Probing Weaknesses in GPU Reliability Assessment: A Cross-Layer Approach

Lishan Yang
George Mason University, USA
lyang28@gmu.edu

George Papadimitriou
University of Athens, Greece
georgepap@di.uoa.gr

Dimitrios Sartzetakis
University of Athens, Greece
sartzet@di.uoa.gr

Adwait Jog
University of Virginia, USA
ajog@virginia.edu

Evgenia Smirni
William & Mary, USA
esmirni@cs.wm.edu

Dimitris Gizopoulos
University of Athens, Greece
dgizop@di.uoa.gr

***Abstract*—Due to extensive deployment and heavy usage of GPUs, ensuring the reliability of such devices is crucial. Current software-based reliability evaluation methodologies, albeit fast, often neglect the intricate hardware complexities of modern GPU designs. This oversight could result in misleading measurements and misguided decisions regarding protection strategies. This work breaks new ground by examining well-established vulnerability assessment methods for modern GPU architectures, from the microarchitecture all the way to the software layers. It highlights divergences between popular software-based vulnerability evaluation methods and the ground truth cross-layer evaluation (which, as we show, holds even when strong protection like triple modular redundancy is employed); accurate evaluation requires considering fault distribution from hardware to software. Our comprehensive measurements offer valuable insights into accurately assessing GPU reliability.**

***Index Terms*—reliability assessment, GPUs, cross-layer**

I. Introduction

Rapid developments in silicon manufacturing have enabled increased performance and improved energy efficiency of current graphics processing units (GPUs) [1]. Beyond performance, reliability assessment is a critical aspect of chip design [2]–[6]. Inaccurate reliability assessments can lead to pitfalls and wrong design decisions, finally resulting in more vulnerabilities [7]. Numerous general purpose GPU (GPGPU) applications have strict reliability requirements [8]–[10]. In the automotive domain where GPUs are also widely used, soft errors can result in hazards or accidents that are life-threatening [10]. The ever-increasing rate of soft errors in newer manufacturing technologies can jeopardize the aggressive evolution of GPUs, posing additional challenges. For example, since GPU applications are written using the Single-Instruction-Multiple-Threads (SIMT) paradigm, a single transient fault in a bit-cell of a hardware structure can result in multiple data corruptions at the application output [11] or a thread affected by a fault may supply several subsequent parallel threads with corrupted data [12], [13].

Assessing the impact of soft errors on GPU workloads at the early (unprotected) GPU design phase is important for unveiling potentially vulnerable hardware areas that need to be protected. Reliability assessment can be realized using different techniques which vary in their design maturity and granularity, the level of accuracy, and the speed of the assessment process [14]. Simulation is a very widely employed method for the assessment of the vulnerability to soft errors. GPU reliability evaluations are often performed on models of the actual GPU design using simulators [3]–[6], [15], [16]. Highly detailed and accurate simulation models at the RTL (Register Transfer Level), gate, or transistor level are extremely slow, not scalable, and not feasible. Less detailed models, for example at the microarchitecture level (using cycle-level simulators), are much faster than low-level highly-detailed models. Higher-level ISA (Instruction Set Architecture) simulation models, although faster, are even more abstract (hardware-agnostic).

Assessing the Architectural Vulnerability Factor (AVF) [17] of each individual microarchitectural structure of a chip during end-to-end program execution is a comprehensive way to evaluate the vulnerability of the entire system stack to soft errors, from the microarchitecture all the way to the software layers [7]. AVF is the probability that a soft error may produce an observable error at the application output [18]. Typically, application resilience is measured by experimental campaigns based on statistical fault injection (SFI) [19] or using analytical methods [17]. AVF measurements based on SFI provide useful and accurate insights for the application reliability profile but come with a limitation: since AVF measurements are based on cycle-level, microarchitecture-detailed simulation, obtaining the AVF of a GPU program is very slow [20].

Software-based vulnerability estimation methods, assuming software-visible origins of hardware bit flips, are significantly faster than full-system hardware measurements [21]. The speed difference can be two orders of magnitude or more.[1] These software-level methods derive the Software Vulnerability Factor (SVF) [7], representing the probability of a fault affecting program execution in a dynamic instruction. They are commonly used under the assumption that (a) reasonably model the effect of soft errors on the software layer (i.e., the overall resilience) and (b) at least provide correct relative vulnerability comparisons among different workloads. This work challenges these assumptions and demonstrates that

[1]For example, the AVF experiments of this study require 1,258 single-core machine days; SVF experiments take 10 machine days.

neither stands for GPU reliability assessment.

In this paper, we present an unbiased comparison of GPGPU reliability evaluation at different layers. To the best of our knowledge, this is the first study that such a cross-layer analysis has been performed in the GPU domain. We *quantify* and explain the diverging estimation results obtained when assessing the reliability of GPUs at different abstraction layers. The contributions of this work are summarized as follows:

- We demonstrate the magnitude of measurement errors introduced by software-level reliability evaluation methods, compared to the ground-truth, cross-layer AVF analysis. We employ two state-of-the-art, open-source SFI frameworks that both focus on NVIDIA GPUs: gpuFI-4 [3], [22] and NVBitFI [21], [23], which operate at the microarchitecture level and at the software level, respectively.
- We conduct a case study to measure the effectiveness of a strong software-based protection method, Triple Modular Redundancy (TMR) [24], which aims to eliminate silent data corruptions (SDCs). Our case study reveals two major insights: 1) although software-level evaluation (i.e., SVF) confirms that SDCs are effectively eliminated, the cross-layer evaluation (i.e., AVF) shows that some SDCs still remain despite the heavy penalty of protection in terms of performance (and thus, energy consumption), and 2) while most of the SDCs are eliminated, Detected Unrecoverable Errors (DUEs) instead increase, resulting frequently in higher vulnerability of the heavily protected application compared to the unprotected one.
- We provide insights and reasoning about the sources of error of software-level evaluation methods, which eventually lead to diverging results, and provide explanation on the reasons that lead to such discrepancies.

II. Characterization Highlights

We focus on a single-bit flip fault model for our evaluation. The differences between AVF and SVF in Fig. 1 are dramatic. Note the different scales of the vertical axis: AVF absolute values (the bottom graph) are always much smaller than the SVF ones (the top graph) because AVF considers the full hardware masking effects. The focus here is the relative *trends* (i.e., the vulnerability ranking of applications) but not the comparison of the actual vulnerabilities. Trends in SVF and AVF occasionally align and sometimes diverge. In certain cases, SVF and AVF produce entirely contrasting vulnerability estimations. This observation is very important, since such diverging SVF evaluations may lead designers to decide and apply a wrong protection scheme in practice. For example, *budgeted protection* (i.e., partial protection) is a common practice [25], [26] which protects only the most vulnerable components in the system. From SVF, high protection priority should be given to LUD. However, since the AVF SDC rate is extremely low, protecting this application from SDCs is unnecessary and the resources are wasted. Even worse, a wrong-decided protection scheme can increase the vulnerability of the application, instead of decreasing it.

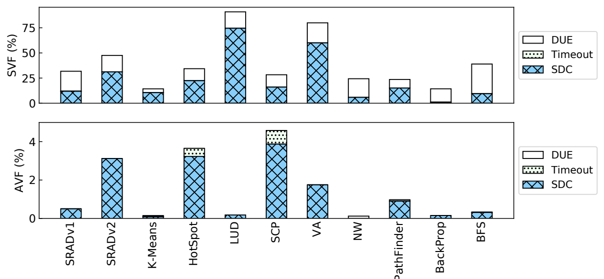

Fig. 1. Application-level comparison: AVF (bottom) and SVF (top).

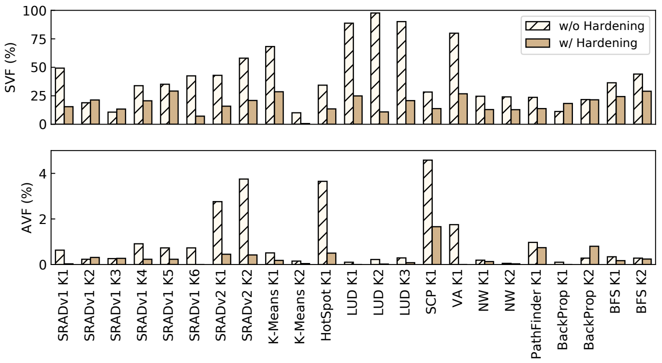

Fig. 2. AVF and SVF of hardened applications.

We employ both AVF/SVF methodologies to measure the effectiveness of a strong software-based protection method, Triple Modular Redundancy (TMR), which aims to eliminate SDCs albeit high cost [24]. Fig. 2 shows the AVF and SVF of kernels with/without hardening. While most kernels demonstrate improved resilience with the application of software-based hardening, several kernels exhibit heightened vulnerability post-hardening. Some kernels present contradictory trends, e.g., BackProp K2 shows a slight decrease in SVF, yet the AVF significantly rises post-hardening. This disparity highlights the potential for SVF to misleadingly indicate improved reliability. Meanwhile, TMR aims to rectify SDC fault effects, which SVF suggests are effectively eliminated. However, we observe a significant number of SDCs persist even after hardening in AVF experiments.

We highlight key reasons why higher-level fault injection methods fail to yield accurate reliability evaluations. In short, a fault can be initially considered architecturally visible, but it may eventually turn invisible to the architecture, and thus, to the software, which changes the distribution of faults that eventually become architecturally visible. As long as software-level fault injection tools do not consider this aspect, they fail to provide correct reliability estimation results.

Acknowledgments

This material is based upon work supported by the Commonwealth Cyber Initiative (CCI) grant (#HC-3Q24-047). This work was performed in part using computing facilities at William & Mary. Jog contributed to this work primarily while he was with William & Mary. This work was supported by the Hellenic Foundation Research and Innovation (HFRI) project VEMER.

References

[1] A. Arunkumar, E. Bolotin, D. Nellans, and C.-J. Wu, "Understanding the future of energy efficiency in multi-module GPUs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 519–532.

[2] L. Yang, B. Nie, A. Jog, and E. Smirni, "Practical resilience analysis of GPGPU applications in the presence of single- and multi-bit faults," *IEEE Transactions on Computers*, vol. 70, no. 1, pp. 30–44, 2021.

[3] D. Sartzetakis, G. Papadimitriou, and D. Gizopoulos, "gpuFI-4: A microarchitecture-level framework for assessing the cross-layer resilience of nvidia gpus," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 35–45.

[4] S. Tselonis and D. Gizopoulos, "GUFI: A framework for GPUs reliability assessment," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 90–100.

[5] B. Nie, L. Yang, A. Jog, and E. Smirni, "Fault site pruning for practical reliability analysis of GPGPU applications," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 749–761.

[6] L. Yang, B. Nie, A. Jog, and E. Smirni, "SUGAR: Speeding up GPGPU application resilience estimation with input sizing," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 1, pp. 1–29, 2021.

[7] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 902–915.

[8] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19 490–19 503, 2020.

[9] J. Athavale, A. Baldovin, R. Graefe, M. Paulitsch, and R. Rosales, "AI and reliability trends in safety-critical autonomous systems on ground and air," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 74–77.

[10] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[11] P. Rech, T. D. Fairbanks, H. M. Quinn, and L. Carro, "Threads distribution effects on graphics processing units neutron sensitivity," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4220–4225, 2013.

[12] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose, "Understanding error propagation in GPGPU applications," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 240–251.

[13] G. Kadam, E. Smirni, and A. Jog, "Data-centric Reliability Management in GPUs," in *the Proceedings of 51st International Conference on Dependable Systems and Networks (DSN), Virtual Event*, June 2021, pp. 271–283.

[14] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 101.

[15] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: Evaluating resilience of GPU applications," in *Proceedings of the Workshop on Silicon Errors in Logic-System Effects*, 2015.

[16] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 221–230.

[17] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. IEEE, 2003, pp. 29–40.

[18] N. Farazmand, R. Ubal, and D. Kaeli, "Statistical fault injection-based AVF analysis of a GPU architecture," in *Silicon Errors in Logic – System Effects (SELSE)*, 2012.

[19] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 502–506.

[20] C. Avalos Baddouh, M. Khairy, R. N. Green, M. Payer, and T. G. Rogers, "Principal kernel analysis: A tractable methodology to simulate scaled GPU workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 724–737. [Online]. Available: https://doi.org/10.1145/3466752.3480100

[21] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "NVBitFI: dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 284–291.

[22] (2022) gpuFI-4 github repository. [Online]. Available: https://github.com/caldi-uoa/gpuFI-4

[23] (2021) NVBitFI github repository. [Online]. Available: https://github.com/NVlabs/nvbitfi

[24] A. Milluzzi and A. George, "Exploration of TMR fault masking with persistent threads on Tegra GPU SoCs," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–7.

[25] L. Yang, B. Nie, A. Jog, and E. Smirni, "Enabling software resilience in GPGPU applications via partial thread protection," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1248–1259.

[26] M. H. Rahman, A. Shamji, S. Guo, and G. Li, "Peppa-x: finding program test inputs to bound silent data corruption vulnerability in hpc applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.