work-flow analysis. CUDA Evaluator does not leverage existing antivirus or IDS engines (e.g. Snort, Aide or Tripwire). On the contrary, it features an internal security engine whose database has been populated with experimentally-provided training data.

CUDACS enjoys the following features: it is transparent to guest machines; it supports full virtualization to protect access to CUDA resources and, it can be deployed on any CUDA-enabled cloud computing platform (see REQ4). CUDACS extends and improves the *ACPS* approach [13]; most important, CUDACS is completely transparent to guest machines given no CPU time is spent on computing checks. Furthermore, CUDA allows to perform faster monitoring and consequently leaves less time to the attacker. CUDACS is integrated in the virtualization software and leverages hardware virtualization support to monitor the integrity of guest and middleware components by scheduling, on CUDA GPUs, a checksum of such objects. Full virtualization and KVM support allow CUDACS to prevent direct access to host hardware, software (see REQ3) and GPU resources (see REQ8) from the guest VMs. As regards REQ7, it is worth

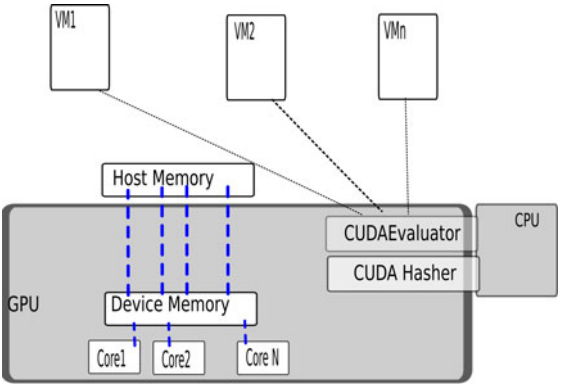

Fig. 3. CUDA-enabled monitoring: details of the interactions among GPU and CPU tasks

noting that if the CUDA device crashes or the library stops working, our security systems falls back to using CPU resources (CPUCS).

4.1 Implementation

Most existing cloud computing systems are proprietary (even though APIs are open and well-known) and as such do not allow modifications, enhancements or integration with other systems for research purposes. At present, Eucalyptus seems the most promising and supported open cloud platform (Ubuntu Cloud is based on it). This is the reason why we have chosen to deploy CUDACS onto Eucalyptus. Further, most considerations that apply to this context will also be valid for other platforms as well.

We implemented a prototype of CUDACS over Eucalyptus, whose system components are implemented as webservices. Eucalyptus [17] is composed of: a Node Controller (NC) that controls the execution, inspection, and termination of VM instances on the host where it runs; a Cluster Controller (CC) that gathers information about VM and schedules VM execution on specific node controllers; further, it manages virtual instance networks; a Storage Controller (SC)—Walrus—; that is, a storage service providing a mechanism for storing and accessing VM images and user data; a Cloud Controller (CLC), the webservices entry point for users and administrators in charge of high level scheduling decisions.

On Eucalyptus, CUDACS has been deployed on every node hosting VM instances. In Figure 4 the leveraged CUDA execution model is depicted: a number of threads are grouped together in blocks, themselves part of a grid. This allows to compute a large (depending on the available CUDA cores) number of security checks in parallel. Protection of GPU resources (see REQ8) is enforced by Kvm and monitored by periodic GPU memory dump and zeroing in order to avoid information leakage.

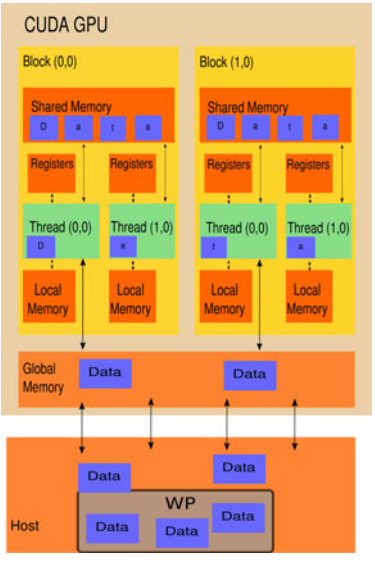

Fig. 4. CUDA-enabled integrity checks: details of the interactions

5 Performance

The experiments described in this section are aimed at evaluating CUDACS performance when deployed on a current CUDA cloud architecture. We compared the performance of a the same guest machine on a plain Kvm host, on a CUDACS host and on a CPUCS host (i.e. fallback mode using the CPU).

In order to better evaluate the performance cost/security benefit ratio of CUDACS, the set of functionalities offered by the CUDACS system has been summarized in Table 2.

The software/hardware configuration adopted for the following tests comprised dual core Athlon CPUs equipped with GT220 and GF8400M GPUs. Hardware virtualization was enabled on the hosts, running Ubuntu Server 10_04 x86_64 with Linux kernel 2.6.33 and KVM 0.12 . Guest operating systems were x86 Centos 5.5 with 1 virtual CPU and 1 GB RAM. Guest virtual disk was an image file on the host filesystem.

We adopted the latest version of the well known Phoronix test suite [18] to test different kind of workloads:

T1. CPU stress tests: the FLAC Audio Encoding 1.2 benchmark tests CPU performance when encoding WAV files to FLAC format.
T2. Ram performance test: system memory suite designed to test ram access speed.
T3. I/O stress test: an a-synchronous I/O benchmark to test read/write disk and file-system performance.

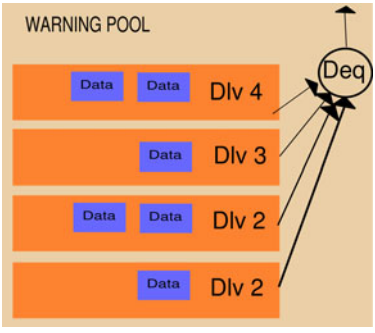

Fig. 5. CUDA priority-queue based Warning Pool: the order of extraction (by Deq) for Warning Objects is based on Dlv priority

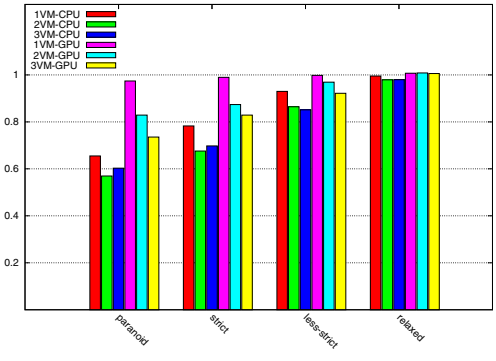

Fig. 6. CUDACS and CPUCS performance (normalized w.r.t. a Kvm guest) - CPU stress test (T1)

Four different approaches to monitoring have been tested for each benchmark:

(1) *paranoid*, where the security checks are repeated at every interaction with the virtual machine;
(2) *strict*, where the frequency of security checks is half as for the paranoid mode;
(3) *less strict*, where the frequency of security checks is half as for the strict mode;
(4) *relaxed*, (default mode) where security checks are less frequent and triggered just by suspicious guest activity.

In the following, graphs bars represent guest performance (averaged over host A and B) normalized with respect to the same test executed on an unmonitored Kvm guest on the same host.

First results where computational capabilities are tested are reported in Figure 6. Various configurations have been tested, featuring different host workloads. In particular, the test was repeated in scenarios where 1, 2, and 3 Virtual Machines where running the same benchmark at the same time on the host.

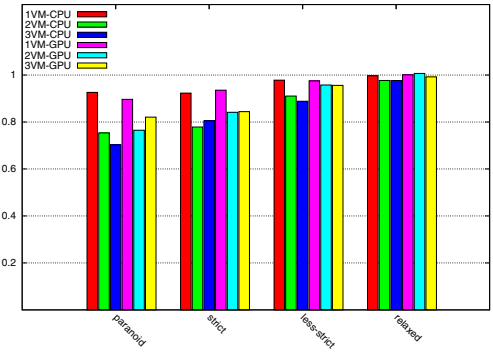

Fig. 7. CUDACS and CPUCS performance (normalized w.r.t. a Kvm guest) - Ramspeed test (T2)

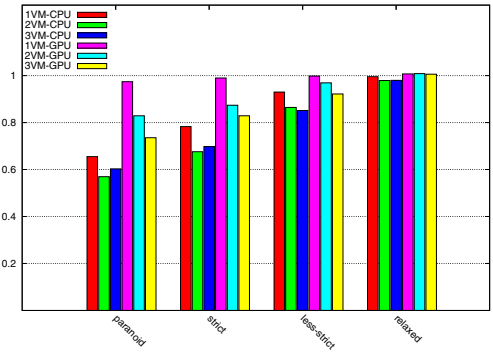

Fig. 8. CUDACS and CPUCS performance (normalized w.r.t. a Kvm guest) - I/O stress test (T3)

The objective was to observe performance under a number of possible configurations, given by the number of guests. To compare bars, remind that the reference value is provided by the Kvm guest run with the same configuration. From bars, it can be noticed that on the one hand the impact of monitoring is present for every configuration, and the performance penalty of the monitoring system becomes important when paranoid or strict modes are adopted. On the other hand, the use of our GPU-based system yields a performance increase over the CPU-based system in every possible configuration. In fact the performance of a CUDA-enabled system (CUDACS) appears better than its CPU-enabled fallback (CPUCS), not leveraging CUDA. We can observe though, that the best performance gains for GPU-monitored guests are obtained when only a single virtual machine was active. Further, there is some performance issue even with CUDACS running in the stricter monitoring modes, while increasing the number of sibling VMs. This is a counterintuitive behavior, since we were expecting the

GPU advantage to be more evident in case of a loaded host, as a cloud physical platform can be under heavy load.

In Figure 7 we report the performance results of a guest VM where memory access speed was tested. As above, CUDACS and CPUCS bars represent guest performance normalized with respect to the same test executed on an unmonitored Kvm guest. This results helped us to quantify the overhead on memory-bound operations that CUDA-enabled monitoring system can mitigate. Values show that CUDACS has a lead over CPUCS. In these tests the impact of monitoring on performance is less relevant than for the T1 benchmark. In particular, this is true for even the performance penalty of the monitoring system when paranoid or strict modes are adopted. Also here CUDACS performance lead over the CPU-based system is clear. However, CUDACS performance benefits are clearer for CPU-bound workloads. Indeed, the overall impact of CUDACS on guest performance is small. We expected the CUDACS performance benefit to be larger and think that a more refined implementation of the CUDA algorithms can lead to better results.

In Figure 8 we report the performance results of the I/O-bound benchmark executed on VMs. As above, CUDACS and CPUCS bars represent guest performance normalized with respect to the same test executed on an unmonitored Kvm guest. Here the performance of the guest VMs are affected from the presence of the monitoring system. Moreover, CUDACS performance is not always better than the CPUCS counterpart. This is a partially unexpected result, that can be explained with the I/O interactions between GPU and disk activity.

These first results are interesting (see REQ6) and encourage us to conduct further investigation aimed at leveraging the improvement margin previously highlighted. Of course in scenarios when manycore GPU resources have to be shared with other tasks, the impact of using such resources for security will yield different performance figures. We are working on this issue and will report in a future work.

6 Conclusion and Future Work

In this paper, we have contributed to achieve an effective use of virtualization techniques—leveraging CUDA architecture—in the context of cloud computing. First, we have proposed an extended architecture (CUDACS) for cloud protection using CUDA-enabled GPUs. CUDACS can monitor both guest and middleware integrity and protect from malware while remaining transparent to the guest. CUDACS enjoys unique features, such as the simultaneous parallel monitoring of multiple components and advanced immunity from guest-originated attacks. Second, CUDACS has been deployed onto a well-known open source cloud implementation. Finally, performance results have been collected. Preliminary results show that the proposed approach is effective and, most important, guest performance can benefit of CUDA-based monitoring and protection in all analyzed scenarios. Further work will be devoted to evaluate other platforms such as Ati Stream using the OpenCL model.

Acknowledgments

This work is partially supported by FP7-IP CONTRAIL Open Computing Infrastructures for Elastic Services. The authors would like to thank Matteo Signorini for his experimental support.

References

1. Bakkum, P., Skadron, K.: Accelerating SQL database operations on a GPU with CUDA. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, GPGPU 2010, pp. 94–103. ACM, New York (2010)
2. Catteddu, D., Hogben, G.: Cloud computing: Benefits, risks and recommendations for information security (2009), http://www.enisa.europa.eu/act/rm/files/deliverables
3. Nvidia Corporation. Nvidia's next generation CUDA compute architecture: Fermi (2009), http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf
4. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
5. Dowty, M., Sugerman, J.: GPU virtualization on VMware's hosted I/O architecture. SIGOPS Oper. Syst. Rev. 43(3), 73–82 (2009)
6. Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., Ranganathan, P.: GViM: Gpu-accelerated virtual machines. In: Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, HPCVirt 2009, pp. 17–24. ACM, New York (2009)
7. Hay, B., Nance, K.: Forensics examination of volatile system data using virtual introspection. SIGOPS Oper. Syst. Rev. 42(3), 74–82 (2008)
8. Hohmuth, M., Peter, M., Härtig, H., Shapiro, J.S.: Reducing TCB size by using untrusted components: small kernels versus virtual-machine monitors. In: Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, EW11, p. 22. ACM, New York (2004)
9. Hu, G., Ma, J., Huang, B.: Password recovery for RAR files using CUDA. In: Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2009, Washington, DC, USA, pp. 486–490. IEEE Computer Society, Los Alamitos (2009)
10. Andres Lagar-Cavilla, H., Tolia, N., Satyanarayanan, M., de Lara, E.: Vmm-independent graphics acceleration. In: Proceedings of the 3rd International Conference on Virtual Execution Environments, VEE 2007, pp. 33–43. ACM, New York (2007)
11. Lin, S., Hao, C., Jianhua, S.: vCUDA: GPU accelerated high performance computing in virtual machines. In: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, Washington, DC, USA, pp. 1–11. IEEE Computer Society, Los Alamitos (2009)
12. Lombardi, F., Di Pietro, R.: Kvmsec: a security extension for linux kernel virtual machines. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009, pp. 2029–2034. ACM, New York (2009)
13. Lombardi, F., Di Pietro, R.: Secure virtualization for cloud computing. Journal of Network and Computer Applications (2010) (in Press) (accepted manuscript), doi: 10.1016/j.jnca.2010.06.008

14. Lombardi, F., Di Pietro, R.: A security management architecture for the protection of kernel virtual machines. In: Proceedings of the Third IEEE International Symposium on Trust, Security and Privacy for Emerging Applications, TSP 2010, Washington, DC, USA, pp. 948–953. IEEE Computer Society, Los Alamitos (June 2010)
15. Luebke, D., Harris, M., Krüger, J., Purcell, T., Govindaraju, N., Buck, I., Woolley, C., Lefohn, A.: GPGPU: general purpose computation on graphics hardware. In: ACM SIGGRAPH 2004 Course Notes, SIGGRAPH 2004, p. 33. ACM, New York (2004)
16. Nottingham, A., Irwin, B.: GPU packet classification using OpenCL: a consideration of viable classification methods. In: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, SAICSIT 2009, pp. 160–169. ACM, New York (2009)
17. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus open-source cloud-computing system. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009, Washington, DC, USA, pp. 124–131. IEEE Computer Society, Los Alamitos (2009)
18. Phoronix. Phoronix test suite (2009), http://phoronix-test-suite.com/
19. Ranadive, A., Gavrilovska, A., Schwan, K.: IBMon: monitoring vmm-bypass capable infiniband devices using memory introspection. In: Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, HPCVirt 2009, pp. 25–32. ACM, New York (2009)
20. Ristenpart, T., Tromert, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2009, pp. 103–115. ACM, New York (2009)
21. Seshadri, A., Luk, M., Qu, N., Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP 2007, pp. 335–350. ACM, New York (2007)
22. Tumeo, A., Villa, O., Sciuto, D.: Efficient pattern matching on GPUs for intrusion detection systems. In: Proceedings of the 7th ACM International Conference on Computing Frontiers, CF 2010, pp. 87–88. ACM, New York (2010)
23. Zanin, G., Mancini, L.V.: Towards a formal model for security policies specification and validation in the SElinux system. In: Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies, SACMAT 2004, pp. 136–145. ACM, New York (2004)
24. Zimmer, C., Bhat, B., Mueller, F., Mohan, S.: Time-based intrusion detection in cyber-physical systems. In: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2010, pp. 109–118. ACM, New York (2010)