REFU: Redundant Execution with Idle Functional Units, Fault Tolerant GPGPU architecture

# REFU: Redundant Execution with Idle Functional Units, Fault Tolerant GPGPU architecture

Raghunandana K K
*U R Rao Satellite Center*
Bangalore, India
kkraghu@ursc.gov.in

Varaprasad BKSVL
*U R Rao Satellite Center*
Bangalore, India
bksvlvp@ursc.gov.in

M. Sonza Reorda
*Politecnico di Torino*
Torino, Italy
matteo.sonzareorda@polito.it

Virendra Singh
*Indian Institute of Technology*
Bombay Mumbai, India
viren@ee.iitb.ac.in

*Abstract*— **The General-Purpose Graphics Processing Units (GPGPU) with energy efficient execution are increasingly used in wide range of applications due to high performance. These GPGPUs are fabricated with the cutting-edge technologies. Shrinking transistor feature size and aggressive voltage scaling has increased the susceptibility of devices to intrinsic and extrinsic noise leading to major reliability issues in the form of the transient faults. Therefore, it is essential to ensure the reliable operation of the GPGPUs in the presence of the transient faults. GPGPUs are designed for high throughput and execute the multiple threads in parallel, that brings a new challenge for the fault detection with minimum overheads across all threads. This paper proposes a new fault detection method called REFU, an architectural solution to detect the transient faults by temporal redundant re-execution of instructions using the idle functional execution units of the GPGPU. The performance of the REFU is evaluated with standard benchmarks, for fault free run across different workloads REFU shows mean performance overhead of 2%, average power overhead of 6%, and peak power overhead of 10%.**

***Keywords— GPGPU, Fault tolerant, Redundant, Functional units***

## I. Introduction

The increased computing power and programmability of GPGPUs expanded their utilization in wide range of parallel applications. The sensitive nature of these applications for data errors, demand 100% correct execution. On the other hand, to meet the demand for faster and high throughput devices, GPGPUs are fabricated with energy efficient, scaled down, faster transistors with reduced voltage levels. The shrinking dimensions and aggressive voltage scaling led to increased susceptibility to intrinsic and extrinsic noise [1] which will be manifested as transient fault. With the technology scaling and high packing density of transistors, soft error rate (SER) of IC's is increasing [2]. The transient faults modify the logic state of processor's memory elements known as single event upset and cause a timing or functional failure.

In current GPGPUs the memory structures such as register files, L1/L2 caches, shared memory and DRAM are protected using the parity or ECC [3] [4] [5]. Other structures like arithmetic logic units (ALUs), instruction schedulers and interconnect network are prone to transient faults. In execution pipeline of GPGPUs, error free operation of the ALUs is key in achieving the programmed order instruction flow and correctness of the program executed.

In this paper, we propose a microarchitecture hardware solution for detecting the transient faults in the execution pipelines of the GPGPUs. We have analysed the vulnerability of microarchitecture structures for transient faults, and fault coverage of overall system is assessed.

In the proposed method, ALU is logically divided into independent sub functional units and temporal re-execution redundancy is used for the error detection. Every ALU executed instructions along with the operands, results and associated flags are stored in a re-execution buffer called replay buffer. The stored instructions are re-executed whenever the functional units are free or replay buffer is full. Errors are detected by comparing the stored result of primary execution with re-execution result.

The benefits of the proposed method are demonstrated through set of ISPASS 2009 and RODINIA benchmarks. Performance and power overhead are compared with the original microarchitecture configuration. Experimental results show almost full fault coverage across all threads is achieved with mean performance overhead of 2%, average power overhead of 6%, and peak power overhead of 10%.

The major contributions of the paper are

1) A fault tolerant (FT) architecture transparent to user is proposed. Replay buffer stores the dependency resolved operand values, result and flags for re-execution and verification. Different warp instructions can reside in replay buffer.
2) The functional units (FU) of the ALU simultaneously execute different primary and redundant instructions minimizing the performance degradation due to temporal sharing of FUs.
3) Near real time error detection capability across different warps and threads.
4) Fault coverage capability comparable with the DMR system.

The paper is organized as follows. The related work on FT processor architectures are discussed in Section II. Section III presents the proposed REFU architecture for fault detection. The effectiveness of the REFU for fault coverage and its analysis is discussed in Section IV. The simulation results estimating performance and power impact are presented in Section V, followed by conclusion of our work in section V

## II. Related Work

A GPGPU consists of a scalable number of in-order Streaming Multiprocessors (SM), each SM use number of Streaming Processors (SP) which are in order machines. Some of the CPUs FT techniques are adopted for GPGPUs with modifications. Sohi et al. [6] proposed a simple instruction duplication and time redundant execution of instruction on execution units. REESE [7] proposed by Nickel et al. utilizes the idle capacity that is inherent in general purpose processors to time multiplex main and redundant execution for transient errors detection. Viney et al. [8] proposed the utilization of floating resources in a fixed pipeline for fault detection, they utilized the idle functional units of in order processor as floating resource to launch simultaneous main & redundant execution. In order to overcome the masking effect of permanent faults in detecting transient faults Patel et al. proposed RESO [9], where the duplicated instruction operates on the shifted operand. For Superscalar processors Shoba et al. [10] proposed an architecture REMO where the concept of re-execute instruction during retire is used for fault detection.

For the GPGPUs Ralph et al. [11] proposed Argus-G to detect errors in a thread's data flow by comparing the static data flow graph (DFG) to the dynamic DFG computed by the SP during execution. Josie et al. [12] evaluated the sensitivity of the pipeline registers in the GPUs to SEU when applications are hardened with the low-level software mechanisms. Fang et al. [13] have analyzed the reliability properties of the application running on GPUs by fault injection. Backer et al. [14] proposed tunable fault detection scheme to balance GPU performance and fault checking, threads are replicated and assigned across different SPs for lockstep execution. Warped-DMR proposed by Jeon et al. [15] uses the inactive threads to verify execution of active threads in the same warp and uses a special purpose Replay Queue to duplicate execution warps and executed on free SM when functional unit within SMs are not free. Condia et al. [16] identified the modules that are more likely to impact the execution and applied a selective hardening strategy at device level by selective hardening the GPU modules to mitigate the soft errors. RISE proposed by Tan et al. [17] predicts the warp stall time and estimates the load imbalances among cores, for soft-error detection re-uses the fully idled SPs in the GPU core caused by the long-latency memory accesses and the load imbalance among cores. All these proposals have high performance and power overhead.

## III. PROPOSED ARCHITECTURE

### A. FT architecture

In GPGPUs each SM consists of multiple SPs using Single Instruction Multiple Thread (SIMT) execution model. In current GPGPUs, the memory structures are protected using the parity or ECC, thus part of the GPGPU structures are protected by information redundancy. To achieve the high fault coverage it is sufficient to develop detection mechanisms for the unprotected system parts. In the GPGPU execution pipeline ALUs are vulnerable for the transient faults.

Instructions which do not use the ALU functional units such as *mov, nop, direct load/store* operation results are protected via parity hence re-execution is not required. Re-execution of ALU instructions will result in high fault coverage of execution pipeline. A typical ALU [18] of SP is as shown in Figure 1.

At any given time all hardware resources of ALU are not fully utilized by ALU instructions. Logically ALU can be divided into smaller independent sub functional units such as add/subtract unit, logical operation unit, multiplier unit, data conversion etc. The different type of instructions issued by the warp scheduler provide an opportunity window for re-execution of instructions when functional units are free.

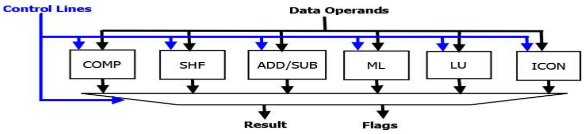


Fig. 1. Functional Units of ALU.

### B. Implementation

To implement the fault tolerant architecture, the following structures are newly added : (i) Replay buffer (ii) Control logic to stall the SP pipeline when replay buffer is full or ALU functional unit is busy, and (iii) Result comparison logic. Figure 2 shows the modified SM with Replay Buffer.

Replay Buffer Data Structure

| Warp ID | Instruction Type | Source Operands | Result | Flags | Valid Bit | Reexecute bit |
|---|---|---|---|---|---|---|

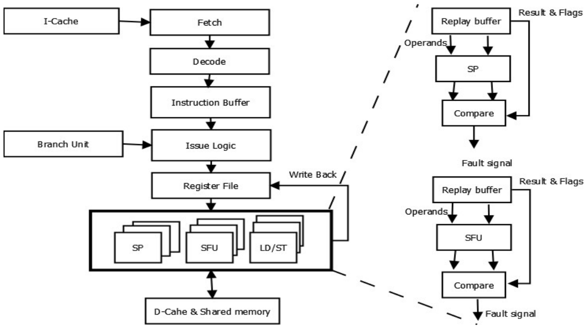
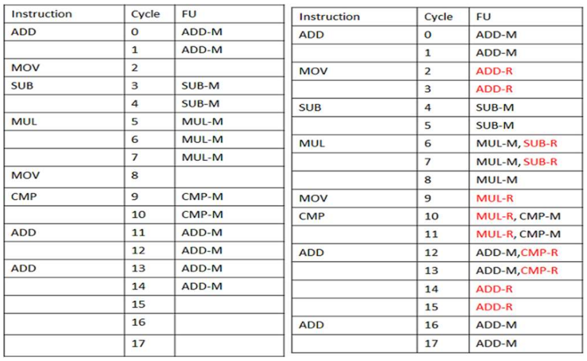


Fig. 2. GPGPU Instruction flow for FT architecture.

Replay buffer's valid bit is to indicate authentic instruction ready for the re-execution and re-execute bit to indicate the progress of re-execution so that no new instruction can be written into current replay buffer location.

### C. Working principle

Working principle of the proposed method is shown in Figure 3 for the replay buffer size of 1. Instructions tagged with the 'M' are the primary executions and 'R' are the redundant instructions executions from the replay buffer. Similar to example illustrated in Figure 3, multiple instructions can execute in an SP every cycle using different functional units.



| Instruction | Cycle | FU |
|---|---|---|
| ADD | 0 | ADD-M |
|  | 1 | ADD-M |
| MOV | 2 |  |
| SUB | 3 | SUB-M |
|  | 4 | SUB-M |
| MUL | 5 | MUL-M |
|  | 6 | MUL-M |
|  | 7 | MUL-M |
| MOV | 8 |  |
| CMP | 9 | CMP-M |
|  | 10 | CMP-M |
| ADD | 11 | ADD-M |
|  | 12 | ADD-M |
| ADD | 13 | ADD-M |
|  | 14 | ADD-M |
|  | 15 |  |
|  | 16 |  |
|  | 17 |  |

| Instruction | Cycle | FU |
|---|---|---|
| ADD | 0 | ADD-M |
|  | 1 | ADD-M |
| MOV | 2 | ADD-R |
|  | 3 | ADD-R |
| SUB | 4 | SUB-M |
|  | 5 | SUB-M |
| MUL | 6 | MUL-M, SUB-R |
|  | 7 | MUL-M, SUB-R |
|  | 8 | MUL-M |
| MOV | 9 | MUL-R |
| CMP | 10 | MUL-R, CMP-M |
|  | 11 | MUL-R, CMP-M |
| ADD | 12 | ADD-M,CMP-R |
|  | 13 | ADD-M,CMP-R |
|  | 14 | ADD-R |
|  | 15 | ADD-R |
| ADD | 16 | ADD-M |
|  | 17 | ADD-M |

Fig. 3. Comparison of Instruction execution in Normal and FT mode.

When instruction is issued by the warp scheduler, the ALU independent instructions are executed and not marked for re-execution. The ALU dependent instructions once execution is over, instructions along with operands and result are stored in the replay buffer and marked for re-execution.

Concurrent execution of multiple instructions in ALU using different sub functional units hide performance loss due to redundant re-execution. Errors are detected by comparing stored primary execution results with that of redundant re-execution.

## IV. FAULT ANALYSIS & COVERAGE

In REFU replay buffer and temporal re-execution provide physical and logical sphere of replication (SOR) for part of processor hardware, apart from this vulnerability of execution pipeline stages are analyzed for SOR to asses fault coverage.

### *A. Coverage Analysis*

Case i) Transient faults in the FU of ALU: A single bit flips in functional unit will alter the result. The same altered result will be stored in the replay buffer along with the operands. Hence during the re-execution, the error will be detected.

Case ii) Any single bit flip in the Replay buffer: A single bit flips in any of (i) operands, (ii) result, and (iii) flags will be detected during the re-execution. A bit flip in the instruction type will lead to wrong operation and result mismatch will be detected. The vulnerable fields are Warp ID, valid-bit and the re-execute bit they are protected using parity.

### *B. Vulnerability of execution pipeline*

1) Fetch unit: The I-Cache and associated bus are protected using parity or ECC so correctness data and instruction is ensured, it falls within SOR.
2) Decoder unit: Decoding error will result in the wrong operation and it cannot be detected so it is outside SOR.
3) Register file: Register file is protected using parity hence input to execution unit is within SOR.
4) Execution unit: The faults in execution unit are detected through redundant re-execution hence it is within SOR
5) Write Back unit: Results are written back to register file after execution is completed and correctness of result is verified later. On detection of a fault entire warp is re-executed from the previous check point hence it is within SOR.
6) Memory unit: It uses the address generated by the execution pipeline stages. On fault detection the recovery mechanism is similar to write back unit. L1/L2 cache and DRAM are protected through parity hence it is within SOR.
7) Instruction scheduler(IS): IS uses Program counter (PC) of warp, re-convergent buffer and scoreboard to schedule an instruction, memory structures used by IS are protected through parity and at every instruction issue the current PC is compared with the predicted PC stored in memory hence it is also falls within SOR.

From SOR analysis of execution pipeline of SM we conclude that decoder unit is vulnerable to faults it need to be hardened with fault detection techniques, other pipe line structures are within SOR and provide fault detection capability hence recovery can be initiated.

## V. EVALUATION

The cycle accurate GPGPU architecture simulator GPGPU-sim [19] is modified to incorporate the replay buffer, pipeline stall mechanism and re-execution logic. GPUWattch [20] is used to estimate the power overhead of redundant execution. Set of ISPASS 2009 and RODINIA benchmarks are used for evaluation with different replay buffer size 1 to 4. For each benchmark power and performance estimate of the unmodified GPGPU-sim over total execution time normalized to 100 is taken for baseline comparison. The Table I shows are over all configuration of the GPGPU-sim.

TABLE I. GPGPU-SIM CONFIGURATION

| GPGPU Architecture: GTX480 | **GPU Pipeline Width** |
|---|---|
| Number of Clusters: 15 | Number of SP units: 2 |
| Cores per Cluster: 1 | Number of SFU units: 1 |
| Maximum Warps per SM: 48 | Number of LD/ST units: 1 |
| Warp scheduler: LRR | L1/Shared memory: 48KB |

### *A. Performance*

The Figure 3 shows the performance of the REFU. Majority of the benchmarks show performance overhead less than 3%, over the entire range of benchmarks mean performance overhead is 2%.

### *B. Power*

The Figure 4 and Figure 5 show average and peak power overhead of the REFU, with minimum performance overhead majority of the workloads show less than 4% increase in average power and less than 8% increase in peak power. Over entire range of benchmarks average power overhead is 6% and peak power overhead is 10%.

The performance improvement is directly proportional to peak and average power overheads. Table II shows mean performance and power overheads taken over all benchmarks.

TABLE II. MEAN PERFORMANCE AND POWER

| Replay Buffer size | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| IPC | 93.83 | 98.52 | 98.63 | 98.42 |
| Average Power | 100.91 | 106.21 | 106.2 | 105.68 |
| peak Power | 105.03 | 109.28 | 110.3 | 109.6 |

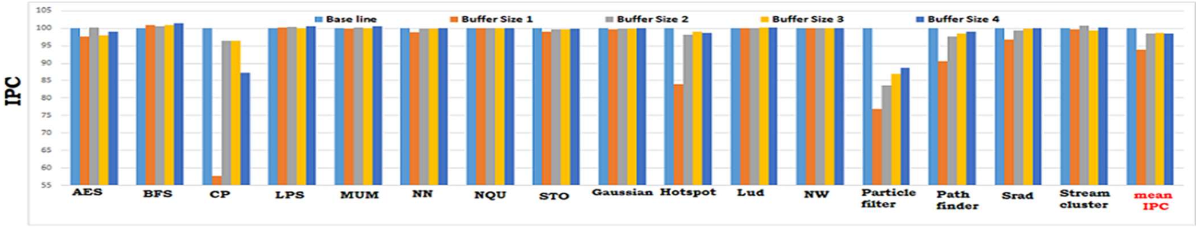


Fig. 4. GPGPU Instruction flow for FT architecture.

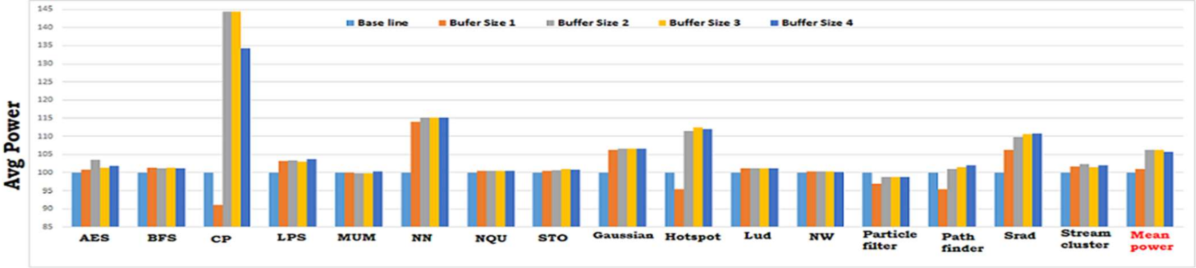


Fig. 5. GPGPU Instruction flow for FT architecture.

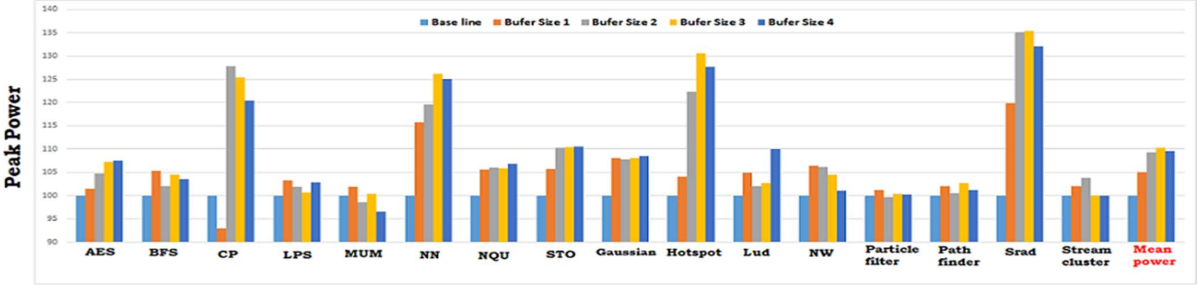


Fig. 6. GPGPU Instruction flow for FT architecture.

## VI. CONCLUSION

GPGPUs are used in wide range of applications such as scientific computing, Machine Learning Classification, Automatic Driving Assistance etc, the sensitive nature of scientific applications to data errors demand 100% correct execution. we presented a microarchitecture solution called REFU, it effectively utilizes the idle functional units of ALU for temporal redundant re-execution to detect transient faults. REFU achieves high fault coverage with minimal performance and power overheads. For standard benchmarks we have shown that REFU has mean performance overhead of 2%, average power overhead of 6%, and peak power overhead of 10%.

## REFERENCES

[1] J. W. McPherson, “Reliability challenges for 45nm and beyond”, in DAC, July 2006, pp. 176–181.

[2] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, “Modeling the effect of technology trends on the soft error rate of combinational logic,” in Proc. Intl. Conf. on Dependable Systems and Networks, pages 389-398, 2002.

[3] NVIDIA Fermi Architecture Whitepaper. [Online]. Available: http://www.nvidia.com/content/pdf/fermi white papers/nvidia fermi compute architecture whitepaper.pdf

[4] NVIDIA Kepler GK110 Architecture Whitepaper. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110- Architecture-Whitepaper.pdf

[5] GP100 Pascal Whitepaper. [Online]. Available: https://images.nvidia. com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[6] Sohi,G.,Franklin, M. and Saluja,K, ”A Study of Time Redundant Fault Tolerant Techniques,” in High Performance Pipelined Computers. Proc. FTCS-19.pp. 436-443,1989.

[7] Nickel, J. B., and Somani, A. K., 2001 July, “Reese: a method of soft error detection in microprocessors”, in 2001 International Conference on Dependable Systems and Networks, pp. 401–410.

[8] Viney Kumar, Rahul Raj Choudhary, Virendra Singh,“FREP: A Soft Error Resilient Pipelined RISC Architecture”, 2010 East west Design and Test symposium 1010, pp 330–333.

[9] Patel, J. H., and Fung, L. Y., 1982 July, “Concurrent error detection in alu’s by recomputing with shifted operands”, IEEE Transactions on Computers C-31, 589–595.

[10] Shoba Gopalakrishnan and Virendra Singh., July 2016, “REMO: Redundant Execution with Minimum Area,Power, Performance Overhead Fault Tolerant Architecture”, 22nd IEEE IOLTS.

[11] Ralph Nathan,Daniel J. Sorin, “Argus-G: Comprehensive, Low-Cost Error Detection for GPGPU Cores”, IEEE Computer Architecture Letters 2015.

[12] Josie E. Rodriguez Condia,Marcio M. Goncalves,Jose Rodrigo Azambuja,Matteo Sonza Reorda,Luca Sterpone, “Analysing the Sensitivity of GPU Pipeline Registers to Single Events Upsets”, 2020 IEEE Computer Society Annual Symposium on VLSI.

[13] Karthik Pattabiraman; Matei Ripeanu; Sudhanva Gurumurthi, “A Systematic Methodology for Evaluating the Error Resilience of GPGPU Applications Bo Fang”, IEEE Transactions on Parallel and Distributed Systems ( Volume: 27, Issue: 12, Dec. 1 2016).

[14] Jerry B. Backer; Ramesh Karri, “Balancing Performance and Fault Detection for GPGPU Workloads”, 2012 IEEE 30th International Conference on Computer Design (ICCD).

[15] Hyeran Jeon Murali Annavaram, “Warped-DMR: Light-weight Error Detection for GPGPU”, 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture.

[16] Josie E. Rodriguez Condia; Paolo Rech; Fernando Fernandes dos Santos; Luigi Carrot; M S Reorda, “Protecting GPU’s Microarchitectural Vulnerabilities via Effective Selective Hardening”, 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design.

[17] ingweijia Tan, Xin Fu RISE, “ Improving the Streaming Processors Reliability Against Soft Errors in GPGPUs”,International Conference on Parallel Architecture and Compilation Techniques (PACT)PACT’ 12.

[18] Josie E. Rodriguez Condia, Pierpaolo Narducci, M. Sonza Reorda, L. Sterpone , “ A dynamic reconfiguration mechanism to increase the reliability of GPGPUs”, 2020 IEEE 38th VLSI Test Symposium.

[19] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, Timothy G Rogers, “ Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling”, in proceedings of the 47th IEEE/ACM International Symposium on Computer Architecture (ISCA), 2020.

[20] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim,Tor M. Aamodt, Vijay Janapa Reddi, “ GPUWattch: Enabling Energy Optimizations in GPGPUs”, in proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA 2013), Tel-Aviv, Israel, June 23-27, 2013.