LAD-ECC: Energy-Efficient ECC Mechanism for GPGPUs Register File

Xiaohui Wei, Hengshan Yue, Jingweijia Tan

College of Computer Science and Technology, Jilin University, Changchun, China, 130012
Emails: weixh@jlu.edu.cn, yuehs18@mails.jlu.edu.cn, jtan@jlu.edu.cn

***Abstract*—Graphics Processing Units (GPUs) are widely used in general-purpose high-performance computing applications (i.e., GPGPUs), which require reliable execution in the presence of soft errors. To support massive thread level parallelism, a sizeable register file is adopted in GPUs, which is highly vulnerable to soft errors. Although modern commercial GPUs provide single-error-correction double-error-detection (SEC-DED) ECC for the register file, it consumes a considerable amount of energy due to frequent register accesses and leakage power of ECC storage.**

In this paper, we propose to *L*everage *A*pproximation and *D*uplication characteristics of register values to build an energy-efficient *ECC* mechanism (LAD-ECC) in GPGPUs, which consists of *AP*roximation-aware *ECC* (AP-ECC) and *D*uplication-*A*ware *ECC* (DA-ECC). Leveraging the inherent error tolerance features, AP-ECC merely protects significant bits of registers to combat the critical error. Observing same-named registers across threads usually keep the same data, DA-ECC avoids unnecessary ECC generation and verification for duplicate register values. Experimental results demonstrate that our LAD-ECC tremendously reduces 69.72% energy consumption of traditional SEC-DED ECC.

***Index Terms*—GPGPUs, Reliability, Soft Error, Energy-Efficiency**

I. Introduction

With their remarkably high computational throughput and good programmability, Graphics Processing Units (GPUs) have been increasingly deployed in high-performance computing (HPC) fields recently (e.g., scientific computing, autonomous vehicles, and financial application) [1]–[4], which is also called General-Purpose computing on GPUs (GPGPUs). The extensive usage of GPUs makes reliability a critical concern. As the shrinking of transistor feature size and growing of chip integration density, GPGPUs are vulnerable to particle strikes [5]–[7], which will cause bit-flips in the stored data, called soft errors.

Each thread is allocated with some dedicated physical registers to support massive parallel data processing, which requires a sizeable register file in GPGPUs. In addition, since register file holds contexts and architectural states of threads, errors in it are highly possible to spread to other GPU structures [5], [8]. Previous work shows register file is one of the soft error reliability bottleneck in GPGPUs [5], [7], [9].

To combat the impact of soft errors, modern commercial GPUs protect their register file with single-error-correction double-error-detection (SEC-DED) ECCs [7], [10]. However, implementing full ECC coverage for such a large register file causes considerable energy and area overhead because of the frequent data access. Every register write generates ECC and read verifies ECC, which consumes large amount of energy. Besides, the leakage energy of ECC storage is nontrivial as well [5], [8]. Previous work [5] found that ECC additionally consumes about 38% energy of register files, which is energy-inefficient.

In this work, we observe two key characteristics of GPGPUs register file, which can be used to effectively reduce the energy consumption of soft error protections. First, we find that some GPGPU applications, such as physical simulations, are resilient to imprecise outcomes [3], [4]. We observe errors occurring in low bits of mantissa have negligible impact on program execution correctness, which indicates merely protecting the significant bits of registers is able to avoid critical errors while tolerating some acceptable errors. Second, since threads within a warp may operate with the same constant or with duplicate controls, same-named registers across threads usually keep the same values [5], [11]. We observe 44.2% of warp instructions write duplicate values to register files on average, which implies generating ECC for one copy of the value provides full protection for all duplicate values.

Based on these two observations, we propose to *L*everage *A*pproximation and *D*uplication characteristics to build an energy-efficient *ECC* mechanism for GPGPUs register file (LAD-ECC), which consists of *AP*roximation-aware *ECC* (AP-ECC) that selectively protects significant bits to combat the critical errors and *D*uplication-*A*ware ECC (DA-ECC) that avoids redundant ECC generation and verification. After exhaustively analyzing the error-sensitivity for all data bits, AP-ECC only generates ECC for the sign bit, exponent field, and 8 mantissa bits (i.e., 15-31 bits) of a 32-bit register. According to the encoded duplication operands information in the instruction, DA-ECC avoids unnecessary ECC generation and verification by reusing ECC among duplicate register values. We summarize the contributions of this study as follow:

- Considering the approximation characteristics of programs, we propose AP-ECC to improve the efficiency of

Corresponding author: Jingweijia Tan (jtan@jlu.edu.cn)

This work is supported by the National Natural Science Foundation of China (NSFC) (Grants No.61772228, No.61802143), National key research and development program of China under Grant No.2017YFC1502306, Jilin Scientific and Technological Development Program (Grants No.20180101046JC, Grants No.20190701016GH), Research Project by the Education Department of Jilin Province under Grant No. JJKH20190159KJ, and Graduate Innovation Fund of Jilin University under Grant No.101832018C026.

ECC in GPGPUs register file by neglecting insignificant errors during protection.

- By reusing the ECC generation and verification processes, we propose DA-ECC that significantly reduces ECC energy consumption of duplicate values in GPGPUs register files without losing any error coverage.
- Building DA-ECC on top of AP-ECC, we form a unified cost-effective ECC mechanism for GPGPUs register file, LAD-ECC. Experimental results show LAD-ECC tremendously reduces 69.72% energy consumption of SEC-DED ECC.

II. Background

A. GPGPUs Architecture and Register File

The primary components of GPGPUs are Single Instruction Multiple Thread (SIMT) cores. We adopt the NVIDIA CUDA programming model in this work but our technique is applicable to other GPGPUs as well.

GPGPUs execute highly-parallel kernel functions launched by the CPU. A kernel is composed of thousands of lightweight threads. These threads are group into a set of blocks, which are the granularities of threads distributions. Multiple blocks are allocated to the same SM as long as there are enough resources. Inside an SM, 32 individual threads are grouped together, called a warp. Warp is the basic unit for instruction scheduling and issuing, where all threads within a warp are strictly executed in the Single Instruction Multiple Data (SIMD) manner. Before the execution of a warp instruction, it first accesses the register file to obtain operand values for all 32 threads. One instruction may read up to 4 registers while writing 1 register at the same time in the NVIDIA Parallel Thread eXecution (PTX) assembly code standard. Thus, a large bandwidth of 32×4 operand values is required.

To provide large bandwidth with low complexity, register file in GPGPU is highly-banked (e.g., 32 banks) for parallel accesses [5], [12]. Fig.1 shows the baseline register file architecture in this work. The 128KB register file per SM is composed of 32 banks, each bank is 4KB. To support 1 read and 1 write per cycle, each bank contains two ports. There are 256 entries per bank, each of which contains four 32-bit registers. All same-named registers in a warp, also called warp-register, are allocated to the same register entries of 8 consecutive banks. In the case of bank conflict, several accesses to the same bank have to be serialized. To avoid the extra access latencies caused by bank conflicts, operand collectors are adopted to buffer the operands from register files. When all operands are available in an operand collector, the warp instruction is ready to be sent to SIMD Execution Units.

B. SEC-DED ECC in GPGPUs Register File

Single-error-correction double-error-detection (SEC-DED) ECC is adopted for soft error protection in the register file of modern commercial GPGPU products. Fig.2 shows its implementation, which mainly adds three types of hardware

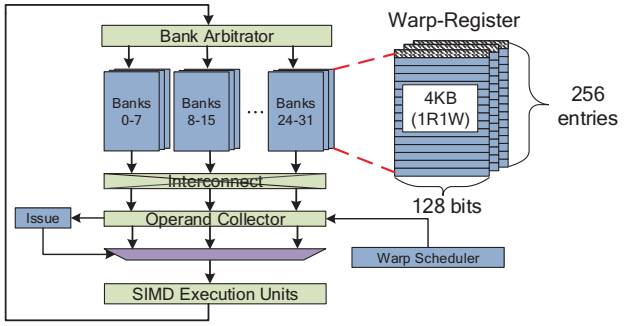

Fig. 1: The structure of register file in a SIMT core.

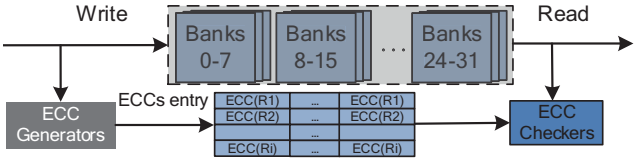

Fig. 2: The implementation of SEC-DED ECC.

components to register file: an ECC table that stores the ECCs for register values, ECC generators, and ECC checkers.

The ECC generation is triggered during the writeback stage of GPU pipeline. First, ECC generators calculate 7-bit ECC for each 32-bit register in parallel. Next, ECCs of the warp-register are sent to the corresponding entry in the ECC table. When a warp-register is read by warp instruction, it is sent to the ECC checkers simultaneously with the data stored in ECC table entry for error detection and correction.

III. *LAD-ECC*: Leveraging Approximation and Duplication Characteristics to Build a Cost-Effective *ECC* Mechanism

We propose LAD-ECC in this section, which *L*everages the *A*pproximate features of programs and the *D*uplicate characteristics in warp-registers to build a unified cost-effective *ECC* mechanism in GPGPUs register file. LAD-ECC is the merge of *AP*proximation aware *ECC* (AP-ECC) and *D*uplication *A*ware *ECC* (DA-ECC). AP-ECC only protects significant bits in order to combat critical errors, and DA-ECC avoids unnecessary ECC generation and verification by reusing ECC in duplicate warp-registers.

A. AP-ECC: Approximation-Aware ECC Mechanism

1) Error-Sensitivity of Data-bits in GPGPUs Register File: In IEEE 754 data representation standard, bit 31 is the sign and 23-30 bits represent the exponent field in 32-bit floating-point data. Thus these bits are crucial for the correct execution of programs [13]. The remaining bits (0-22 bits) are mantissa for the floating-point number, and we observe that errors occurring in the lower few bits of them have insignificant effects on execution correctness. Thus if the precision of the applications is relaxed, such insignificant errors are tolerable [3], [4].

For example, Fig. 3 shows the effect of errors in each bit of register value under fault injections for `covariance`, a data

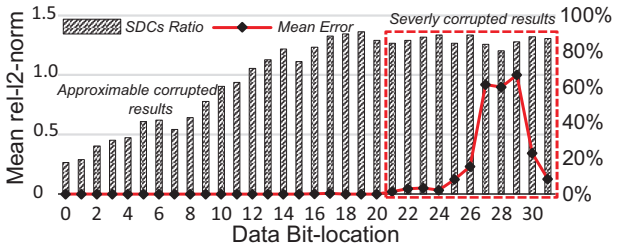

Fig. 3: Mean error rate distribution and SDCs ratio among bit-locations.

mining benchmark from Polybench [14]. First, we observe that errors occurring in the high bits (i.e., bit 21-31) are easy to incur Silent Data Corruptions (SDCs), which require careful attention in practice because there is no indication that the application has produced erroneous results [9], [15]. Besides, errors occurring in these bits cause large deviations from standard system outputs. Thus, we refer these high bits as significant bits and pay more attention to them when applying accurate protections for registers. Next, we observe that for errors occurring in 0-20 bits, the SDCs' differences with error-free results are insignificant. Such errors have negligible impacts on final results, thus only protecting the significant bits (i.e., 21-31 bits) of `covariance` would achieve near-full protection.

The mean error rates are low for errors occurring in bits 30 and 31. The reasons are: (1) data usually flip from "1" to "0" in bit 30 in `covariance`, which causes less relative error of register value than flipping '0" to "1" [1]; (2) error occurring in the sign bit (i.e., bit 31) has less effect on data correctness than the exponent field.

These observations guide us to leverage selective soft error protections to improve the energy-efficiency of ECCs in GPGPUs register file. The detailed error-sensitivity analysis for other benchmarks will be exhibited in Section V-A.

2) AP-ECC and Its Implementation: Based on the observations in Section III-A1, we protect the sign bit and all exponent bits for each register. Besides, according to the sensitivity analysis in Section V-A, we also harden 8 mantissa offsets (i.e., 15-22 bits) of each register.

Fig. 4 shows the implementation of AP-ECC at the ECC generation stage. Based on the content of significant bits in every register, we design a simpler "Parity Bits Generation" logic than full-ECC technology to generate 6-bit ECC for these protected bits (i.e., 15-31 bits). Hamming code is adopted for parity bits generation, which can be implemented via simple XOR trees. At ECC checking stage, these ECC bits will be used for error detection and correction. Similar to the generation step, AP-ECC only uses the ECCs to detect and correct errors of significant bits, while errors occurring in other bits are ignored.

Since fewer bits are protected comparing to the baseline SEC-DED ECC mechanism, AP-ECC incurs less energy and area overhead while guaranteeing protections against significant errors.

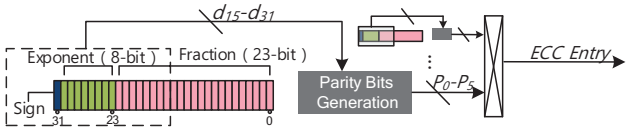

Fig. 4: The implementation of AP-ECC at ECC generation stage.

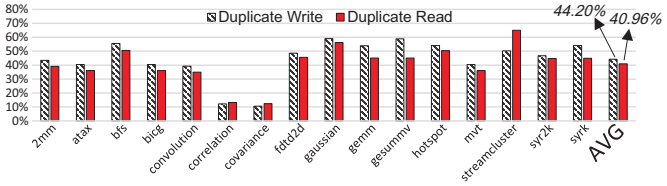

Fig. 5: Percentage of duplicate values warp-registers.

B. DA-ECC: Duplication-Aware ECC Mechanism

1) Duplicate Values in Warp-Register: All threads within warp access the register file simultaneously. As an interesting characteristic, sometimes all same-named registers contain the same value during program execution [11]. This is because different threads may operate on the same constant, such as the block dimension, which is stored independently in registers of all threads within the same block. Also, threads within a warp may contain duplicate control instructions, which is also redundant. Fig. 5 shows the ratio of duplicate read and write values in warp-registers. On average, 44.20% of warp instructions write redundant values. In addition, 40.96% of warp read operations load duplicate values from the register file. The discrepancy is because a write data may be read multiple times by the subsequent instructions. These observations motivate us to avoid generating and checking duplicate ECCs in GPGPUs register file for energy saving.

2) DA-ECC and Its Implementation: During compilation, the duplicate values are detected via the data dependence graph reachability analysis proposed by Chen *et al* [11]. Fig. 6 shows a simple example of the analysis process. We first mark all apparently divergent register variables (e.g., thread IDs) in the data flow graph. Next, all the reachable registers are marked as divergent (black circles) while the others are marked as duplicate (white circles). Then, according to the number of operands in each instruction, we encode the duplication information using the corresponding number of extra bits, where the highest one represents the write operand. The bit is set to "1" ("0") when the corresponding operand is duplicate (divergent). For instance, the three extra bits for the last `add` instruction indicates that it reads duplicate r_2 and divergent r_3, then writes a divergent variable to r_4.

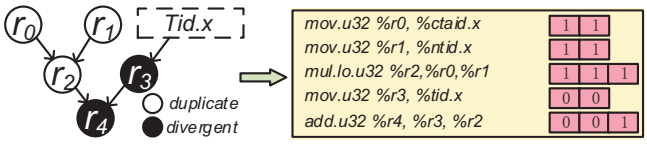

Fig. 6: Duplicate variable detection and extra information bits for the instruction.

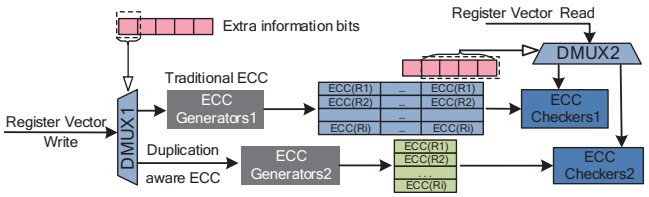

Fig. 7: The implementation of DA-ECC.

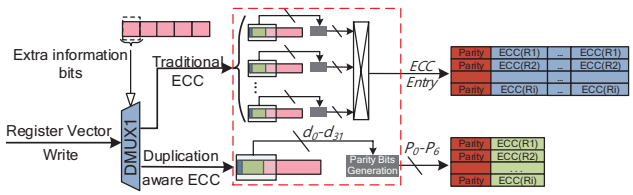

Fig. 8: The implementation of LAD-ECC at ECC generation stage.

Fig. 7 depicts the implementation of DA-ECC. At writeback stage, a demultiplexer (i.e., DMUX1) is used to determine whether it is a duplicate register write based on the highest bit of the duplication information. If "1" is detected by DMUX1, DA-ECC only generates ECC for the first valid thread's written register. In this case, only this thread's ECC is valid in the corresponding ECC entry. We power gate other redundant ECC fields in this entry for leakage energy saving. Otherwise, DA-ECC generates ECC for each same-named register as in the baseline architecture.

For register read, DMUX2 is used to determine whether to perform duplicate register verification based on duplication information of the instruction. For divergent reads, the ECCs are obtained from ECC table and sent to the checkers for verification as in the baseline. Otherwise, DA-ECC only performs ECC verification for the first valid thread's source register. However, an error may occur in other threads, which would affect the correct execution of the program. To tackle this issue, we force all threads within a warp to read the first valid thread's register value for a duplicate operand.

Leveraging the value duplication characteristics in the warp-register, DA-ECC avoids unnecessary ECC generation and verification, which significantly reduces the energy overhead of ECC in GPGPUs register file without losing any error coverage.

C. LAD-ECC: Merging Them Together

AP-ECC and DA-ECC can be combined for a unified energy-efficient ECC mechanism in GPGPUs register file. Fig. 8 shows the implementation of LAD-ECC at ECC generation stage, which replaces the ECC generators in DA-ECC with AP-ECC. The implementation of ECC verification is similar to the generation stage.

The ECC table is also vulnerable to soft errors, which may cause incorrect data verification. Thus in each ECC entry, we add a parity bit for the ECCs. During the data verification stage, we first check the parity bit to ensure the correctness of ECCs. If an error is detected by the parity bit, it indicates the corresponding ECCs are invalid. In this case, LAD-ECC ignores the data verification process and keep the program running. Due to the nature of soft error, the probability of errors occurring in both register variable and the corresponding ECC entry is extremely low. When the write operation occurs in the corresponding warp-register next time, the incorrect ECC value will be replaced. Besides, the demultiplexer, ECC generator, and ECC checker are also vulnerable to soft errors, we adopt gate-sizing technique to ensure the data correctness in these hardware components. Due to their small sizes, it introduces negligible area overhead.

IV. Experimental Methodology

A. Benchmarks and Quality Metrics

To evaluate our *LAD-ECC* mechanism, we execute a wide range of benchmarks from Rodinia [16], Polybench [14], and GPGPU-Sim [17] in our experiments. To quantitatively analyze quality degradations under soft errors, we choose two efficient metrics as our quality metrics: relative error and relative l2-norm. Relative error calculates the normalized absolute error between the error-free outcomes and soft error corrupted outcomes. Similarly, we calculate the normalized l2 difference to represent the quality degradation of the corrupted matrix, which accumulates element-wise square root differences of the corrupted and error-free matrices.

B. Experiment Setup

We use a cycle-accurate simulator GPGPU-Sim (v4.0.0) [17] in our experiments. The detailed microarchitecture configuration of the baseline GPGPUs are as follows: the warp size is 32; there are 24 SMs; each SM supports 2048 threads, 64 warps, and 32 blocks at most; each SM contains 128KB register file. We use HSPICE [18] to estimate the dynamic and static power of ECC logic. The numbers of ECC generations and verifications are collected from the modified GPGPU-Sim. We use CACTI [19] to estimate the power and area overhead of the ECC table.

To accurately simulate soft error occurrence, we choose SASSIFI [9] as our error injection tool, which is able to inject errors into architecture-visible states. For each program, we inject 1000 errors to each bit offset, which have maximum error bars of 3.1% at the 95% confidence level [9], [15].

V. Evaluation

A. Finding the Significant Mantissa Bits

AP-ECC protects the sign and exponent bits by default due to their importance on program execution correctness. We further protect some bits in mantissa for better reliability. Fig. 9(a-o) plots the errors under various numbers of protected mantissa for each benchmark[1]. Take `syr2k` as an example, if the largest error (relative l2-norm) we can tolerate is 10^{-2}, we need to ensure the correctness for the upper 6 bits of mantissa at least.

[1]Errors occurring in `streamcluster` usually cause serious outcomes (e.g., hang or crash), hence, we don't apply AP-ECC for it.

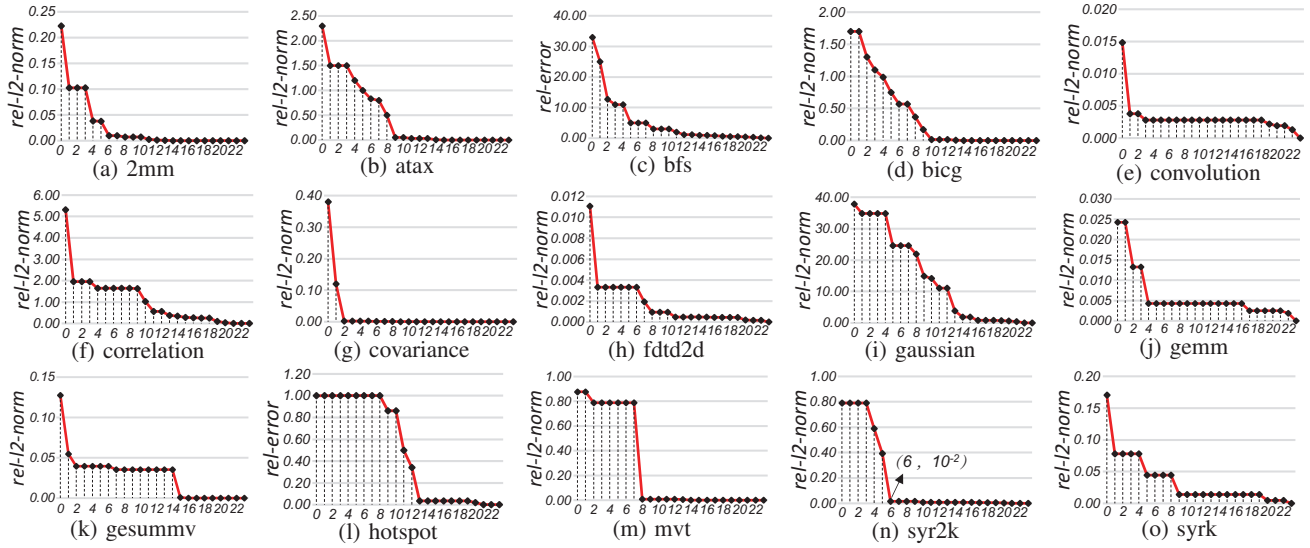

Fig. 9: The errors under different protected mantissa number. The 0 (23) in the X-axis represents the architecture with no (full) protection for the mantissa.

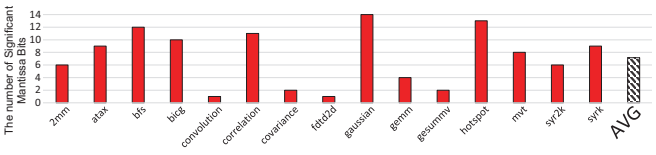

Fig. 10: The number of protected mantissa.

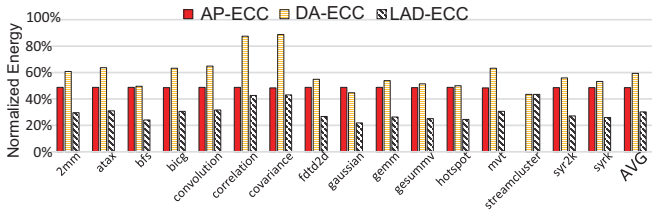

Fig. 11: Normalized energy overhead.

According to the error distribution and program reliability characteristics, Fig.10 shows the reasonable number of significant mantissa for each program. On average across all benchmarks, the number of the protected mantissa is 7.15, which imply that we can protect fewer mantissa bits by slightly relaxing the precision requirement of the program. We adopt a fixed set of ECC generation and ECC verification logic to ensure the correctness of the upper 17-bits (i.e., sign, exponent and 8 mantissa bits) for each register in this work, which would achieve a better trade-off between energy and reliability for most programs.

B. The Effectiveness of LAD-ECC

Fig. 11 shows the normalized energy consumption obtained by AP-ECC, DA-ECC, and LAD-ECC when running our chosen 16 benchmarks. The results are normalized to the baseline, which is register file with full SEC-DEC ECC protection.

We observe that AP-ECC only consumes 48.65% energy comparing with full-ECC. This indicates that since only the upper 17-bits are protected, AP-ECC consumes less energy to generate, store and check parity bits compared to the baseline. Besides, we observed that the normalized energy efficiency of AP-ECC is similar among various investigated benchmarks. This is because all benchmarks protect the same number of bits in register file, which reduces 52.9% generation and 50.5% verification energy respectively for each register value. Since these two phases constitute the majority of energy consumption of ECC in GPGPUs register file and the energy reduction for these two steps are similar, the normalized energy reductions than baseline are also similar across applications.

By avoiding duplicate ECC generation and checking, DA-ECC reduces redundant energy consumption of ECC by 40.65% on average. Especially, applications with substantial duplicate operands gain abundant energy savings. For instance, `streamcluster` and `gaussian` only consume 43.49% and 44.74% energy respectively compare with full ECC.

As a combination of DA-ECC and AP-ECC, LAD-ECC achieves significant energy reduction for all benchmarks. For example, due to their low ratios of redundant register variables, `correlation` and `covariance` gain little energy savings (i.e., 12.45%, 11.18%) under DA-ECC. However, by combining with AP-ECC, LAD-ECC increases the energy saving to 57.34% and 56.99% for these two benchmarks respectively. On average across the investigated benchmarks, LAD-ECC only consumes 30.28% energy of full-ECC. The baseline full SEC-DEC ECC additionally consumes about 38% energy of register files, hence, our LAD-ECC achieves about 26.49% energy savings for the entire GPGPUs register file.

Adding up the parity in the ECC table, we estimate the area overhead of our technology. AP-ECC use 6-bit ECC for the significant bits of each register. Compared to full-ECC that needs 7-bit ECC for each register, our technique reduces 14.22% area of the baseline, which reduce 4KB of the ECC table size per SM. The area overhead of ECC generators, ECC checkers, and power gating logics are not considered due to their negligible size [8].

Fig. 12: Comparison among Shield, AP-ECC, DA-ECC and LAD-ECC.

C. Comparing LAD-ECC with Shield

We further compare our techniques with an efficient ECC technology, Shield [8], which utilizes a relatively small ECC table to protect long-lived register variables. Fig. 12 shows the comparison of Shield, AP-ECC, DA-ECC and LAD-ECC. The results are normalized to full-ECC. These techniques do not affect performance because the ECC generation and checking processes are in parallel with register file access.

Shield reduces ECC's energy by around 30% owing to the small ECC table. Excitingly, our LAD-ECC tremendously reduces 69.72% of ECC energy with the combination of AP-ECC and DA-ECC, which is more energy-efficient than Shield.

As its only drawback, LAD-ECC shows fewer error coverage than Shield. This is because AP-ECC merely protects very few bits within a register. However, since we consider the error-sensitivity difference among data bits during the protection, our LAD-ECC would not incur large magnitude errors.

VI. Related Work

Recently, several prior works have been proposed to explore the impact of soft errors in GPGPUs [6], [9], [13], [15]. For example, *Palframan et al* [13] propose a precision-aware protection technology for execution logic in GPGPUs. Leveraging the soft-error robustness of spin-transfer torque RAM (STT-RAM), *Tan et al* [5] proposed the hybrid STT-RAM and SRAM based resister file to combat soft errors while significantly reducing the energy consumption with negligible performance loss. *Oliveira and Rech et al* [7] discussed the efficiency and overhead of ECC on modern GPGPUs.

All these works are orthogonal to our LAD-ECC mechanism. LAD-ECC delivers a suite of progressively capable techniques, which is able to reduce the area and energy overhead of ECC aggressively by exploring the approximation and duplication characteristics in GPGPUs register files.

VII. Conclusion

In this work, we propose LAD-ECC, which leverages the approximate features of program and duplicate characteristics in warp-registers to design an energy-efficient ECC mechanism for GPGPUs register file. LAD-ECC is composed of two techniques: (1) by only providing a shield for the the upper 17-bits of the 32-bit register, AP-ECC is proved able to combat critical errors to ensure the reliability of programs; (2) leveraging duplication characteristics among same-named registers, DA-ECC improve the energy efficiency of ECC by avoiding unnecessary ECC generation and verification. Our experimental results show that, on average, LAD-ECC achieves 69.72% energy saving of the traditional SEC-DED ECC.

References

[1] A. Azizimazreah, Y. Gu, X. Gu, and L. Chen. Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–10, 2018.

[2] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. Reddi. Mavbench: Micro aerial vehicle benchmarking. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 894–907, 2018.

[3] Ang Li, Shuaiwen Leon Song, Mark Wijtvliet, Akash Kumar, and Henk Corporaal. Sfu-driven transparent approximation acceleration on gpus. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 15:1–15:14. ACM, 2016.

[4] Daniel Maier, Biagio Cosenza, and Ben Juurlink. Local memory-aware kernel perforation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, pages 278–287. ACM, 2018.

[5] Jingweijia Tan, Zhi Li, and Xin Fu. Soft-error reliability and power co-optimization for gpgpus register file using resistive memory. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, DATE '15, pages 369–374. EDA Consortium, 2015.

[6] F. G. Previlon, C. Kalra, D. R. Kaeli, and P. Rech. Evaluating the impact of execution parameters on program vulnerability in gpu applications. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 809–814, March 2018.

[7] D. A. G. Oliveira, P. Rech, L. L. Pilla, P. O. A. Navaux, and L. 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.

[8] P. Montesinos, W. Liu, and J. Torrellas. Using register lifetime predictions to protect register files against soft errors. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 286–296, 2007.

[9] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. 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, April 2017.

[10] C Nvidia. Nvidias next generation cuda compute architecture: Kepler gk110. *Whitepaper (2012)*, 2012.

[11] Z. Chen, D. Kaeli, and N. Rubin. Characterizing scalar opportunities in gpgpu applications. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 225–234, 2013.

[12] Sangpil Lee, Keunsoo Kim, Gunjae Koo, Hyeran Jeon, Won Woo Ro, and Murali Annavaram. Warped-compression: Enabling power efficient gpus through register compression. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 502–514. ACM, 2015.

[13] D. J. Palframan, N. S. Kim, and M. H. Lipasti. Precision-aware soft error protection for gpus. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 49–59, Feb 2014.

[14] Louis-Noël Pouchet. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[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)*, pages 749–761, 2018.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009.

[17] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009.

[18] I Meta-Software. Hspice users manual, 1996.

[19] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.1. Technical report, Technical Report HPL-2008-20, HP Labs, 2008.