© 2018 IJSRST | Volume 4 | Issue 5 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Light-weight Process variation-aware instruction distribution algorithm for Embedded GPUs

K. S. Balamurugan, G. P. Ramacharyulu, R. Sathish Kumar
Associate Professor, Department of ECE, Bharat Institute of Engineering and Technology, Hyderabad, Telangana, India

ABSTRACT

In the Embedded systems, Graphics Processing Units (GPUs) are used to manage the huge number of computation and to convince the timing limit. Future size, chip aging and die-parameter variations are challenging issues in GPUs. To solve the process variation issues, some processors operate lowest operating frequency in chip-level guard banding, that effects reduce the performance in chip-level. Some other processors improve their performance through core-level guard banding that possibly use various operating frequency for every core. Presents of Process variation, each cluster has a dissimilar level of degradation for the equal amount of instructions. For that reason, a process variation-aware instruction distribution algorithm is essential to balance the stress across the embedded GPU. Suggested light-weight process variation-aware instruction distribution algorithm estimate weight factor of the process variation, level of stress, the current aging status, cluster information (availability and operating frequency) and hierarchy of Real-time Application then gives a various level of instructions to clusters to reduce the aging effect. Simulation results shown that suggested technique improved the GPU aging in 85% and reduce the 40% overhead than compiler-based technique.

Keywords: aging-aware, process variation embedded GPUs.

I. INTRODUCTION

Graphics Processing Units (GPUs) has required to progress the system performance in GPUs based embedded systems such as mobile devices to satisfy the timing constraint (H. Lee et al., 2016). Due to the small size, nano-scale multi-core processors, including GPUs, have faced several reliability challenges such as aging effects and (die-to-die and with-in-die) parameter variations (Aguilera et al., 2014). Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) are considered among the most critical aging-related reliability challenges in nano-scaled semiconductors(J. Sun, 2014).The amount of transistor degradation caused by NBTI and HCI is proportional to the time a transistor is stressed or switched (X. Chen et all., 2014). Various workload management techniques have been proposed to balance the stress level and reduce switching activity across the chip to minimize NBTI and HCI effects. To improve the special effects of process variation, normally, multi-core processors ware employed a chip-level guard-banding method, which operates at low frequency. Core-level guard banding allow the various frequencies for every core. In Core-level guard-banding, every core of a single-chip have a various operating frequency and duty cycle, which directs to changeable stress levels and thermal variations in the chip. To address the above challenges, proposed algorithms to calculate approximately the aging status of the GPU and allocate the workload across the GPU with respect the process variation and the core-level guard-banding. To express the unbalanced workload distribution with the process variation

and core-level guard-banding, carry out experiments by real world applications. GPGPU-Sim simulator with NVIDIA's Tegra TK1 configuration is used. The SP and SFU units are mapped into a process variation map that is generated in (D.Gnad, 2015). During the experiments, evenly distribute the instructions to the SP and SFU units. Due to the PV and cluster-level guard-banding, the stress is not equally spread across the GPU even if every cluster processes the equal number of instructions. (H. Lee et al., 2017) point out that the current warp schedulers and the instruction dispatchers have a limitation in minimizing the NBTI and HCI effects in the process variation. Most of research suggested minimizing the effects of NBTI and HCI on multi/many-core systems. (J. Henkel et al., 2013) surveys different trade-off points connecting with power, A job migration method (M.Banda et all., 2013) is planned to reduce the effect of NBTI. The planned method uses the spare processor to migrate responsibilities from the near-to-die processor to the young processor. But, this technique is not relevant to the GPUs. The proposed techniques assign the application's function to processors based on the application information and the current NBTI status. (T. R. Mck et al., 2017) Job mapping technique for heterogeneous multiprocessors (HMP) is proposed. Based on the application profile, the sensing data, performance counters and the job are mapped to various processors to get better chip era. Conversely, each and every one the cores in the GPU carry out the same kernel function, it is rigid to allot special kernel functions to core on the GPU. Besides, obtainable GPUs do not permit mapping between the tasks and cores on the GPU when the mapping is controlled by the hardware. A thread-to-core mapping method is planned in (M. Shafiqu et al., 2015) to take full advantage of the performance of a many-core processor.

(Lotfi et al., 2015) suggested compiler-based techniques to decrease the NBTI effect and increase the lifetime of the GPU. In run-time, the Just-In-Time (JIT) compiler generates a strong kernel function based on the current aging status of the GPU. Strong kernel needs supplementary workloads to reassign the workloads from the corrupted cores to strong cores. The amount of extra workload depends on the number of corrupted cores. The proposed procedure founds the most favorable amount of SMs to manage the GPU applications and direct the clock signal at SM granularity. But the embedded GPUs contain few SMs only. Different scheduling properties are assigned to each application. (H. Lee et al., 2016) proposes a fine-grained GPU resource management framework that framework partitions the GPU workload into tiny-workloads to execute preemption. Since this outline just considers periodic applications, it has restrictions in partitioning random GPU workloads during run-time. Few proposed scheduling framework for event-driven real-time systems that generates the mapping between the Streaming Multiprocessors (SMs) and the applications in order to offer temporal and spatial preemption. For that reason, well-established workload management technique is needed to reconfigure the cluster of cores and distributes the workload to reduce the NBTI and HCI effects in the embedded GPUs. In this circumstance, Suggested Light-weight Process variation-aware instruction distribution algorithm, allots the instruction to cluster with respect PV for improve the ageing in embedded GPU. The rest of the document is prearranged as follows. In Section II, Problem formation is illustrated for improve the ageing-aware in embedded GPU Section III, Proposed algorithm was explained step-by-step. In Section IV, simulation results are discussed and finally section V, bring to a close the research work.

II. PROBLEM FORMATION

Aging factor depends on the actual stress/recovery phase, degradation by NBTI depends on supply voltage Vdd, temperature T, threshold voltage V_{th}, and device parameters, comprising for instance oxide geometrical and electrical parameters, activation

energy, device size, and load. Transistors age mainly when they are under stress (NBTI) and switch their state (HCI).

Table 1. Basic information in NBTI and HCI

NBTI	PMOS-Under stress	V_{th} is increased due to traps, which are generated in the interface between the oxide layer and silicon/channel.
	PMOS-Not under stress	V_{th} is decreased (recovery phase)
HCI	NMOS	Collision between the accelerated electrons and the gate oxide interface generates electron-hole pairs.

In NMOS transistors, the collision among the accelerated electrons and the gate oxide interface generates electron-hole pairs. After that, free electrons get trapped in the gate oxide layer and the V_{th} is increased since NMOS transistor changes its state, the total amount of V_{th} shift is most sensitive to the number of state transitions. In short, HCI depends upon the switching activity. (H.Lee et al., 2017) proposed a fresh technique to reduce the NBTI and HCI effects on an embedded GPU under the process variation. The proposed technique includes an aging-aware cluster formation algorithm that creates core clusters based on the present NBTI and HCI information. In suggested algorithm, each cluster has less aging variation and aged core clusters are power-gated during run-time. A process variation-aware workload distribution algorithm that generates the instruction distribution level based on the process variation information. The proposed algorithm takes aging and the process variation information about active core clusters and generates the instruction distribution ratio to evenly distribute the stress across the GPU under the process variation. An instruction distribution unit that incorporates the existing warp scheduler and instruction dispatch unit. During run-time, right before the kernel launch, the proposed instruction distribution unit is configured based on the weight factor distributed algorithms. After the configuration, the instructions are distributed based on the instruction distribution level to minimize the NBTI and HCI effects.

Aging-Aware Cluster Formation

The degradation of the components in a multi/many-core processor is closely related to stress and power management. Moreover, the aging gap between the clusters will become larger over time due to the process variation, cluster-level guard-banding, and unbalanced stress distribution. In order to minimize the effects of the process variation and the cluster-level guard-banding, the proposed aging-aware cluster formation algorithm (re)configures the clusters by using the current aging information. Before the host launches a kernel function, the host obtains the current aging information for all the cores in the GPU through the delay monitors. Then, the host sorts the cores based on the aging information in descending order and groups the cores to create the clusters. Thus, each cluster has cores that have similar degradation levels and minimum aging variations. The sorting and clustering process do not need to consider the process variation at this point because the amount of degradation is the result of the process variation. After that, each cluster sets its operating frequency by finding a core with minimum operating frequency. Since the entire GPU may not be required to execute the kernel function, after the clustering, the proposed aging-aware cluster formation algorithm selects some degraded clusters for power-gating based on the GPU resource utilization information, which is generated in design-time.

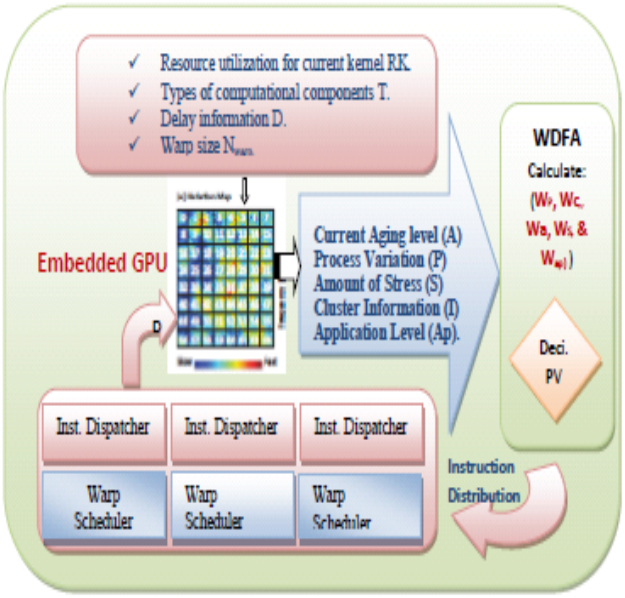

Figure 1. Proposed architecture for Light Embedded GPUs.

Process Variation-Aware Workload Distribution Algorithm

Due to the process variation, the same number of instructions causes a different amount of degradation for each cluster. In order to evenly distribute the stress across the GPU, each working cluster should process a different number of instructions. The proposed process variation-aware workload distribution algorithm estimates the instruction distribution ratio between the active clusters. For each kernel, the proposed algorithm assigns the same number of instructions for each working cluster and gets the aging estimation. Each working cluster will have a different aging status due to the different operating frequencies, which are caused by the process variation. Then, the proposed algorithm gets the average from the aging estimation of working clusters and sets this average as a desired aging status after executing the kernel function. The amount of stress for each cluster can be obtained by subtracting the current aging status from the desired aging status. By using the amount of stress and the process variation information, the instruction distribution ratio is estimated.

Applications-aware GPU resources (re)allocation

While driving, mixture of safety critical and non-safety applications are launched by the system depending on the current system status. For example, when driving on a highway, the system would launch a set of safety critical and high priority applications for vehicle detection. On the other hand, when driving in a school area, the system would launch a different set of safety critical and high priority applications for pedestrian detection and traffic sign recognition. At the same time, non-safety critical and low priority application may be running on the system. Each application would have a different priority and deadline depending on the current system status. Note that in this type of embedded system, small timing violations may cause degradation in Quality of Service (QoS), such as glitches in an instance of a traffic sign image. However, small quality degradation would not cause the system to fail. Therefore, the system is a soft real-time system and the QoS of the system highly depends on the status of safety critical and high priority applications. The example scenario indicates that the GPU needs to be efficiently assigned and provide preemption capabilities, in order to meet the deadlines of the applications. Moreover, the results imply that the performance overhead of launching multiple sub-kernels may be relatively small compared to the execution time, of the kernel, within a single launch. The problem of run-time scheduling on a GPU-based embedded system poses the following research challenges: How to partition GPU kernels into multiple sub-kernels during run-time, such that multiple application kernels could concurrently occupy the GPU, where the number of the applications that meet their deadline is maximized and How to dynamically generate a schedule for the GPU that would increase the number of high priority applications meeting the deadline as much as possible. In order to address the mentioned challenges, proposed a novel scheduling framework to partition

GPU application kernels and generate sub-kernel launch sequences, for the GPU during run-time. Based on the current status of GPU-based embedded systems and application deadlines, the proposed GPU execution schedule generator generates sub-kernel launch sequences that maximize the number of applications meeting deadlines.

Instruction distribution unit

The proposed instruction distribution unit incorporates the existing warp scheduler and instruction dispatch unit to balance the stress across the GPU. The host configures the instruction distribution unit using the instruction distribution ratio before launching a kernel function. After the host launches a kernel function, the instruction distribution unit controls the behavior of the warp scheduler and the instruction dispatch unit to evenly distribute the stress over the GPU. When an instruction is ready, the instruction dispatch unit sends the instruction to the first available cluster and increases the corresponding instruction distribution counter. When all the instruction distribution counters reach their limits, then the instruction distribution unit resets the counters and sends the instruction to the first available cluster. As discussed in Section 1, the basic execution unit of the target embedded GPU is a warp, which represents the behavior of 32 cores. The instruction distribution unit needs to track the instruction distribution ratio and status for 8 clusters of cores (6 SP clusters and 2 SFU clusters). This can be done with eight 16-bit registers and counters. In addition, the existing warp scheduler and instruction dispatcher need to be modified with additional control signals and to change their behavior depending on the status of the instruction distribution. The area and power consumption overheads of the above-mentioned logic are therefore negligible, i.e., only a few registers/counters. The performance overhead results are presented in Section iv.

III. PROPOSED WORK

3.1. Weight Factored Distribution Algorithm

In this section, the Weight Factored Distribution Algorithm estimates the weight factor of following input parameters like mention given below.

- ✓ Current Aging level (A)
- ✓ Process Variation (P)
- ✓ Amount of Stress (S)
- ✓ Cluster Information (Availability, Operating Frequency) (I)
- ✓ Application Level (Ap).

The weight factor distribution algorithm precedes the parameters of the cluster/wrap as inputs and creates weight factors with respect to an application specified needs. The weight factors are evaluated in order to find the levels of the parameters desired to reach improved performance.

Procedure for WFD Algorithm is,

Step 1: Following are the assumptions thought-about,

- The Availability of Cluster and operating frequency as describe the Cluster information level of the Cluster Cw, and Process variation level Pw wherever $0 < Pw < 1$, ($Pw = 0$ means that the all cluster in same level- run out wrap scheduler algorithm, clusters are utmost good and $Pw = 1$ means that the Clusters are dissimilar performance.)
- The Weight Factors of the four parameters, obtainable Cluster information, Current Aging level, amount of Stress and process variations are Wc, Wa, Ws and W_P severally, wherever Wc = 1 and $W_C + Wa + W_S + W_P = 1$.

The factors that reason importance levels like high, medium, low and none are I_H, I_M,I_L and 0, respectively, Wherever their values are determined

by the system designer, and 0 <I_H <I_M<I_L< 1. The numbers of various importance levels the user has such that are N_H, N_M, N_L and N_N respectively, Wherever $N_H + N_M + N_L + N_N = 3$ (since the satisfied aging level of the Cluster parameters that a user may Specify is three)

Step 2: The weight factor of the four vital levels when adjusted to user preferences and battery power are WI_H, WI_M, WI_L and WI_N, respectively.

$$(N_H * WI_H) + (N_M * WI_M) + (N_L * WI_L) + (N_N * WI_N) = P_w$$

$$WI_M = WI_H * \frac{I_M}{I_H}$$

$$WI_L = WI_H * \frac{I_L}{I_H}$$

$$WI_N = 0$$

$$(N_H * WI_H) + \left(N_M * WI_M * \frac{I_M}{I_H}\right) + \left(N_L * WI_H * \frac{I_L}{I_H}\right) = P_w$$

Step 3: The Weights of four importance levels are evaluated by using the following calculations

$$WI_H = \frac{\frac{I_H}{P_W}}{(N_H * I_H) + (N_M * I_M) + (N_L * I_L)}$$

$$WI_M = \frac{\frac{I_M}{P_W}}{(N_H * I_H) + (N_M * I_M) + (N_L * I_L)}$$

$$WI_L = \frac{\frac{I_L}{P_W}}{(N_H * I_H) + (N_M * I_M) + (N_L * I_L)}$$

$$WI_N = 0$$

From these formulas the weight factor ranks of each parameter are evaluated. These weights factors values are set as the input to the suggested light weight process variation-aware instruction distributed algorithm and estimate the instruction distribution ratio to clusters.

1. The proposed algorithm receives the following information as input: resource utilization for current kernel RK, Types of computational components T, delay information D, and warp size Nwarp.
2. The computational components are sorted based on the current delay information.
   After that, based on the warp size, Nwarp, the components are clustered and the cluster information is updated.
3. Next, the proposed WFD algorithm selects clusters for power-gating based on the resource utilization information.
   The remaining clusters are selected as the running clusters.
4. The operating frequency is selected for working clusters.
   At the end, the clustering information, which includes power-gating, and the operating frequencies are returned.
5. Takes the working cluster information $\mathbf{C}_{work}$, the process variation information **PV**, and the number of instructions N_{inst} as inputs.
6. At the beginning, the algorithm evenly distributes the instructions to working clusters. The algorithm estimates the aging status of each working cluster using the number of

instructions neven inst and the process variation information.

7. Then, the average aging is estimated based on the total amount of aging. After estimating the average aging information, the algorithm estimates the amount of stress for each working cluster.
8. Calculate weight factor of stress and the process variation, the algorithm decides the number of instructions for each cluster.
9. Finally, the algorithm returns the instruction distribution level for each working cluster.
10. After the configuration, the host launches a kernel function and the instructions are fetched. For each instruction, the algorithm gets the instruction type and the corresponding instruction distribution level.
11. Next, the instruction is sent to the best suitable working cluster and the algorithm increases the instruction distribution counter. At last, the algorithm resets the instruction distribution counter if all the instruction distribution counters reach their limit.
12. After the configuration, the host launches a kernel and the instruction distribution unit starts fetching and distributing the instructions based on the instruction distribution level.

IV. SIMULATION RESULTS

Experimental Setup

We have extensively evaluated our framework by comparing it to several state-of-the-art existing frameworks. We have built the simulator that represents our target GPU-based embedded system and the simulator is assumed to resemble Nvidia's Tegra mobile embedded system. We also assumed that our target GPU has a total of 13 SMs and the off-chip memory is shared by the CPUs and the GPU. We have selected various benchmark applications from NVIDIA's Compute Unified Device Architecture (CUDA) toolkit to evaluate our technique with different computational workloads for the embedded GPUs. We have generated 50 different process variation maps by using the area information in GPGPU-Sim and the process variation model from(H.lee et al., 2016). In addition, the aging traces of SO and MC applications are collected over the 10 years of period to observe the long term aging behavior. During the experiments, the aging traces are generated for each process variation map. The compiler-based technique shows varying aging improvements for different process variation maps, whereas our technique and the even distribution technique consistently improve the aging of embedded GPUs. This is because the compiler-based technique only disables the aged clusters and sends all the workloads to the healthy clusters. In addition, our results show that the ranges of aging distribution are increased over time with the compiler-based technique and the original application. This is because the compiler based technique and the original application do not consider the process variation and the core-level guard-banding properly.

Table 2: Standard Application and Configuration

application	Short Form	Grid Size	Block Size
Binomial Option	BN	16	128
Convolution Separable	CONV	288	64
FastWalshTransform	FWT	128	256
SobelFilter	SF	1024	64

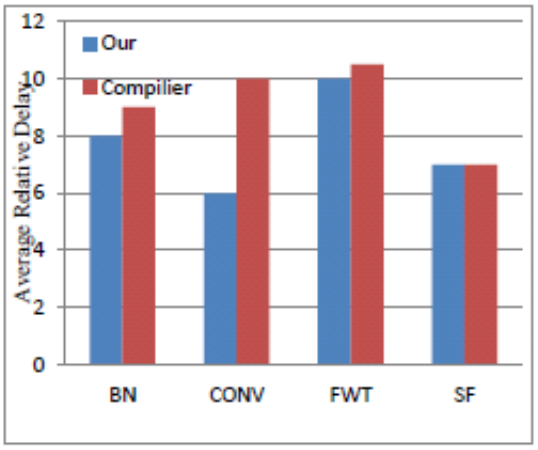

Figure 2: Relative Delay for Std. Application Compared to the Compiler-based Technique.

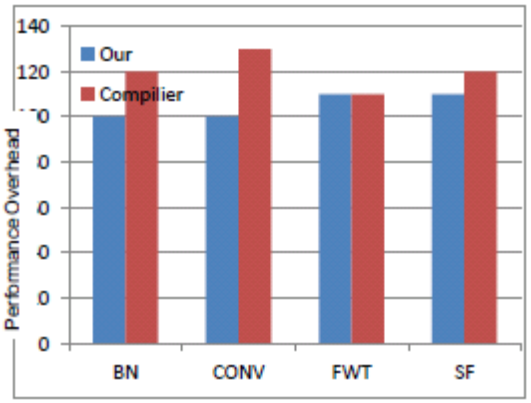

Figure 3: Performance Overhead Compared to the Compiler-based Technique

We selected various benchmark applications to represent the computational workload for the embedded GPUs. Table 2 shows the list of the benchmark applications and their configurations. During the experiments, we extracted duty cycle information of each application from the GPGPU-Sim. we can observe that the even distribution and the compiler-based technique have lower success rates compared to our technique. This is because the process variation and the cluster-level guardbanding may cause a different amount of stress for the same number of instructions and increase the randomness in the system state. Since this random behavior is not considered by the other techniques, they have lower success rate compared to our technique. We measure the relative standard deviation of aging of SP/SFU units to observe the impact of the above mentioned randomness. Figure 2 shows the average relative standard deviation for all the 50 process variation maps. We can observe that the aging is well balanced across the embedded GPU with our technique. However, the other techniques do not show the balanced aging distribution. This is because other techniques' optimization may not capture the randomness of the process variation. The results in Figure 2 and 3 imply that other techniques may worsen the aging of embedded GPU without proper consideration for process variation.

V. CONCLUSION

We proposed a Light-weight Process variation-aware instruction distribution algorithm for Embedded GPUs. By extending the functionality of the existing warp scheduler and instruction dispatcher, the workload is distributed across the embedded GPU to minimize its aging and the performance overhead. The warp formation and workload distribution algorithms generate information to (re)configure the cluster and balance the stress across an embedded GPU. Then, the host configures the GPU with the results from the algorithms before it launches a kernel function. After that, the GPU distributes the instructions based on the instruction distribution ratio. Compared to the original applications, our technique improves the aging of the embedded GPU by 30% on average. Moreover, compared to the state-of-the-art technique, our technique further improves the aging of the embedded GPU by 3% on average while reducing the performance overhead by 16.4% on average. These experimental results show that our technique may minimize the aging of the embedded GPU while maximizing the probability to meet the timing requirements of the system. Moreover, our technique has less soft-error susceptibility than the state-of-the-art technique due to the reduced performance overhead.

VI. REFERENCES

[1]. H. Lee, M. Shafique, and M. A. A. Faruque. "Low-overhead Aging-aware Resource Management on Embedded GPUs". 54th ACM/EDAC/IEEE Design Automation Conference (DAC'17), pages 1–6, 2017.

[2]. P. Aguilera, J. Lee, A. Farmahini-Farahani, K. Morrow, M. Schulte, and N. S. Kim. "Process variation-aware workload partitioning algorithms for GPUs supporting spatial-multitasking". Design, Automation Test in Europe Conference Exhibition (DATE'14), pages 1–6, March 2014.

[3]. F. Kriebel, S. Rehman, M. Shafique, and J. Henkel. "ageOpt-RMT: Compiler-driven variation-aware aging optimization for redundant multithreading". 53nd ACM/EDAC/IEEE Design Automation Conference (DAC'16), pages 1–6, June 2016

[4]. Q. Xu and M. Annavaram. "PATS: Pattern Aware Scheduling and Power Gating for GPGPUs". Proceedings of the 23rd International ConferenceonParallelArchitecturesandCompilation(PACT'14),pages 225–236, 2014.

[5]. Y. Zhang, S. Chen, L. Peng, and S. Chen. "NBTI alleviation on FinFET-madeGPUsbyutilizingdeviceheterogeneity". Integration, the VLSI Journal, 51:10–20, 2015.

[6]. Rahimi, L. Benini, and R. Gupta. "Aging-aware CompilerdirectedVLIWAssignmentforGPGPUArchitectures". Proceedings ofthe50thAnnualDesignAutomationConference(DAC'13),pages1-6, 2013.

[7]. K.S.Balamurugan and Sri Sahithi. "Log Likelihood Ratio Based Quantizer design for target tracking in wireless sensor networks", IJSRST, 2018.

[8]. D. Mirzoyan, B. Akesson, and K. Goossens. "Process-variationaware Mapping of Best-effort and Real-time Streaming Applications to MPSoCs". ACM Transactions on Embedded Computing Systems, 13(2s):1–24, Jan. 2014.

[9]. T. R. Mck, Z. Ghaderi, N. D. Dutt, and E. Bozorgzadeh. "ExploitingHeterogeneityforAging-AwareLoadBalancinginMobile Platforms". IEEE Transactions on Multi-Scale Computing Systems, 3(1):25–35, Jan 2016

[10]. F. Kriebel, S. Rehman, M. Shafique, and J. Henkel. ageopt-rmt: Compiler-driven variation-aware aging optimization for redundant multithreading. 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC'16), pages 1–6, June 2016.

[11]. H. Lee, H. Chen, and M. A. A. Faruque. "PAIS: Parallelization aware instruction scheduling for improving soft-error reliability of GPU-based systems". Design, Automation Test in Europe Conference Exhibition (DATE'16), pages 68–73, 2016.

[12]. H. Lee and M. A. A. Faruque. "GPU-EvR: Run-Time Event Based Real-Time Scheduling Framework on GPGPU Platform". Design, Automation and Test in Europe Conference and Exhibition (DATE'14), pages 1–6, 2014.

[13]. H. Lee and M. A. A. Faruque. "Run-Time Scheduling Framework for Event-Driven Applications on a GPU-Based Embedded System". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35(12):1956–1967, 2016.

[14]. Lotfi, A. Rahimi, L. Benini, and R. Gupta. "Aging-Aware Compilation for GP-GPUs". ACM Transactions on Architecture and Code Optimization (TACO'15), pages 1–20, 2015.

[15]. Y. Zhang, S. Chen, L. Peng, and S. Chen. "NBTI alleviation on FinFET-made GPUs by utilizing device heterogeneity". Integration, the VLSI Journal, 51:10–20, 2015.

[16]. S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel. "dTune: Leveraging Reliable Code Generation for Adaptive Dependability Tuning

Under Process Variation and Aging-Induced Effects". Proceedings of the 51st Annual Design Automation Conference (DAC'14), pages 1–6, 2014.

[17]. J. Tan, M. Chen, Y. Yi, and X. Fu. "Mitigating the Impact of HardwareVariabilityforGPGPUsRegisterFile". IEEETransactions on Parallel and Distributed Systems, 27(11):3283–3297, 2016.

[18]. J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda. "Workload Assignment Considering NBTI Degradation in Multicore Systems". ACM Journal on Emerging Technologies in Computing Systems (JETC'14), pages 1–22, 2014.

[19]. D.Gnad,M.Shafique,F.Kriebel,S.Rehman,D.Sun,andJ.Henkel. Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. Proceedings of the 52nd Annual Design Automation Conference (DAC'15), pages 1–6, 2015.

[20]. J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn. "Reliable on-chip systems in the nanoera: Lessons learnt and future trends". 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13), pages 1–10, May 2013.

[21]. M.Bandan,S.Bhattacharjee,R.Shafik,D.Pradhan,andJ.Mathew. "Lifetime Reliability-Aware Checkpointing Mechanism: Modelling and Analysis". 2013 International Symposium on Electronic System Design (ISED'13), pages 128–132, 2013.