GPU Fault Tolerance

Toru Yamaguchi

1 Background

The GPU was originally developed to handle graphics processing and multimedia, with its primary application in gaming and its main use in consumer desktops. Research around fault tolerance, which is the ability for a system to maintain its specified operations despite failure of one or more of its components, for GPUs wasn't seen as such a priority as it is today. Early research by NVIDIA [1] on the issue of transient faults (i.e. random, temporary defects) focused on faults that would only end up affecting the user's visual experience by a certain magnitude. The paper referenced the Architectural Vulnerability Factor (AVF) [2], which is a strict metric to quantify whether a fault will progress to an architecturally visible error, and challenged it with the Visual Vulnerability Spectrum (VVS). This separate taxonomy for classifying transient faults on the GPU, aimed to relax fault tolerance in order to reduce unnecessary overhead, as most errors were deemed unnoticeable. Many pieces of literature in this area, including much of the papers cited here, like to mention in their introduction on how this unusually high tolerance to errors used to make up the view for GPU fault tolerance when it was first discussed. The gradual rise of General Purpose computing on GPU (GPGPU), instigated by incredible showcases of GPGPU performance [3] [4], re-aligned the understanding towards GPU fault tolerance, from treating the non-graphical market as niche [5] to recognising it as an expansive area of critical research. GPUs have provided a new paradigm of performance for High Performance Computing (HPC), machine learning, and various safety critical applications, which has inevitably lead to an increased interest in supporting the reliability of these applications. In the following sections, after going through some terminology, we will review several key fault tolerance techniques for GPUs.

2 Taxonomy and Basic Techniques

GPUs are architecturally unique compared to CPUs, hosting thousands of cores designed to provide massive parallelism. Despite their differences, GPUs experience many of the same faults as CPUs. We will introduce the main taxonomy for hardware faults and their relevance to GPUs. Starting with permanent and intermittent faults, they describe faults caused by manufacturing defects and aging, with intermittent representing temporary versions of permanent faults. One study, GPBurn [6], used extreme temperature tests to simulate accelerated aging, finding that at 170°C GPUs exhibit intermittent errors. As GPUs are continually adopted for automotive and space systems, where extreme temperatures are more frequent, this research helps to inform the state of GPUs to these faults. Further studies find some areas of the GPU, such as register files are naturally more robust towards permanent faults [7]. On the other hand, a more recent study [8] found that parallelism management units (PMU) exhibit high levels (20% to 60%) of silent data corruption (SDC), which are errors that don't trigger alerts or crashes but produce incorrect results, and high levels of detected unrecoverable errors (DUE), which are errors that result in a crash or system hang, in Fetch/Decoder units (>90%, 70%). It's clear that several areas in the GPU are more prone to SDC/DUEs than others. The mention of SDCs and DUEs helps to expand our taxonomy to the class of transient faults, which were introduced earlier, and represent the bulk of fault tolerance research for GPUs. Early research showed that GPUs are not only susceptible to SDCs caused by transient faults [9] but as the device continues to shrink, the SDC rate will also rise [10]. A hardware fault tolerance technique, error correcting codes (ECC), adds extra bits to memory data, which are checked when read to auto-correct single-bit errors and flag multi-bit errors. Whilst ECC significantly reduces the occurrence of SDCs, many papers find it insufficient by itself to consider GPUs as reliable [11] [12] [13], nevertheless it is a fault tolerance technique that can be effective for applications that require it. Another technique, commonly found in CPUs brought to GPUs, is a checkpoint and recovery system (CPR), which creates periodic checkpoints during execution, resorting back to one in the event of a failure. Checkpointing the GPU status was introduced with CheCUDA [14], and subsequently many iterations and improvements have been made as the GPU framework changes. Both ECC and CPR are essential techniques for fault tolerance, however, in the proceeding sections, we will dive deeper into other techniques that are crucial for GPU fault tolerance.

3 Redundancy

Redundancy is a common fault tolerance technique built on the premise that random faults are unlikely to occur at the same position for multiple instances of the same input. A good starting point would be a simple duplication of processes with comparison (DWC) for error detection, which unsurprisingly it sees double the execution time [15]. Variations of DWC that duplicate instructions in either space or time [16] saw varied overheads 90% to 151% but found they reduced the SDC rate by one order of magnitude compared to ECC, mentioned previously. Several other techniques extend it by exploiting GPU hardware and software [17] [18], such as utilising unused parallelism or by increasing the number of registers required for program execution, interestingly they find DWC has application dependent performance. This finding was supported by another redundancy technique called redundant multi-threading, involving one thread with full computation and another with minimal, which showed mixed performance [19]. If the GPU optimisation involves utilising unused resources, which many redundancy optimisations do, this can explain the differing outcomes since certain applications may naturally use more resources than others. This insight into application dependent performance will be further explored later. If we turn our attention to safety-critical applications, the emerging use of GPUs demand high reliability. Triple mode redundancy (TMR) involves having three executions of a single process, with a voting system in place if we observe a fault in an execution. Implementations will try to avoid the expected 200% overhead, such as Warped-RE [20] which exploits inherent redundancy between threads of the same warp. It uses dual mode redundancy (DMR) to detect errors but forces TMR if an error is detected, achieving an average overhead of just 29%. However, this method will still suffer from additional overhead if the application has high resource utilisation. This re-occurring theme that the performance of redundancy is application dependent can bring forward the idea of selective redundancy. Explored in The Hauberk approach [21], selective redundancy proposes that we observe and select the areas of the program with the highest error propagation for the most protection, thereby reducing unnecessary overhead. Many methods build on this approach, one used fault injections to examine vulnerable registers [22] and provided software/hardware based hardening such as register duplication to those areas. An important aspect of this method is choosing the critical areas (fault sites), this is often done through exhaustive fault injections (FI). However, research suggests that not all GPGPU fault sites are fault prone, with lots being considered redundant [23] [24]. This research is important when devising the FI process, with specific methodologies [25] being proposed to significantly reduce the FI campaign time. Redundancy has found its place as a staple method to effectively provide fault tolerance, as with most techniques, high reliability comes at the cost of performance, something safety-critical applications such as automotive systems have accepted [26] [27] to maintain reliability. This section has covered redundancy when applied to GPUs, with an emphasis on the lessons learnt, an important one being that many redundancy techniques find that performance is often application dependent, this is something at the very heart of our next fault tolerance technique.

4 Algorithm Based Fault Tolerance

Algorithm Based Fault tolerance offers a different approach to reliability, like the name suggests, it implements hardening techniques to specific algorithms so that faults are detected based on the algorithm's result. The main method involves encoding the input data for the algorithm, modifying the algorithm to operate on encoded data, and finally to perform checks using checksums to validate the result. An insight into why ABFT has potential to work is given in figure 1, which shows the mean work between failure (MWBF) for several algorithms on different CPUs and GPUs. If we focus on Maxwell, Steamroller, Kepler, and Pascal as the GPU results, our main takeaway is that algorithms naturally exhibit different levels of failure, something ABFT is designed to consider. When it was first proposed [29], ABFT pre-emptively highlighted its complementary nature with multi-processor systems, making it ideal for GPUs. A comparison of ABFT specifically with matrix multiplications (MM) [30] with techniques discussed previously such as DWC or TMR, highlighted the efficiency of ABFT, with added benefits such as avoiding unnecessary round-off error corrections, which is something ABFT with MM has seen strong results with [31]. Being the first algorithm to have ABFT applied to it, MM has seen various implementations on the GPU. A popular use case would be for Neural Networks, where reliable and efficient MM is required. Several studies[32] show that ABFT outperforms other FT methods such as ECC, in detecting and correcting errors. A notable implementation [33] which moved the original method from off-line, meaning core computation and error checking are

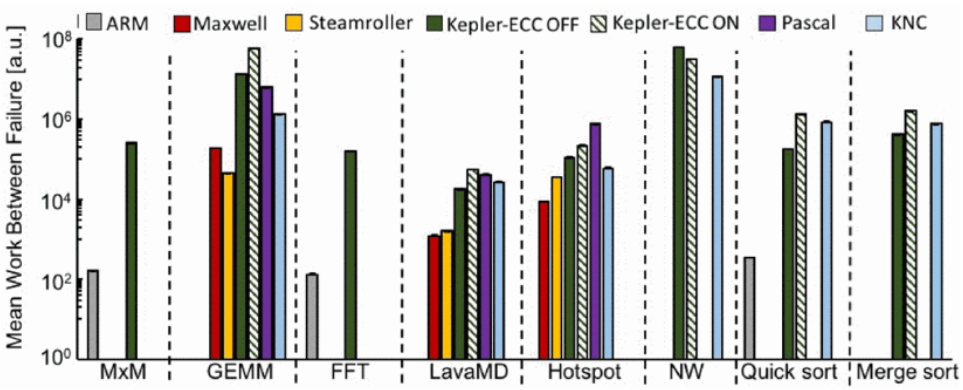

Figure 1: Injection results of different algorithms on CPUs and GPUs [28]

decoupled, to an on-line approach, where error checking is done during computation, provided the mechanism to immediately check for errors whilst being able to handle multiple errors. This approach has been used for various other studies such as combining it with kernel fusion [34], Cholesky Decomposition [35], and Fast Fourier Transform [36] making it a technique that can generalise to many algorithms. Conversely, an insight in [37] suggests that more than 50% of errors are masked within a single kernel execution, therefore checking for errors during computation as with on-line ABFT could produce unnecessary overhead. Its well known that the limitation to ABFT is its algorithm dependent nature, despite being the reason it has such a low overhead compared to techniques like TMR and DWC [30]. It can be seen though that GPU implemented algorithms aren't in abundance compared to CPU-based, due to its parallel constraints, most relying on efficient matrix and vector operations or linear algebra methods. This naturally allows most algorithms on the GPU to share properties that can be exploited further, something ABFT has managed to do to be a key fault tolerance technique for GPU applications.

5 Conclusion

We have reviewed various fault tolerance techniques for GPUs. Although GPUs are still primarily used for graphical applications, hence some hardware features such as ECC aren't enabled by default, the adoption of GPUs for HPC and safety-critical applications requires continued research especially as GPUs evolve for new use cases. As we look at the current state of GPUs, performance is still improving, but a study shows that continued reduction in execution times will not necessarily require more hardware and software sacrifices to maintain reliability [38], this is a positive sign for the future of GPU reliability.

References

[1] Jeremy W. Sheaffer, David P. Luebke, and Kevin Skadron. The visual vulnerability spectrum: characterizing architectural vulnerability for graphics hardware. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '06, page 9–16, New York, NY, USA, 2006. Association for Computing Machinery.

[2] 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.*, pages 29–40, 2003.

[3] John Nickolls and William J. Dally. The gpu computing era. *IEEE Micro*, 30(2):56–69, 2010.

[4] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[5] Jeremy W. Sheaffer, David P. Luebke, and Kevin Skadron. A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '07, page 55–64, Goslar, DEU, 2007. Eurographics Association.

[6] David Defour and Eric Petit. Gpuburn: A system to test and mitigate gpu hardware failures. In *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 263–270, 2013.

[7] Sotiris Tselonis, Vasilis Dimitsas, and Dimitris Gizopoulos. The functional and performance tolerance of gpus to permanent faults in registers. In *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, pages 236–239, 2013.

[8] Juan David Guerrero Balaguera, Josie Esteban Rodriguez Condia, Fernando Fernandes Dos Santos, Matteo Sonza Reorda, and Paolo Rech. Understanding the effects of permanent faults in gpu's parallelism management and control units. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Imran S. Haque and Vijay S. Pande. Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 691–696, 2010.

[10] Shuguang Feng, Shantanu Gupta, Amin Ansari, and Scott Mahlke. Shoestring: probabilistic soft error reliability on the cheap. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, page 385–396, New York, NY, USA, 2010. Association for Computing Machinery.

[11] Daniel A. G. Oliveira, Paolo Rech, Laércio L. Pilla, Philippe O. A. Navaux, and Luigi Carro. Gpgpus ecc efficiency and efficacy. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 209–215, 2014.

[12] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, Luigi Carro, and Arthur Bland. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–342, 2015.

[13] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 249–258, 2017.

[14] Hiroyuki Takizawa, Katsuto Sato, Kazuhiko Komatsu, and Hiroaki Kobayashi. Checuda: A checkpoint/restart tool for cuda applications. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 408–413, 2009.

[15] Martin Dimitrov, Mike Mantor, and Huiyang Zhou. Understanding software approaches for gpgpu reliability. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, page 94–104, New York, NY, USA, 2009. Association for Computing Machinery.

[16] Daniel A. G. Oliveira, Paolo Rech, Heather M. Quinn, Thomas D. Fairbanks, Laura Monroe, Sarah E. Michalak, Christine Anderson-Cook, Philippe O. A. Navaux, and Luigi Carro. Modern gpus radiation sensitivity evaluation and mitigation through duplication with comparison. *IEEE Transactions on Nuclear Science*, 61(6):3115–3122, 2014.

[17] Hyeran Jeon and Murali Annavaram. Warped-dmr: Light-weight error detection for gpgpu. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 37–47, 2012.

[18] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. Optimizing software-directed instruction replication for gpu error detection. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 842–854, 2018.

[19] Jack Wadden, Alexander Lyashevsky, Sudhanva Gurumurthi, Vilas Sridharan, and Kevin Skadron. Real-world design and evaluation of compiler-managed gpu redundant multithreading. *SIGARCH Comput. Archit. News*, 42(3):73–84, June 2014.

[20] Mohammad Abdel-Majeed, Waleed Dweik, Hyeran Jeon, and Murali Annavaram. Warped-re: Low-cost error detection and correction in gpus. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 331–342, 2015.

[21] Keun Soo Yim, Cuong Pham, Mushfiq Saleheen, Zbigniew Kalbarczyk, and Ravishankar Iyer. Hauberk: Lightweight silent data corruption error detector for gpgpu. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 287–300, 2011.

[22] Marcio Goncalves, Fernando Fernandes, Ivan Lamb, Paolo Rech, and Jose Rodrigo Azambuja. Selective fault tolerance for register files of graphics processing units. *IEEE Transactions on Nuclear Science*, 66(7):1449–1456, 2019.

[23] Bin Nie, Lishan Yang, Adwait Jog, and Evgenia Smirni. Fault site pruning for practical reliability analysis of gpgpu applications. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 749–761, 2018.

[24] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. Practical resilience analysis of gpgpu applications in the presence of single- and multi-bit faults. *IEEE Transactions on Computers*, 70(1):30–44, 2021.

[25] Fritz Previlon, Charu Kalra, Devesh Tiwari, and David Kaeli. Characterizing and exploiting soft error vulnerability phase behavior in gpu applications. *IEEE Transactions on Dependable and Secure Computing*, 19(1):288–300, 2022.

[26] Sergi Alcaide, Leonidas Kosmidis, Carles Hernandez, and Jaume Abella. Achieving diverse redundancy for gpu kernels. *IEEE Transactions on Emerging Topics in Computing*, 10(2):618–634, 2022.

[27] Sergi Alcaide Portet, Leonidas Kosmidis, Carles Hernandez, and Jaume Abella. Software-only triple diverse redundancy on gpus for autonomous driving platforms. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 82–88, 2020.

[28] Vinícius Fratin, Daniel Oliveira, Caio Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. Code-dependent and architecture-dependent reliability behaviors. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 13–26, 2018.

[29] Kuang-Hua Huang and Jacob A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, C-33(6):518–528, 1984.

[30] Hans-Joachim Wunderlich, Claus Braun, and Sebastian Halder. Efficacy and efficiency of algorithm-based fault-tolerance on gpus. In *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, pages 240–243, 2013.

[31] Claus Braun, Sebastian Halder, and Hans Joachim Wunderlich. A-abft: Autonomous algorithm-based fault tolerance for matrix multiplications on graphics processing units. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 443–454, 2014.

[32] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. Analyzing and increasing the reliability of convolutional neural networks on gpus. *IEEE Transactions on Reliability*, 68(2):663–677, 2019.

[33] Chong Ding, Christer Karlsson, Hui Liu, Teresa Davies, and Zizhong Chen. Matrix multiplication on gpus with on-line fault tolerance. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pages 311–317, 2011.

[34] Shixun Wu, Yujia Zhai, Jinyang Liu, Jiajun Huang, Zizhe Jian, Bryan Wong, and Zizhong Chen. Anatomy of high-performance gemm with online fault tolerance on gpus. In *Proceedings of the 37th ACM International Conference on Supercomputing*, ICS '23, page 360–372, New York, NY, USA, 2023. Association for Computing Machinery.

[35] Jieyang Chen, Xin Liang, and Zizhong Chen. Online algorithm-based fault tolerance for cholesky decomposition on heterogeneous systems with gpus. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 993–1002, 2016.

[36] Shixun Wu, Yujia Zhai, Jinyang Liu, Jiajun Huang, Zizhe Jian, Huangliang Dai, Sheng Di, Franck Cappello, and Zizhong Chen. Turbofft: Co-designed high-performance and fault-tolerant fast fourier transform on gpus. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '25, page 70–84, New York, NY, USA, 2025. Association for Computing Machinery.

[37] Guanpeng Li, Karthik Pattabiraman, Chen-Yang Cher, and Pradip Bose. Understanding error propagation in gpgpu applications. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 240–251, 2016.

[38] Fernando dos Santos and P. Rech. Can gpu performance increase faster than the code error rate? *The Journal of Supercomputing*, 80:1–29, 04 2024.