Long-Term Reliability Management For Multitasking GPGPUs

Zeyu Sun*, Taeyoung Kim†, Marcus Chow†, Shaoyi Peng*, Han Zhou*,
Hyoseung Kim*, Daniel Wong* Sheldon X.-D. Tan*
*Department of Electrical and Computer Engineering, University of California, Riverside
†Department of Computer Science and Engineering, University of California, Riverside

***Abstract*—This paper proposes long-term reliability management for spatial multitasking GPU architectures. Specifically, we focus on electromigration (EM)-induced long-term failure of the GPU's power delivery network. A distributed power delivery network model at functional unit granularity is developed and used for our EM analysis of GPU architectures. We use a recently proposed physics-based EM reliability model and consider the EM-induced time-to-failure at the GPU system level as a reliability resource. For GPU scheduling, we mainly focus on spatial multitasking, which allows GPU computing resources to be partitioned among multiple applications. We find that the existing reliability-agnostic thread block scheduler for spatial multitasking is effective in achieving high GPU utilization, but poor reliability. We develop and implement a long-term reliability-aware thread block scheduler in GPGPU-Sim, and compare it against existing reliability-agnostic scheduler. We evaluate several use cases of spatial multitasking and find that our proposed scheduler achieves up to 30% improvement in long-term reliability.**

I. Introduction

The use of general-purpose graphics computing units (GPGPUs) for high-performance computing has recently gained much attention. However, as the complexity of high-performance computing systems continues to increase, the probability of failure in one of the machines is also expected to increase. This naturally brings up the necessity of research in reliability, especially with the focus on GPGPUs. Long-term reliability such as electromigration (EM) is one of the concerns in modern VLSI design. As technology advances, reliability is perceived as a major constraint in high-performance computing due to the high failure rates in deep submicron devices.

While long-term reliability of computing systems has emerged as a serious problem, to the best of our knowledge, no prior work has assessed its impact on GPGPUs. Only some initial efforts on soft errors have been carried out for GPGPUs [1] where most of the radiation-induced (soft error) failures are caused by the corruption of memory resources. Our main contribution in this work is the quantification of the EM-induced interconnect reliability of GPGPUs using spatial multitasking, with widely-used GPGPU benchmarks. Spatial multitasking is a technique commonly used in GPUs to improve GPU utilization [2]. Under spatial multitasking, thread blocks from different applications are allocated to their own exclusive sets of Streaming Multiprocessors (SMs), enabling simultaneous execution of multiple kernels.

In this paper, we propose long-term reliability management for GPGPUs using spatial multitasking for executing general-purpose workloads. We develop a distributed power delivery network model at functional unit granularity. We utilize this PDN model for our EM analysis of GPU architectures using a recently proposed physics-based EM reliability model and consider the EM-induced time-to-failure (TTF) at the GPU system level as a reliability resource. For GPU scheduling, we focus on spatial multitasking, which allows GPU computing resources, i.e., SMs, to be partitioned among multiple applications. We find that the existing reliability-agnostic thread-block scheduler for spatial multitasking is effective in achieving high GPU utilization, but ineffective for reliability. We develop a long-term reliability-aware thread-block scheduler in GPGPU systems and evaluate with widely-used GPGPU benchmarks.

This work is supported in part by NSF grant under No. CCF-1527324, and in part by NSF grant under No. CCF-1816361.

II. Long-Term Reliability Model for GPGPU

A. Physics-based electromigration assessment model

EM is a major challenge and limitation for VLSI design with increasing temperature, current and voltage in interconnect trees. Atoms (either lattice atoms or defects/impurities) migrate toward the anode end of metal wire along the trajectory of conducting electrons and eventually form a void leading to increase of resistance. In order to model this phenomenon, a recently proposed three-phase EM model [3], [4] has been employed in this work.

In this model, the EM wear-out process consists of three phases: (a) nucleation phase; (b) incubation phase; (c) growth phase. In the nucleation phase, stress starts to build-up over time. When it exceeds critical stress, void will be formed and we enter the incubation phase. When the void is larger than the critical size for the wire, the wire resistance starts to change and we enter the growth phase. The new 3-phase EM model gives a more accurate time to failure estimation and can be applied to more general multi-segment wires since it is based on the stress diffusion physics in confined copper wires.

1) EM assessment on on-chip power delivery network: Because of the concern with the long-term average effects of the current, EM related work generally assumes a DC model of the power delivery network (PDN). Resistance change for branches in growth phase will affect the current density in other branches.

In order to get an accurate result, in the new EM-induced reliability analysis algorithm for PDN, we check the voltage drops of the nodes at fixed time steps. The resistance of one or more wires begins to change (increase) starting with the void nucleation times. At each time step, we collect new wires whose nucleation times were reached, and compute the new resistance for existing wires in the growth phase and the corresponding voltage drops of the whole grid. This process is repeated until the voltage drop of one or more nodes exceeds the critical voltage drops allowed.

B. System-level reliability resource consumption model

Given the new physics-based EM model, we now introduce our system level EM-reliability resource consumption model with TTF. We notice that treating the EM as a resource was first introduced in [5]. But this work is still based on the traditional Black's equation.

Assume that we have a set of different time intervals Δp_k characterized by different workloads in terms of current density j_k and temperature T_k for a processor. It means that $P = \sum_{k=1}^{n} \Delta p_k$ is the total execution time. Each kth workload, if it lasts until imaginary failure, provides time-to-failure TTF_k. Thus the failure rate at the kth workload, which last Δp_k is $\lambda_k = 1/TTF_k$. Then the average failure rate for the considered set of work loads can be expressed as follows.

$$\lambda_{avg} = \sum_{k=1}^{n} \frac{\Delta p_k}{\sum_{j=1}^{n} \Delta p_j} \lambda_k = \sum_{k=1}^{n} \frac{\Delta p_k}{P} \lambda_k \tag{1}$$

As a result, the expected time to failure or average lifetime of the whole processor, TTF is [5],

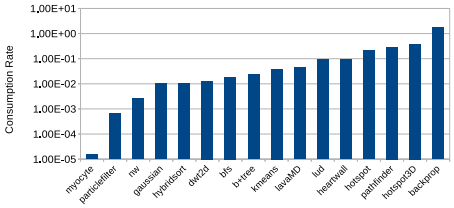

Fig. 1. Consumption Rate of Rodina Benchmarks

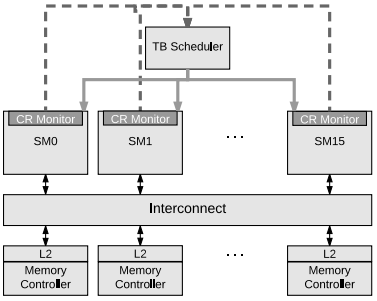

Fig. 2. GPU Architecture

$$TTF = \frac{1}{\lambda_{avg}} = \frac{1}{(\sum_{k=1}^{n}(\Delta p_k \frac{1}{TTF_k}))/P} \tag{2}$$

Based on the (2), we can treat the lifetime of the processor specified by TTF as a resource that could be consumed as the SM works. We first define the specified TTF as a nominal value, denoted as TTF_N, which is the intended or required life of the SM under a typical temperature and power setting for a core or system.

However, in reality, TTF varies under different temperature and power settings and we define *consumption rate* for workload k as

$$cr_k = \frac{TTF_N}{TTF_k} \tag{3}$$

in which the lifetime in real case (TTF_k under the kth workload) could be estimated by the new proposed reliability model in the previous sub-sections.

If $TTF_k > TTF_N$, then $cr_k < cr_N$, which indicates that heavy tasks are assigned and the SM is consuming its nominal lifetime at a lower rate, and thus the real lifetime is longer than the nominal one. Conversely, if $TTF_k < TTF_N$, then $cr_k > cr_N$, which indicates that light tasks are assigned and the SM is consuming its nominal lifetime at the higher rate, and thus the real lifetime is shorter than the nominal one.

In Fig. 1 we show the consumption rate of a range of common GPU applications. Clearly, there is a wide range of consumption rate behavior across benchmarks.

III. GPGPU Architecture and Stream Multiprocessor Scheduling

Fig. 2 provides an overview of the GPU architecture in this work. This work uses NVIDIA GTX480 Fermi GPGPU with 15 Streaming multiprocessors (SMs) as the baseline architecture. Each SM is comprised of a 128KB register file, two warp schedulers, two SP execution units, one SFU, and 16 Load/Store units. The SM core clock is 700MHz with each SP unit containing 16 double frequency CUDA cores, each with individual integer and floating point pipelines. Each SM has its own 64KB shared memory and L1 cache. Fermi supports up to 48 active warps per SM. Each warp comprises of 32 threads executing in a lockstep manner, also called Single Instruction Multiple Thread (SIMT) execution model. There

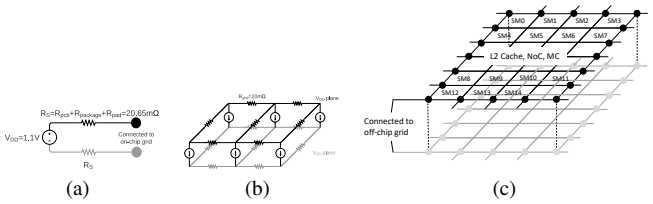

Fig. 3. Off-chip PDN (a), details of PDN of each SM (b), and On-chip PDN (c)

are a total of 1,536 active threads per SM. The SMs are all connected to an interconnection network which leads to 6 memory partitions, containing a shared L2 cache and memory controller.

GPU applications are processed on the GPU as CUDA contexts that consist of kernels. Kernels are further broken up into thread blocks, which are scheduled to individual SMs. These thread blocks are further broken up into warps, which are executed upon. GPUs can run multiple independent kernels from the same application or from multiple applications through multi-process service. This enables spatial multitasking where applications can run concurrently on different SMs to improve the utilization and efficiency of the GPU.

The goal of this work is to optimize the GPU for long-term reliability. To achieve this, we add a lightweight on-chip EM-aging sensor [6] to each SM. This EM-based aging sensor exploits the natural aging/failure mechanism of interconnect wires to time the aging of the chip. Compared with existing aging sensors, this sensor provides a more accurate prediction of the chip usage time at smaller area footprints due to its simple structure. By utilizing the EM-aging sensor, we estimate the reliability consumption rate (cr_k) of each SM in order to make fine-grain scheduling decisions.

IV. Simulation Framework for EM Assessment on GPGPU

For the EM assessment of GPGPUs, PDN should be designed and simulated as discussed in II-A1. The PDN consists of two parts, off-chip network and on-chip network. The off-chip part is shown in Fig 3(a). It takes account of the resistors between on-chip PDN and the power source, which are the resistors of PCB, package, and pad. The on-chip part is built based on the dimensions of the publicly available specification and die photo for GTX480. The description explains that the L2 cache, Network on Chip (NoC), and memory controllers (MC), which are located in the middle, take approximately the area of 8 SMs. Thus, we split the PDN for the whole chip to 24 wide sections (4 horizontally and 6 vertically), which are shown in Fig 3(c). The 20 grid points on the edges, which are emphasized by dots, are connected to off-chip PDN respectively. Additionally, it is also shown in the die photo that the aspect ratio of each SM is about 2. Based on the fact that resistors of interconnects are proportional to their length, we model the PDN for each SM by a simple network consisting of 12 resistors shown in Fig 3(b). Besides the 12 resistors for VDD and GND networks, there are 6 current sources connecting two corresponding grid points in the two networks, which model the current that flows through the SM. We assume that the overall current of each SM is evenly distributed to these six current sources. For the current flowing through L2 cache, NoC and MC, it is modeled by 15 evenly distributed current sources as this part is modeled by 15 grid points. For the grid points at the border of two, or four SMs (or L2 cache, NoC, MC), the value of the current source is calculated by summing the two, or four SMs' currents, and then as stated above, dividing by 6. It is assumed in GPGPU-Sim that the operation voltage for the chip is $1V$, so we can easily get the current that flows through each part from the simulated power trace by GPGPU-Sim.

Fig. 4 shows an overview of our long-term reliability simulation framework for GPUs. Benchmarks are run on GPGPU-Sim, in which activity traces are fed into GPUWattch. GPUWattch extracts a power trace with a sampling rate of 500 cycles. This power trace is then fed

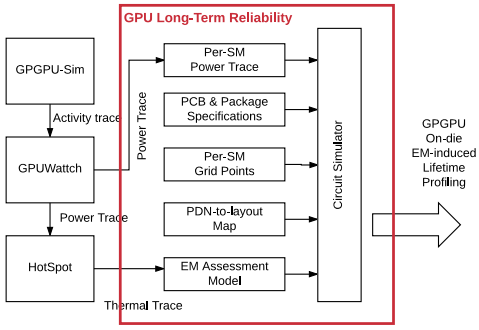

Fig. 4. Simulation framework for long-term reliability assessment on GPGPU

into HotSpot in order to derive a thermal trace of the GPU. Both the power trace and the thermal trace are fed into the long-term reliability framework for EM simulation on PDN.

V. Resource Consumption Rate-Aware Thread Block Scheduling

As shown in Fig. 1, consumption rate (cr_k) of benchmarks varies greatly. In this work, we focus on the lifetime of the entire GPU which is SM with shortest lifetime. Therefore, our goal is to balance the wear-out of all SMs in the GPU. We assume that the number of SMs allocated for each application is given by the user. Hence, we will focus on the scheduling of partitioned applications for long-term reliability.

There are two key observations for the GPU long-term reliability: (i) the computational performance of applications depends only on the number of SMs allocated. (ii) wear-out balancing can be easily achieved by rotating applications across SMs. So we can find that shifting each application's designated SM subset by one is a simple but effective balancing method. When the thread block scheduler swaps the target SMs for each application's kernel, we ensure that the already-executing thread blocks in an SM will continue until completion and will not be preempted or migrated in the middle of thread execution. This is due to the fact that many of GPU architectures as yet do not support preemption at an arbitrary point and doing so causes high overhead for GPU context switching.

The main challenge in designing our consumption rate-aware thread block scheduler is in determining the triggering condition. To this end, we will explore the following thread block mapping schemes.

Baseline: In the baseline, the thread block scheduler utilizes a loose round-robin scheme based on prior work on spatial multitasking [2]. Once given an application's SM partitioning, the thread blocks of an application are scheduled to a static subset of SMs. Due to this mapping, the GPU's long-term reliability is directly tied to the consumption rate of workload.

Fixed scheduling: The fixed scheduling scheme triggers rotation after a fixed number of cycles has elapsed. This scheme can lead to sub-optimal consumption-rate balancing due to its fixed nature and applications' dynamic behavior.

Monitoring-aware scheduling: In the monitoring-aware scheduling scheme, we utilize the embedded EM sensor to make triggering decisions. The use of the EM sensors allows capturing dynamic application behaviors and their time-varying consumption rate usage. The monitoring-aware scheduler is given in algorithm 1. The index of each SM starts from 1 and ends at N; hence, the total number of SMs is N. We measure the accumulated consumption of each SM i (denoted as $AccumulatedCR_i$ in the algorithm), until the difference between the maximum and minimum accumulated consumption of two SMs exceed a certain threshold. We specify the threshold as a unit normalized to nominal average consumption rate. This ensures that the difference in consumption rate between SMs does not diverge too greatly from each other, leading to a balanced long-term reliability.

Monitoring-aware Scheduling;
while *Application is running* **do**
  foreach SM_i *in all SMs* **do**
    $CR_i \leftarrow$ Current EM sensor reading for SM_i ;
    $AccumulatedCR_i \leftarrow AccumulatedCR_i + CR_i$;
  end
  $CR_{max} \leftarrow \max_{1 \le i \le N}(AccumulatedCR_i)$;
  $CR_{min} \leftarrow \min_{1 \le i \le N}(AccumulatedCR_i)$;
  if $CR_{max} - CR_{min} \geq threshold$ **then**
    Rotate application's SM subset by 1 ;
    $\forall i : 1 \le i \le N$, clear $AccumulatedCR_i$;
  else
    Schedule round-robin to current subset ;
  end
end

Algorithm 1: Monitoring-aware Scheduling

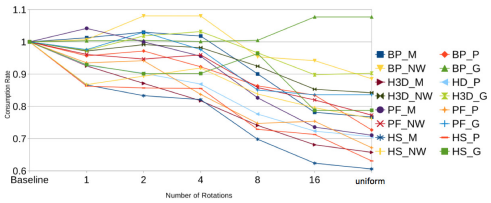

Fig. 5. Normalized Consumption Rate with Fixed Scheduling Policy

Uniform scheduling: Lastly, we compare against a uniform scheduling scheme serveing as target goal for long-term reliability. This scheduling policy does not take into account SM partitioning with the goal of maximizing consumption-rate balance across SMs.

VI. Experimental Results

To evaluate our long-term reliability GPU approach, we utilize the Rodinia benchmark suite. The consumption rate of 16 Rodinia benchmarks was shown previously in Fig. 1. We group the Rodinia benchmarks into two categories; high consumption(BP, H3D, PF, HS) and low consumption(M, P, NW, G).

We selected 8 benchmarks consisting of the 4 highest and lowest consumption rates. We then mix the benchmarks from each group to evaluate a multitasked GPU running a high consumption and a low consumption application so that there are 16 mixes. For our mixes, we allocate 8 SMs for the higher consumption application and 7 SMs for the lower consumption application. In section VI-B, we explore how our proposed techniques fair under different partitioning.

A. Fixed Scheduling Performance

Fig. 5 shows the results of fixed scheduling policy. The x-axis shows the number of rotations triggered using fixed intervals, and the y-axis shows the consumption rate, normalized to the baseline. In this experiment, we choose the period to trigger as the total runtime of the longer application, divided by the x-axis. This figure demonstrates both the effectiveness of a fixed scheduling technique along with the sensitivity to the period length. The right-most data points show the optimal consumption rate with the given mix. Using our 16 mixed benchmarks, as shown in Fig. 5, when the frequency of rotations increase (and the period of rotations decrease), overall consumption rate goes down because the work across each SM effectively becomes more and more uniform. This is why the consumption rate of each benchmark slowly gets closer to uniform. Fig. 7 shows that the fixed scheduler geometric mean has a 17.9% average improvement over the baseline.

B. Monitoring-Aware Scheduling Evaluation

In Fig. 7 we show the results of the baseline, fixed scheduling, monitor-aware scheduling, and uniform scheduling. Here the rotation

TABLE I. SENSITIVITY TO SM PARTITION FOR BACKPROP-MYOCYTE

	1:14	2:13	3:12	4:11	5:10	6:9	7:8	8:7	9:6	10:5	11:4	12:3	13:2	14:1
uniform	1.2079	1.1863	1.3825	1.3743	0.8122	0.6183	0.6796	0.7669	0.6052	0.6121	0.5795	0.9831	1.0713	1.0238
16	1.2048	1.1587	1.2895	1.3066	0.8436	0.6796	0.6119	0.7816	0.6497	0.6457	0.6443	0.9935	1.0884	1.0146
8	1.4017	1.4443	1.6696	1.6558	1.0514	0.8485	0.7456	0.9007	0.7055	0.6987	0.6482	0.9895	1.1402	1.0661
4	1.2156	1.3135	1.5632	1.6358	1.0998	0.9383	0.8923	1.0182	0.7943	0.8283	0.807	1.1017	1.1492	1.0587
2	0.9252	0.845	1.1264	1.4427	1.1471	0.9082	0.7568	1.03	0.8453	0.784	0.7849	1.0197	0.9904	0.9267
1	0.9982	0.9466	0.8854	1.2234	1.1695	1.0491	0.8823	1.0128	0.9384	0.9416	0.8606	1.0728	1.0239	0.9087
baseline	1	1	1	1	1	1	1	1	1	1	1	1	1	1

threshold of 7 was used for our monitoring-aware scheduler. A sensitivity analysis of the threshold value will be shown later in this section. In summary, our monitoring-aware scheduler reduces the consumption rate by 30% on average. This result is very close to that of the uniform scheme (average 31% reduction).

Monitoring-Aware Scheduler is able to outperform the Fixed Scheduler because it catches any dynamic workloads that is missed in the Fixed Scheduler if the rotation frequency is too low. This creates uniform workloads across all SM while also minimizing the number of rotations needed over the lifetime of the applications.

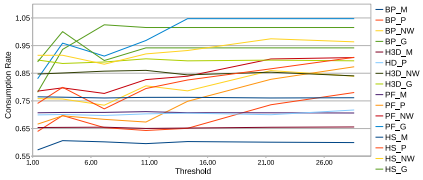

Fig. 6. Sensitivity to rotation threshold for monitoring-aware scheduling

C. Scheduling Overhead

Our scheduler incurs minimal overheads as we only modify the mapping policies between thread blocks and the target SM. However, the monitoring-aware schedule's rotation of an application's SM subset forces a context switch for all SMs. Our SM rotation can be supported in hardware by the fine-grained context switching mechanism found in Simultaneous Multikernel GPU [7]. In their paper, they found that context switching contributes, on average, a 2% throughput overhead over the application's lifetime. This is due to the extra time used for transferring data for the new context, as well as some loss of L1 cache locality. We found similar results during our experimentation.

D. Sensitivity to SM partitioning

Table I shows the consumption rate normalized to the baseline of a Backprop-Myocyte mixed workload. The numbers on the left column represent numbers of rotations triggered using fixed intervals. The top row shows the ratio of SMs provisioned for Backprop and Myocyte, respectively. This table shows several notable trends. First, reliability-aware scheduling is most effective when the number of SMs allocated to each application is relatively balanced, with an 11 to 4 ratio at most. Furthermore, it is also more effective if the low consumption-rate application has more SMs allocated. In addition, this table utilized the fixed scheduling policy in order to demonstrate the variance in benefits due to the static policy. In many of the scenarios, the consumption rate impact is variable as the fixed rotation cycle length decrease. This further supports the need for a dynamic reliability monitoring aware scheduler.

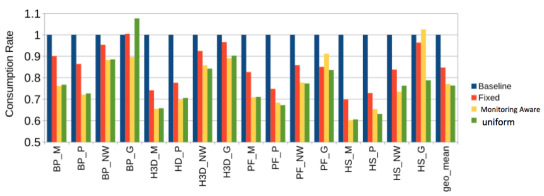

Fig. 7. Consumption Rate of Rodina Benchmarks

Fig. 6 shows the results of varying the rotation threshold of the monitoring-aware scheduler. The x-axis shows the threshold value and the y-axis show the consumption rate normalized to the baseline. A smaller threshold leads to more rotations, thereby leading to more balanced wear across SMs. In general, as the threshold increases, the consumption rate also increases due to less rotation. The consumption rate typically begins increasing with a threshold value of 10. This value means that we will trigger rotation when the maximum and minimum consumption rate differs by 10 times the nominal consumption rate. Therefore, in our scheme, we select a threshold of 10 as it gives a balance between minimizing the number of rotations while still providing a lower consumption rate and it is used throughout the experimental evaluation.

VII. CONCLUSION

In this paper, we proposed thread-block schedulers for GPU long-term reliability against electromigration (EM)-induced failure. We modeled the EM-induced time-to-failure (TTF) at the GPU system level as a reliability metric. For GPU architectures, we focused on spatial multitasking which allows GPU resources to be partitioned among multiple applications. We developed long-term reliability-aware thread-block schedulers in GPGPU-sim and compared them against the existing approach. Experimental results indicate that our proposed scheme improves GPU system-level reliability by 30% on average. To the best of our knowledge, this work is the first approach to quantify and address the EM-induced reliability of GPU architectures.

REFERENCES

[1] D. Tiwari, S. Gupta, G. Gallarno, J. Rogers, and D. Maxwell, "Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*, pp. 1–12, 2015.

[2] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The case for gpgpu spatial multitasking," in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, HPCA '12, (Washington, DC, USA), pp. 1–12, IEEE Computer Society, 2012.

[3] S. X.-D. Tan, H. Amrouch, T. Kim, Z. Sun, C. Cook, and J. Henkel, "Recent advances in EM and BTI induced reliability modeling, analysis and optimization," *Integration, the VLSI Journal*, 2017. in press.

[4] S. X.-D. Tan, M. Tahoori, T. Kim, S. Wang, Z. Sun, and S. Kiamehr, *VLSI Systems Long-Term Reliability – Modeling, Simulation and Optimization*. Springer Publishing, 2019.

[5] Z. Lu, W. Huang, J. Lach, M. Stan, and K. Skadron, "Interconnect lifetime prediction under dynamic stress for reliability-aware design," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 327–334, IEEE, November 2004.

[6] K. He, X. Huang, and S. X.-D. Tan, "EM-Based on-chip aging sensor for detection and prevention of counterfeit and recycled ICs," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, Nov. 2015.

[7] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel gpu: Multi-tasking throughput processors via fine-grained sharing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 358–369, March 2016.