Examining the Impact of ACE interference on Multi-Bit AVF Estimates

Fritz Previlon, Mark Wilkening, Vilas Sridharan†, Sudhanva Gurumurthi‡ and David R. Kaeli
ECE Department, Northeastern University, Boston, MA, USA,
previlon@ece.neu.edu, wilkening.m@husky.neu.edu, kaeli@ece.neu.edu
†RAS Architecture, Advanced Micro Devices, Inc., Boxborough, MA, USA, vilas.sridharan@amd.com
‡AMD Research, Advanced Micro Devices, Inc., Boxborough, MA, USA, sudhanva.gurumurthi@amd.com

***Abstract*—Until recently, soft error reliability has been focused on single-bit errors and as a consequence, methodologies for Architectural Vulnerability Factor (AVF) analysis have been well defined and established for single-bit faults. However, studies have shown that multi-bit faults are becoming more prevalent with technology scaling [1]. If this trend continues, multi-bit faults will eventually become a high percentage of all microprocessor faults.**

Research into modeling methodologies for multi-bit faults is scarce. Recently, we presented an Architecturally Correct Execution (ACE) analysis methodology to evaluate the AVF of spatial multi-bit faults (MB-AVF) [2]. We used our methodology to study multi-bit AVF by extending the ACE analysis infrastructure of a performance simulator. While this methodology precisely measures AVF for Detected Unrecoverable Errors (DUEs), it only approximates AVF for Silent Data Corruptions (SDC). This is because the methodology determines a bit's ACE state using a single-bit ACE analysis. However, a bit's ACE state may change due to the presence of another bit flip, a condition termed *ACE interference*, and this effect will not be captured by the MB-AVF modeling methodology. As a result, our SDC calculation method is accurate only if ACE interference is rare for multi-bit faults.

In this paper, we present a fault injection study to determine the prevalence of ACE interference in typical benchmarks executing on a GPU. Our results show that ACE interference is a rare event in GPUs: we found that the ACE state of a bit rarely changes in presence of other faults. These results support the conclusion that our multi-bit ACE analysis can accurately estimate the SDC AVF of a processor design.

I. Introduction

With hundreds of cores and improved programmability, Graphics Processing Units (GPUs) have become the *compute accelerator of choice* for general purpose and supercomputing applications. However, these high-throughput devices are designed primarily for graphics rendering (written in DirectX and OpenGL), which tend to be fault insensitive compared to the applications that leverage the GPU for computation. As more and more applications start to leverage the GPU for compute acceleration, GPU reliability is quickly becoming a major concern. Reliability studies have shown that particle-induced transient faults in SRAM are major contributors to the rate of microprocessor faults [3]. These transient faults arise from high-energy particle strikes, which deposit charge onto transistors as they pass through a silicon device. A sufficient deposited charge has the potential to invert the state of a logic device, therefore causing a temporary fault in this device.

To cope with transient faults, microprocessor vendors establish target fault rates for each design. Designers then perform extensive analysis to ensure that a design meets these target fault rates. One such analysis is architectural vulnerability factor (AVF) analysis, which measures the fraction of faults in a structure that become program-visible errors. When combined with the raw fault rate of a structure, AVF can be used to calculate the soft error rate of a structure. Widely-used techniques to measure AVFs include statistical fault injection and ACE analysis [4][5][6]. These two methods help designers analyze the AVF of an architecture in various stages of the design process.

A fault injection campaign compares the reference behavior of the circuit for a given workload (that is, the correct behavior validated by the designer) with the behavior obtained in the presence of each fault in a predetermined set [7]. Hardware-implemented fault injection, software-implemented fault injection, and simulation-based fault injection are the three most common approaches used to evaluate and study error-handling mechanisms. Error injection techniques, at high-level abstraction layers, are helpful to understand application-level erroneous behaviors [8].

ACE analysis identifies bits necessary for architecturally correct execution (ACE bits) of a program and measures the percentage of ACE bits in a hardware structure. Corruption of an ACE bit results in a visible error at the output of the program. ACE analysis is conservative, as it initially assumes that all bits in a hardware structure are ACE, then finds the bits that can be proven unnecessary for the correct execution of the program (unACE bits). Hardware designers can perform ACE analysis in performance simulators early in the design cycle.

Due to technology scaling, current trends suggest that particle strikes will increase in size and scale, with increases in both the number of strikes affecting multiple bits, as well as the number of bits flipped by a single strike. Research suggests that the multiplicity of multi-cell upset will grow with decreasing feature size [1]. We commonly refer to these multi-cell upsets as *spatial multi-bit faults*.

While there have been countless studies on single-bit transient faults and their effects, and on methodologies to quantify AVFs for single-bit faults, few studies have assessed the impact of spatial multi-bit transient faults. We recently introduced a method to quantify and measure MB-AVFs for spatial multi-

bit faults [2]. Using this method, we measured MB-AVFs in a performance simulator for detected, unrecoverable errors (DUE MB-AVFs) and for silent data corruption errors (SDC MB-AVFs).

Our approach relies on performing single-bit ACE analysis to identify a bit's ACE state, and uses these values to calculate multi-bit AVF. As a result, our methodology does not accurately classify a bit whose ACE state changes in the presence of a fault in another bit. We refer to this behavior as *ACE interference*. Because ACE interference is not captured by the SDC ACE analysis, the calculated SDC MB-AVFs are only approximations.

To determine whether this is a significant source of error, we conducted a fault injection study and showed that ACE interference occurs rarely, therefore introduces negligible errors into the calculation of SDC MB-AVF. However, in our fault injection study, we only injected faults into the vector general purpose register file (VGPR) structure of the GPU. In this work, we extend this fault injection study to more structures of the GPU, including the local data share (LDS) and active mask stack (AMS). Unlike our previous work, we also look for two types of ACE interference. The first is when an ACE bit becomes unACE in the presence of one or more additional bit flips. The second case is when an unACE bit becomes ACE in the presence of one or more additional bit flips, which has not been studied in prior work [2].

Overall, our results further support the notion that ACE interference is rare in common workloads, and therefore that our methodology to calculate SDC MB-AVF using single-bit ACE analysis is accurate enough to support early design-time protection decisions.

The paper is organized as follows: Section II presents the terminology used in Architectural Vulnerability Factor analysis. Section III briefly presents the methodology for multi-bit AVF calculation as introduced in our previous work [2]. Section IV presents our experimental setup and Section V presents our results. We finally conclude in Section VI.

II. Background and Terminology

A *transient fault* arises from energetic particles generating electron-hole pairs as they pass through a semiconductor device. Transistors can collect these charges, and a sufficient amount of accumulated charge can invert the state of a logic device. This introduces a logical fault to the circuit's operation. Since this type of fault does not cause a permanent failure, it is referred to as a soft or transient fault [9].

A single particle strike can result in two different classes of faults: when it causes one single bit in a hardware structure to change state, the resulting fault is termed *single-bit fault*; when it affects multiple bits, the fault is termed a *spatial multi-bit fault* [2]. In this work, we focus on the spatial multi-bit faults where one particle strike leads to the flipping of multiple bits in a hardware structure.

Multi-bit faults can happen on any number of bits in any particular pattern. However, in this work, we focus on multi-bit faults in adjacent bits. We define *fault mode* as a specific multi-bit fault pattern (for example, we call a fault on 3 adjacent bits in a cache line a 3x1 fault mode). A *fault group* is a set of bits of a particular fault mode. For example, Figure 1 shows that there are three fault groups (G0, G1, and G2) for a 2x1 fault mode on a 4-bit x 1-bit structure.

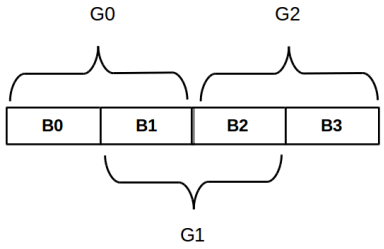

Fig. 1. A fault mode is a specific multi-bit fault pattern, and a fault group is a set of bits in a structure that match this pattern. For example, a 2x1 multi-bit fault mode has three unique fault groups (G0 through G2) in this 4x1 array (B0 through B3) [2].

A transient fault can yield various outcomes. We start by identifying non-error conditions, which include when the location of the fault is either: 1) not read by the running program, 2) read, but corrected by hardware error correction (e.g., error correcting codes), or 3) read, but has no effect on the program output (i.e., the output of the program in the presence of the fault is identical to the output in an error-free environment). These cases are all classified as unACE because they do not affect the final program output. We then classify the remaining error conditions as *Detected Unrecoverable Errors (DUEs)* or *Silent Data Corruptions (SDCs)*. These cases are both classified as ACE because the program will generate an incorrect output. DUEs happen in hardware structures that were designed with error detection (but not correction) schemes, such as parity. DUEs can be further divided into true DUEs and false DUEs. A false DUE is a detected error that would have been unACE if left undetected, whereas a true DUE is an error that would be ACE if left undetected. SDCs happen when an undetected fault induces the system to generate an erroneous output.

The fraction of the faults in a system that result in an erroneous output is the Architectural Vulnerability Factor (AVF) of the system. Consequently, the fraction of faults that become DUE is called the DUE AVF for the system, and the fraction of faults that result in SDC is the SDC AVF of the system.

III. Multi-Bit AVF Estimation

Although the focus of this work is not on the methodology of MB-AVF calculation, to keep our discussion self-contained, we provide a short description of the methodology developed in our previous work [2]. More information on the methodology can be obtained in our paper.

A. Definition and Methodology

The MB-AVF of a hardware structure for a particular multi-bit fault mode is defined as the fraction of faults of that

fault mode that result in a visible error in the final output of a program. Therefore, the MB-AVF over a period of N cycles for fault mode M of a hardware structure H containing $G_{H,M}$ unique fault groups of fault mode M can be expressed mathematically as follows:

$$MB\text{-}AVF_{H,M} = \frac{\sum_{n=1}^{N} [\text{ACE groups in H at cycle n}]}{G_{H,M} \times N} \quad (1)$$

B. Measurement Methodology

It is crucial to have MB-AVF estimation for both DUEs and SDCs. Without these estimates, a designer is forced to assume that all bits in a hardware structure are vulnerable and add protection accordingly. This is undesirable, as it adds unnecessary overhead in both area and power.

One method to perform MB-AVF estimation is ACE analysis. ACE analysis is conservative; thus, a designer can be confident that AVF estimates from ACE analysis are an upper bound on the true AVF. Therefore, under-designing protection is not a concern when using ACE analysis.

In ACE analysis for multi-bit AVF, fault groups are considered, as opposed to single-bit ACE analysis, which considers single bits. For multi-bit AVF, ACE analyses are performed on different fault modes independently, that is, we will perform perform different ACE analyses for 2x1, 3x1 and 4x1 fault modes. Using the principles above, MB-AVF can be calculated in a similar fashion to single-bit AVFs.

We extended a performance simulator-based ACE analysis infrastructure to measure MB-AVFs. A multi-bit fault group is considered ACE if any bit within that fault group is ACE. Therefore, a fault group with no ACE bits is an unACE group. Our prior MB-AVF work also considered protection schemes such as parity and ECC through the use of *protection domains*.

MB-AVFs can be calculated for both DUEs and SDCs. As discussed earlier, however, SDC MB-AVF measurement using ACE analysis is an approximation because it does not consider the effect of ACE interference. More details can be found in [2].

C. ACE interference

As previously mentioned, we did not model ACE interference in our estimation of the SDC MB-AVF. As such, our SDC MB-AVF estimation methodology is only valid if ACE interference does not have a major effect on AVF. ACE interference occurs when a bit changes ACE state in the presence of one or more other faults in the system.

There are two types of ACE interference: 1) an ACE bit becomes unACE in the presence of another fault; and 2) an unACE bit becomes ACE in the presence of another fault. An example of the first case is if a program performs an XOR operation on 2 bits. A fault in either one of the bits will lead to an incorrect output of the XOR operation; both bits are individually ACE. However, if both bits are flipped simultaneously, the XOR operation will have a correct output, as if there were no fault present (i.e., both bits are unACE). An example of the second case is if a program performs an AND operation on 2 bits, both of which have value 0. Both bits are individually unACE because a fault in either bit will still result in a 0 from the AND. However, a fault in both bits simultaneously will result in a 1 from the AND; thus, both bits are ACE when flipped simultaneously. Both of these cases break the assumption of multi-bit ACE analysis that the ACE state of a bit does not change in the presence of other faults in the system.

IV. Experimental Setup

Fault injection is conducted using an architectural simulator, Multi2Sim [10]. In Multi2Sim, we can inject errors during any cycle of the runtime of any hardware model. The fault mechanism is not specific to the microarchitecture and can be applied to different GPU architectures [11]. For this experiment, the AMD Radeon™5870 was used. A fault file with the fault information such as the fault mode, location, and time is fed to the simulator. A fault is represented by a bit flip in the simulated hardware structure at the specified cycle and location. The faulty value is potentially propagated to other locations or masked by the program.

To perform our ACE interference study, we first performed a round of at least 5000 random single-bit fault injections for each workload in order to identify single-bit ACE and unACE faults. Although our goal is not to statistically estimate the AVF of the workloads, we observed that, for most of our benchmarks, the AVF tends to show little variance after 5000 fault injections. We identify the faults that induce erroneous output by comparing the output of each run with that of a fault-free run. We then inject 2x1, 3x1, and 4x1 multi-bit faults in the identified ACE bits and adjacent bits. A correct output obtained with any of the multi-bit faults indicates an ACE interference with this ACE bit.

Next, we identify groups of all unACE bits (adjacent bits which, if flipped individually, will not cause an erroneous output) starting with the previously determined single-bit unACE faults. We then perform 2x1, 3x1 and 4x1 multi-bit injections on these fault groups. An incorrect output obtained with any of these fault groups indicates an instance of ACE interference.

We used workloads from the AMDAPP SDK benchmark suite [12]. We inject faults into three different structures of the GPU, the vector general-purpose register file (VGPR), the local data share (LDS) and the Active Mask Stack (AMS). These structures are large and occupy a substantial fraction of GPU die area in modern GPU processors [13].

V. Results

This section presents the results of our fault injection studies. These results include both the case of a single ACE bit becoming unACE in the presence of another fault as well as the case of multiple unACE bits becoming ACE when flipped together.

As the AVF for each hardware structure varies across the workloads, we selected workloads which had a relatively high number of single-bit ACE faults. The number of injections for

TABLE I
ACE INTERFERENCE IN VGPR MULTI-BIT FAULTS.

Benchmark	SDC ACE Bits	Multi-bit Fault Groups with ACE Interference			SDC un-ACE Bits	Multi-bit Fault Groups with ACE Interference		
		2x1	3x1	4x1		2x1	3x1	4x1
DCT	199	0	0	0	396	0	0	0
DwtHaar1D	12	0	0	0	500	0	0	0
FastWalshTransform	236	0	0	1	300	0	0	0
Histogram	300	0	0	0	300	0	0	0
MatrixMultiplication	300	0	0	0	300	0	0	0
MatrixTranspose	300	0	0	0	350	0	0	0
PrefixSum	300	0	1	0	300	0	0	0
RecursiveGaussian	47	0	0	0	300	0	0	0
ScanLargeArrays	36	0	0	0	300	0	0	0

TABLE II
ACE INTERFERENCE IN LDS MULTI-BIT FAULTS.

Benchmark	SDC ACE Bits	Multi-bit Fault Groups with ACE Interference			SDC un-ACE Bits	Multi-bit Fault Groups with ACE Interference		
		2x1	3x1	4x1		2x1	3x1	4x1
DCT	16	0	0	0	577	0	0	0
DwtHaar1D	9	0	0	0	800	0	0	0
Histogram	4800	0	0	0	182	0	0	0
MatrixMultiplication	68	0	0	0	700	0	0	0
MatrixTranspose	19	0	0	0	1860	0	0	0
PrefixSum	319	0	0	0	2979	0	0	0
RecursiveGaussian	47	0	0	0	100	0	0	0
Reduction	245	0	0	0	1000	0	0	0
ScanLargeArrays	35	0	0	0	2359	0	0	0

TABLE III
ACE INTERFERENCE IN AMS MULTI-BIT FAULTS.

Benchmark	SDC ACE Bits	Multi-bit Fault Groups with ACE Interference			SDC un-ACE Bits	Multi-bit Fault Groups with ACE Interference		
		2x1	3x1	4x1		2x1	3x1	4x1
DCT	54	0	0	0	75	0	0	0
DwtHaar1D	5	0	0	0	975	0	0	0
Histogram	12	0	0	0	1	0	0	0
MatrixMultiplication	99	0	0	0	1000	0	0	0
MatrixTranspose	39	0	0	0	237	0	0	0
PrefixSum	24	0	0	0	900	0	0	0
RadixSort	12	0	0	0	3	0	0	0
Reduction	21	0	0	0	500	0	0	0
ScanLargeArrays	5	0	0	0	458	0	0	0

each benchmark/structure was chosen based on the number of ACE bits and unACE bits identified after the single-bit injections.

In our experiments, we observe that the second type of interference is extremely rare. An unACE bit is very likely to remain unACE in the presence of other faults, unless these other faults were ACE to begin with.

A. Vector General Purpose Register File (VGPR)

We performed an ACE interference study in the vector register file of the GPU. The results are presented in Table I. Out of 1730 injections into single-bit SDC ACE locations, we observe only 2 cases of interference, in FastWalshTransform and PrefixSum. We found no cases of interference in single-bit unACE locations. For the cases of interference, while both the single-bit and the multi-bit faults caused the program to diverge from its execution path, in the case of the multi-bit faults, the program converged back to its correct execution path a few hundred cycles later. This was not the case for the single-bit faults. These were essentially occurrences of *Y branches*: branches that do not affect correct program behavior when forced down the incorrect path [14].

Overall, only 0.1% of fault groups showed ACE interference, indicating that ACE interference is not a common occurrence in the GPU register file.

B. Local Data Share (LDS)

The results of injections into the local data share (LDS) are shown in Table II. This study on the local data share shows that interference is not an issue in common GPU workloads.

One particularly interesting case was ScanLargeArrays. This benchmark shows no ACE interference when we perform bit-by-bit comparisons of the output of each run to the golden run. However, if we instead compare the output of each run to the golden run using a threshold criterion (i.e., a small difference in numerical results is acceptable), we found that, this benchmark exhibits a significant amount of interference, where single ACE bits would seem to become unACE in the presence of adjacent faults. These faults occurred at the exponent fields of floating point values contained in the local

data share and caused these values to pass the self-checking mechanism of this benchmark. More than 75% of our 4x1 injections led to output values close enough to be considered correct within this threshold. This makes an interesting case for future work to analyze ACE interference on approximate computing workloads, where approximate values can be treated as correct output.

C. Active Mask Stack (AMS)

The Active Mask Stack is a structure that influences the control flow of a program. Faults in the AMS can result in computing extraneous code or wrongly continuing the execution of a basic block. For this reason, we expect that if a single-bit fault in the AMS induced an erroneous output, it will almost certainly always produce an erroneous output, whether or not another faulty bit is present. The results in Table III confirm our assumptions.

VI. Conclusion and Future Work

As multi-bit faults are receiving more attention from researchers, it becomes more crucial to agree on standards and methodology to evaluate the vulnerability of hardware structures in the presence of these faults.

In this paper, we confirmed the methodology we previously presented [2] to estimate Multi-Bit AVFs is accurate. To do this we expanded the fault injection study we had done previously on the register file to include additional structures of a GPU.

Our results show that ACE interference is a rare event in GPUs: we found that an SDC ACE bit rarely becomes unACE in the presence of other faults in the system, and vice versa, an unACE bit rarely becomes ACE in the presence of another unACE bit. Thus, ACE interference will have a negligible impact on SDC MB-AVF estimates. Our results further support prior work that asserts that ACE analysis can be used to produce accurate SDC MB-AVF values, and so can be used to guide early design-time protection decisions.

Future work includes generalizing these results to approximate computing workloads and other microarchitectures, including different GPU microarchitectures and to CPUs.

References

[1] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *Electron Devices, IEEE Transactions on*, vol. 57, no. 7, pp. 1527–1538, July 2010.

[2] M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. R. Kaeli, "Calculating architectural vulnerability factors for spatial multi-bit transient faults," in *Microarchitecture, 2014. MICRO-47. Proceedings. 47th Annual IEEE/ACM International Symposium on*, Dec 2014.

[3] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 305–316, Sept 2005.

[4] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *IEEE Computer*, vol. 30, no. 4, pp. 75–82, Apr. 1997.

[5] C. Constantinescu, M. Butler, and C. Weller, "Error injection-based study of soft error propagation in amd bulldozer microprocessor module." in *DSN*, R. S. Swarz, P. Koopman, and M. Cukier, Eds. IEEE Computer Society, 2012, pp. 1–6. [Online]. Available: http://dblp.uni-trier.de/db/conf/dsn/dsn2012.html

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

[7] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 502–506.

[8] H. Cho, S. Mirkhani, C.-Y. Cher, J. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, May 2013, pp. 1–10.

[9] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: an architectural perspective," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, Feb 2005, pp. 243–247.

[10] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," in *Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2012.

[11] R. Ubal, D. Schaa, P. Mistry, X. Gong, Y. Ukidave, Z. Chen, G. Schirner, and D. Kaeli, "Exploring the heterogeneous design space for both performance and reliability," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 181:1–181:6. [Online]. Available: http://doi.acm.org/10.1145/2593069.2596680

[12] AMD, "OpenCL accelerated parallel processing (APP) SDK," http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/downloads/, 2013.

[13] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron, "Real-world design and evaluation of compiler-managed gpu redundant multithreading," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 73–84. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665686

[14] N. Wang, M. Fertig, and S. Patel, "Y-branches: when you come to a fork in the road, take it," in *Parallel Architectures and Compilation Techniques, 2003. PACT 2003. Proceedings. 12th International Conference on*, Sept 2003, pp. 56–66.