Typhoon: Enabling GPGPU Application Resilience Estimation with Different Input Types

Lishan Yang
William & Mary
Williamsburg, VA
lyang11@email.wm.edu

***Abstract*—Graphics Processing Units (GPUs) are embraced in various domains because of their massive parallelism to process large amounts of data. Reliability concerns are raised as critical applications such as self-driving cars are deployed on GPUs. While different techniques are proposed to measure the resilience of a GPGPU application on one input, the reliability assessment has to be repeated for every different input. In this paper, we propose a methodology, *Typhoon*, which uses the norm of the input matrix to project application resilience. We observe an approximately monotonic trend between resilience and the norm of the input matrix. By sampling several selected input, the trend is captured and used to estimate the resilience of any new input.**

I. Introduction

The growing demand for large scale and fast computation embraces massive parallelism and multicore architecture, which contributes to the popularity of Graphics Processing Units (GPUs). General Purpose GPUs (GPGPUs) are widely used in different domains such as autonomous vehicles [1], high performance computing (HPC) [2], [3], [4], [5], [6], [7], deep learning [8], and network functions [9]. A large number of GPGPU applications are critical to errors, such as hurricane prediction and self-driving cars. Similar to CPU and other customized accelerators, GPUs are susceptible to transient faults [10], [11], [12], [13], [14]. This raised the importance of GPU reliability research, including GPU application resilience analysis and error detection/protection mechanisms.

One of the common ways of assessing GPGPU application resilience is to perform a fault injection campaign. For each application fault injection run, one or multiple bits are flipped, and then the final output of the application is compared with the fault-free execution. Because of the large number of threads executing in parallel, the fault site space of a GPGPU application can reach the order of billions [15]. Statistical sampling is used to reduce the required number of experiments [16], [17]: the results of $1,000$ experiments can achieve 95% confidence intervals and $\pm 3\%$ error margins. Even with the state-of-art fault site pruning method [15] to reduce the sampling fault site space, several hundreds to thousands of experiments are necessary to evaluate the application resilience. All of the above methods focus on only one input. Since the input of real world applications cannot always stay the same, applying resilience estimation to real world applications is challenging. If input value changes, the resilience may also change, and the estimation process needs to be repeated. The state-of-art resilience estimation technique [18] considers different inputs, but only with the same *input type* (the actual input that the application admits, such as integers or floats and their range of values) and scaling up the *input size* (the actual number of input elements).

When considering different input types, the problem is much more complicated than considering different input sizes: when input size changes, only the thread structure of the application scales accordingly, while analyzing different input types should consider different data type, range of values, distributions, and the sign of the value. In this work, we propose *Typhoon*, a methodology to estimate the GPGPU application resilience considering different input types. By examining various inputs and their resilience, we find the *norm* of the input matrix is a good abstraction of the input characteristics. The relationship between resilience and norm is approximately monotonic. Therefore, fault injection campaigns targeting several selected input types covering different norm values can capture this approximately monotonic trend. For *any* new input type, its application resilience can be easily retrieved from the captured trend.

In summary, we make the following research contributions:

- We identify the usefulness of input matrix norm when dealing with different input types.
- We propose a methodology to estimate GPGPU application resilience with different input types.

II. Background and Fault Model

A. GPU structure and input format

Baseline GPU Architecture. A GPU has a large number of cores (streaming-multiprocessors or SMs in NVIDIA terminology [19]), with private L1 cache, software-managed scratchpad memory, and a register file in each core. Cores are connected to global memory through an interconnection network. The L2 cache is shared within the same memory channel.

GPGPU Execution Model. GPGPU applications execute thousands of threads in parallel over large amounts of data. The logic of a GPGPU application is organized as a sequence of kernels (GPGPU functions). Each kernel is partitioned into groups of threads, known as *thread blocks* or *Cooperative Thread Arrays* (CTAs) in NVIDIA terminology. Threads inside one CTA are grouped into warps (32 individual threads per warp) as the smallest scheduling unit.

Input Format and Benchmarks. The input of GPGPU applications can be matrices, vectors, images, and graphs, and can be represented in float points, integers, or booleans. In this paper, we focus on integers. We select four benchmarks from Polybench [20]: MVT (matrix-vector multiplication), 2DCONV (2D convolution), GEMM (general matrix multiplication), and 3MM (three matrices multiplication).

B. Fault Model

The fault model assumes that register files and other components such as caches and memory are protected by ECC, and we only consider commonly occurring computation-related errors due to transient single-bit faults (also known as soft errors) in components that cannot be protected by ECC, such as ALUs and LSUs, which leads to wrong outputs in destination registers. This is standard experimental methodology for GPGPU reliability studies [16], [17], [21], [15], [22].

For each fault injection run, we flip a bit at a destination register identified by its instruction id and a bit position. This is commonly used in GPU application resilience estimation [17], [23], [16], [15], [18]. We consider single-bit fault model, i.e., in each fault injection run, there is only one bit flip.

We use NVBit-FI [24] to perform fault injection experiments. NVBit-FI is an architecture-level fault injection tool built on top of NVBit [25] (a dynamic binary instrumentation library of NVIDIA GPUs).

There are three possible outcomes for one fault injection run: 1) **masked** outcome: the application output is correct; 2) **silent data corruption (SDC)** outcome: the output is incorrect, but the execution is successful without any error; 3) **other** outcome: a crash or hang is observed.

For each application input, we perform $1,000$ fault injection experiments with random sampling to achieve 95% confidence intervals and $\pm 3\%$ error margins. We calculate the percentage of **masked**, **SDC**, and **other** outcomes of fault injection experiments to get the application *resilience profile*. In this work, we focus on the benign outcomes, i.e., the percentage of **masked** outcomes.

III. Methodology and Preliminary Results

We start with a motivation example with MVT. We generate 15 different inputs with Binomial and Equilikely distributions, as listed in Table I. The *Frobenius norm* [26] of the input matrix A in the third column is calculated using Eq. 1:

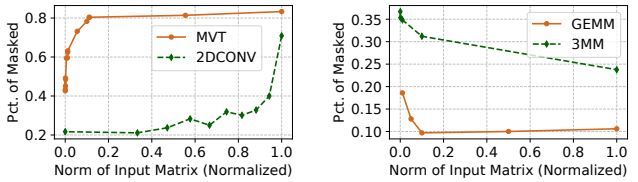
$$\| A \|_{Frob.norm} = \sqrt{[\Sigma_{i,j} abs(a_{i,j})]^2}, \quad (1)$$

where function $abs(a_{i,j})$ calculates the absolute value of matrix element $a_{i,j}$.

We plot the relationship of resilience (as a function of the percentage of masked outcomes obtained from fault injection experiments) to the norm value of input matrices, see the solid line in Fig. 1(a). There is an approximate monotonic trend: as the norm increases, the percentage of Masked outputs also grows larger. There is no distinct difference among different distributions and parameters, and all the characteristics of the input matrix are summarized using its norm.

TABLE I
Input examined in MVT.

Distribution	Parameters	Frob. Norm
Binomial $P(x) = \binom{N}{x} p^x (1-p)^{n-x}$	$p = 0.1, n = 10$	356.00
	$p = 0.1, n = 100$	2692.62
	$p = 0.1, n = 1000$	25907.79
	$p = 0.1, n = 10000$	258096.24
	$p = 0.5, n = 10$	1351.48
	$p = 0.5, n = 100$	12965.70
	$p = 0.5, n = 1000$	129053.56
	$p = 0.5, n = 10000$	1289958.94
	$p = 0.9, n = 10$	2333.20
	$p = 0.9, n = 100$	23229.20
	$p = 0.9, n = 1000$	232204.45
	$p = 0.9, n = 10000$	2321934.89
Equilikely $P(x) = \frac{x-a}{b-a}$	$a = 10, b = 20$	3811.68
	$a = 100, b = 110$	26968.00
	$a = 1000, b = 1010$	259151.83



(a) MVT and 2DCONV. (b) GEMM and 3MM.

Fig. 1. Resilience as a function of norm.

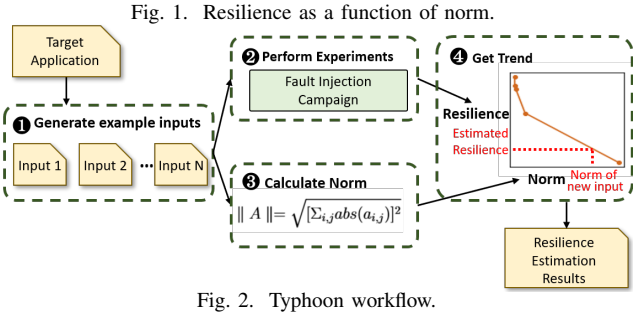


Fig. 2. Typhoon workflow.

Similar observations are also shown in other benchmarks from Polybench in Fig. 1. The approximate monotonic trends for different applications are different: the trends of MVT and 2DCONV are both monotonous increases, while in GEMM and 3MM they are monotonous decreases. Note that the oscillations observed in 2DCONV and GEMM are within the error margin.

We summarize the workflow of *Typhoon* in Fig 2. For the target application to be estimated, the first step is to generate example inputs covering the desired range of matrix norm. In step 2, fault injection experiments are conducted to obtain the resilience of the selected example inputs. Meanwhile, the Frobenius norm of the input matrices is calculated in step 3. With the resilience and norm values, we can determine the approximately monotonic trend of the target application, see step 4. Given a new input, the norm of the input matrix is calculated, and the estimated resilience is retrieved from the derived function of norm-resilience.

Typhoon drives the monotonic trend between resilience and input matrix norm, which is used to estimate the resilience of new inputs. Here we only consider benchmarks performing matrix operations and integers. Other facts such as floating points or negative values, mixes of different distributions, and other types of applications are subject of ongoing work.

REFERENCES

[1] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 586–597, IEEE, 2018.

[2] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU–past, present and future," *Medical image analysis*, vol. 17, no. 8, pp. 1073–1094, 2013.

[3] S. S. Stone, J. P. Haldar, S. C. Tsao, W. mei W. Hwu, B. P. Sutton, and Z.-P. Liang, "Accelerating advanced MRI reconstructions on GPUs," *J. Parallel Distrib. Comput.*, vol. 68, no. 10, pp. 1307–1318, 2008.

[4] R. Foster, "How to harness big data for improving public health," *Government Health IT*, 2012.

[5] I. Schmerken, "Wall street accelerates options analysis with GPU technology," *Wall Street Technology*, vol. 11, 2009.

[6] NVIDIA, "Computational finance."

[7] NVIDIA, "Researchers deploy GPUs to build world's largest artificial neural network."

[8] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: an efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–23, 2019.

[9] Q. Gong, P. DeMar, and W. Wu, "Deep packet/flow analysis using gpus," tech. rep., 2017.

[10] B. Nie, D. Tiwari, S. Gupta, E. Smirni, and J. H. Rogers, "A large-scale study of soft-errors on gpus in the field," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 519–530, IEEE, 2016.

[11] B. Nie, J. Xue, S. Gupta, C. Engelmann, E. Smirni, and D. Tiwari, "Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities," in *MASCOTS 2017*, pp. 22–31.

[12] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for GPU error prediction in a large scale HPC system," in *DSN 2018*, pp. 95–106.

[13] S. Jha, S. S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*, pp. 112–124, IEEE, 2019.

[14] A. Mahmoud, N. Aggarwal, A. Nobbe, J. Vicarte, S. Adve, C. Fletcher, I. Frosio, and S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," pp. 25–31, 06 2020.

[15] 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)*, pp. 749–761, IEEE, 2018.

[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*, pp. 221–230, IEEE, 2014.

[17] 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.

[18] L. Yang, B. Nie, A. Jog, and E. Smirni, "Sugar: Speeding up gpgpu application resilience estimation with input sizing," *Proc. ACM Meas. Anal. Comput. Syst. (Sigmetrics 2021), to appear*, 2021.

[19] "NVIDIA Fermi Architecture Whitepaper."

[20] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *Innovative Parallel Computing (InPar), 2012*, pp. 1–10, IEEE, 2012.

[21] 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*, pp. 240–251, IEEE, 2016.

[22] B. Sangchoolie, K. Pattabiraman, and J. Karlsson, "One bit is (not) enough: An empirical study of the impact of single and multiple bit-flip errors," in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*, pp. 97–108, IEEE Computer Society, 2017.

[23] 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*, pp. 90–100, IEEE, 2016.

[24] "Nvbitfi." https://github.com/NVlabs/nvbitfi.

[25] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 372–383, 2019.

[26] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 1985. pg. 15.