DOI 10.1109/LES.2020.3021854

Aging-aware Parallel Execution

Thiarles S. Medeiros*, Gustavo P. Berned*, Antoni Navarro§, Fábio D. Rossi†, Marcelo C. Luizelli*,
Marcelo Brandalero¶, Michael Hübner¶, Antonio Carlos S. Beck‡, and Arthur F. Lorenzon*
*Optimization System Laboratory - Federal University of Pampa, Brazil
†Campus Alegrete, Federal Institute Farroupilha, Brazil
‡Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
§Barcelona Supercomputing Center, Barcelona, Spain
¶Brandenburg University of Technology, Germany

***Abstract*—Computation has been pushed to the edge to decrease latency and alleviate the computational burden of the IoT applications in the cloud. However, the increasing processing demands of Edge Applications make necessary the employment of platforms that exploit thread-level parallelism (TLP). Yet, power and heat dissipation rise as TLP inadvertently increases or when parallelism is not cleverly exploited, which may be the result of the non-ideal use of a given PPI (Parallel Program Interface). Besides the common issues, such as the need for more robust power sources and better cooling, heat also adversely affects aging, accelerating phenomenons such as negative bias temperature instability (NBTI) and hot-carrier injection (HCI), which further reduces processor lifetime. Hence, considering that increasing the lifespan of an edge device is key, so the number of times the application set may execute until its end-of-life is maximized, we propose BALDER. It is a learning framework capable of automatically choosing optimal configuration executions (PPI and number of threads) according to the parallel application at hand, aiming to maximize the trade-off between aging and performance. When executing ten well-known applications on two multicore embedded architectures, we show that BALDER can find a nearly-optimal configuration for all our experiments.**

***Index Terms*—Parallel Computing, Aging, NBTI, HCI**

I. Introduction

THE number of cores in a single chip has been increasing to meet the demands of applications at the edge running on top of high-end embedded systems (e.g., facial recognition, human body interaction, or neural networks). However, heat dissipation has becoming more significant, since power dissipated per area raises at each new node generation (i.e., the well-known end of Dennard scaling [1]). Besides common issues such as cooling, it also stimulates the aging process of hardware components, resulting in undesired system behavior and reducing their lifetime. Aging also makes hardware components more susceptible to different types of failures (e.g., electromigration and dielectric breakdown).

Considering that aging phenomena (e.g., NBTI and HCI) are highly influenced by many factors, such as the temperature, supply voltage, and operating frequency [2], controlling these hardware-related characteristics is essential to reduce aging effects. Nonetheless, when a parallel application is running, the number of concurrent threads influences the processor temperature as a result of the increased switching activity in the hardware components (e.g., cores and cache memories). This temperature rise is related to (*i*) the number of threads distributed across cores; (*ii*) the communication model employed by threads or processes; and (*iii*) how synchronization is performed. All these factors are related to the underlying parallel programming interfaces (PPIs) used.

Manuscript received XXX; revised YYYY. Corresponding author: A. F. Lorenzon (email: aflorenzon@unipampa.edu.br).

PPIs speed up the development of parallel applications and make it as much transparent as possible to the programmer. OpenMP - Open Multi-Processing [3], PThreads - POSIX Threads [4], or MPI - Message Passing Interface [5] are some of the most popular ones. However, each one of these has different characteristics regarding the management of threads, workload distribution, synchronization, and communication [6]. Therefore, each PPI will not only influence the application performance, but also the aforementioned hardware variables, affecting the processor aging in different ways. However, we show that there is no single combination of PPI and the number of threads that maximizes the lifetime of an embedded device while keeping performance level as high as possible for all applications or devices.

Considering that increasing the lifespan of an edge device is key, so the number of times the application set may execute until its end-of-life is maximized, we propose BALDER. It is a framework that employs a learning algorithm to find the combination of PPI and number of threads for a given application that maximizes the trade-off between performance and aging. BALDER exploits the fact that many applications may present the same behavior in terms of TLP even when different input sizes are considered, which enables it to apply its learning algorithm using a smaller input size instead of the original one (which is used during the application execution). With that, BALDER significantly reduces the learning cost (i.e., the time spent to find the best configuration). By executing ten well-known algorithms implemented with four of the most widespread PPIs (OpenMP, PThreads, MPI-1, and MPI-2) on two multicore embedded systems, we show that BALDER find a nearly-optimal configuration, but with an overall learning time (i.e., time for converging to a solution) 97.9% lower than an exhaustive search.

II. Related Work

Namaki et al. [7] propose an aging-aware technique for register files on GPGPUs. Shafique et al. [8] present a content-aware micro-architectural-level technique to reduce the aging of SRAM-based memories. Gnad et al. [9] propose a runtime

system that leverages dark silicon and process variations to optimize the NBTI-induced aging. Hanumaiah and Vrudhula [10] propose a thermal constraint technique to determine the CPU frequency and voltage that guarantee tasks completion within a deadline. Khdr et al. [11] propose a DVFS-based boosting technique to reduce the aging effects that are induced by higher temperatures.

Rahimi et al. [12] present a very-long instruction word (VLIW) reallocation strategy for reducing the aging of GPGPU architectures. Mulas et al. [13] propose a thermal balancing policy that exploits tasks migration for MPSoCs architectures. Chantem et al. [14] present a solution for assigning and scheduling tasks on a MPSoC architecture to reduce the processor aging. Lee et al. [15] propose a workload management technique that considers the process variation and aging status together to reduce the aging of embedded GPUs. Rathore et al. [16] uses a reinforcement learning-based strategy to map tasks in many-core systems to improve system health.

Our contributions. To the best of our knowledge, this is the first work that assesses the design space exploration regarding the use of different PPIs and degrees of TLP to maximize the number of times an application set may execute until the processor's end-of-life. Besides that, BALDER is the first approach that automatically finds the best combination of PPI and number of threads for a given application. Hence, compared to all previous works, BALDER is orthogonal and can be used with them in order to further reduce aging.

III. BALDER

BALDER aims to maximize the number of applications that may be executed by an embedded device until its end-of-life by selecting the ideal combination of PPI and degree of TLP. For that, it considers as optimization metric the trade-off between performance (the execution time of a given application) and the total aging due to NBTI and HCI. The workflow of BALDER is shown in Figure 1. The user provides application binaries (one for each *PPI*), and both small and regular problem sets, which are the inputs of BALDER. If the application has not yet been trained, BALDER applies the learning algorithm (Algorithm 1) over the execution of the application binaries and small input set to find the combination (PPI-TLP) that optimizes the target metric. Once this configuration is found, it is stored into the BALDER's database to be used next time the application is executed. Otherwise, if the application has already been trained, BALDER access the database to get the best configuration for its input (e.g., application and regular input set). Then, the application is executed with the best configuration and the regular input set. BALDER works with applications parallelized with OpenMP, PThreads, MPI-1, and MPI-2. We implemented BALDER in the Python3 language in the way that users only need to provide application binaries and input sets (small and regular).

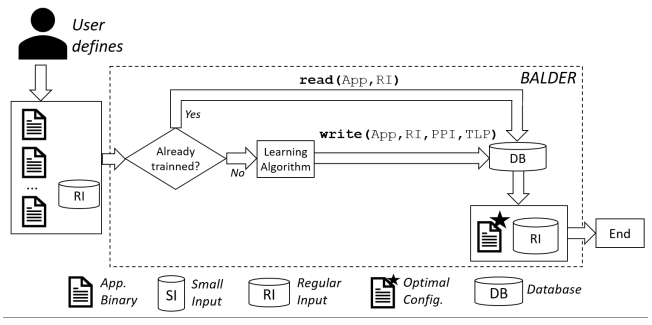


Fig. 1: BALDER Execution Flow

A. Modeling Aging Phenomena

We use the aging model from [15], [17], which takes into account both the effects of NBTI and HCI. Both these phenomena lead to increases in the threshold voltage (V_{th}) of MOSFET devices over time and, consequently, to slower switching speeds. Ultimately, devices have a lifetime after which the timing paths become larger than the clock period, and thus the timing errors that show up forbid any reliable computation. In the model from [15], [17], depicted in Fig. 2, given the processor temperature (T, in Kelvin), the supply voltage (V_{dd}), the sampling period (t_m), the processor frequency (f), the $\Delta V_{th(NBTI)}$ and $\Delta V_{th(HCI)}$ at time $t > 0$ can be estimated by Eqs. 1 and 2[1] (Fig. 2). To calculate the duty cycle of each core running the application (δ_C), we consider the ration of time that each core is under stress. The same approach was used to get the activity factor (α_C).

B. Learning Algorithm

BALDER's learning algorithm is given an application *A* with a *small input set S* that can be executed by *c* distinct cores $C = \{C_1, ..., C_c\}$ and with p PPIs $P = \{P_1, ..., P_p\}$. The optimization problem we are interested in seeks an assignment of the application *A* to a subset of *threads/cores* in C and PPI in P. We denote by C^* the set of all subsets of threads/cores in C, that is, $C^* = \cup_c \binom{C}{c}$ and P^* the set of all *p* subsets of PPIs, that is, $P^* = \cup_p \binom{P}{p}$. A feasible assignment can be defined as a function $\phi : A \rightarrow C^* \times P^*$ that assigns a subset of threads/cores and PPI to a parallel application. Therefore, we denote by $\mathbb{M} : (A \times C \times P) \rightarrow \mathbb{R}^+$ the measured metric

[1]Constant values were taken from [17]–[19]: $n = 1/6$, $E_0 = 0.335$, $E_a = 0.49$, $E_1 = 0.8$ $C = 0.0163$, $\xi 1 = 0.9$, $\xi 2 = 0.5$, $\xi 3 = 1.0$, $\xi 4 = 10^{-8}$, $r = 1.6$, A_N and A_H accordingly to [19], and k is the Boltzmann constant. For the other values, BALDER uses data obtained from hardware, with $t_m = 1s$. T is read either with the *vcgencmd measure_temp* command or *lm-sensors*; V_{dd} with *vcgencmd measure_volts* or with *lm-sensors*. As for core frequency f, BALDER uses the *cpufreq* tool.

$$\Delta V_{th(NBTI)} \leq \int_0^1 A_N u(Vdd) \frac{(v(T) \cdot \delta_C \cdot \delta_e \cdot t_m)^n}{w(\delta_C \cdot \delta_e \cdot T \cdot t)^{2n}} d\delta_e \quad (1)$$

$$v(T) = \xi_4 \cdot \exp\left(-\frac{E_a}{kT}\right); \quad d = C \cdot (V_{dd} - V_{th})^{-r}; \; u(V_{dd}) = (V_{dd} - V_{th}) \cdot \exp\left(\frac{V_{dd} - V_{th}}{E_0}\right)$$

$$w = 1 - \left(1 - \frac{\xi_1 + \sqrt{\xi_3 \cdot v(T) \cdot (1 - \delta(t)) \cdot t_m}}{\xi_2 + \sqrt{v(T) \cdot t}}\right)^{\frac{1}{2n}}; \text{ and } \delta_C = \frac{cyc_{stress}}{cyc_{total}}$$

$$\Delta V_{th(HCI)} = A_H \cdot \sqrt{\alpha_{avg,C}} \cdot u(V_{dd}) \cdot v(T) \cdot \sqrt{\alpha_C \cdot f \cdot t} \quad (2)$$

$$\text{with, } \; u(V_{dd}) = \exp\left(\frac{V_{dd} - V_{th}}{E_1}\right) \text{ and } v(T) = \exp\left(\frac{-E_a}{kT}\right)$$

Fig. 2: Equations used to estimate the total ΔV_{th}.

Algorithm 1 BALDER'S Learning Algorithm

Input: $C \leftarrow \{C_1, C_2, \ldots, C_k\}$: set of threads/cores
$P \leftarrow \{P_1, P_2, \ldots, P_p\}$: set of PPIs
$S \leftarrow \{Input\}$: small input set of A
ω: Initial number of threads
β: Increasing number of threads
$\phi(\omega, p) \leftarrow \infty$: best PPI and number of threads found so far
for each p in P **do**
  $\phi' \leftarrow 0$: maximum measured metric found so far
  $\Omega \leftarrow \omega$ and $M' \leftarrow \mathbb{M}(A, \Omega, p)$
  while $M' \geq \phi'$ **do**
    $\phi' \leftarrow M'$ and $\Omega \leftarrow \Omega + \beta$ and $M' \leftarrow \mathbb{M}(A, \Omega, p)$
  end while
  if $\phi' \geq \phi(\Omega, p)$ **then**
    $\phi(\Omega, p) \leftarrow \phi'$
  end if
end for
return $\phi(\Omega, p)$

(e.g., tradeoff between performance and aging) of executing the parallel application on c distinct cores and p PPIs. An optimized assignment ϕ consists of finding a valid assignment ϕ that leads to maximum value to $\mathbb{M}$. Algorithm 1 receives the set of cores *C*, the set of PPIs *P*, the small input set for an application *A*, and two parameters: ω – the initial number of threads given to application *A*, and β – the increasing factor for the number of threads/processes given to *A* (we consider $\omega = 2$ and $\beta = 2$). The procedure starts selecting a PPI (*p*) from the set of PPIs. Then it executes application *A* parallelized with *p* and exploiting TLP with Ω number of threads/processes. This number increases by a factor β while maximizing the score function $\mathbb{M}(\phi) = M'$. Because finding the ideal number of threads to execute a parallel application can be considered a convex optimization problem, there will be a single specific number of threads/processes per PPI that delivers the best tradeoff between performance and aging.

IV. Methodology

Benchmarks: We consider ten parallel applications already parallelized from [20], which have different communication demands and operations to exchange data, as depicted in Table I. They are classified into two classes: High Communication (HC) and Low Communication (LC) and were executed with a small input set, used for the learning phase, and a regular input set (Table I), used to evaluate BALDER.

Execution Environment: We consider two embedded platforms: (*i*) Raspberry PI, with four ARM Cortex-A53 running at distinct frequency levels (0.6GHz-1.2GHz), and $V_{th} = 0.395$; and (*ii*) Jetson TX-2 machine (4-plus-1 quad-core ARM Cortex-A15 CPU, and $V_{th} = 0.395$). We used the Linux Ubuntu OS, kernel v.4.15, GCC v. 8.3 with the -O3 optimization flag, and the DVFS governor set to ondemand. The following PPIs were used: OpenMP 5.0, PThreads/POSIX.12008, and OpenMPI 3.1.4. Each configuration (algorithm, PPI, TLP) was executed ten times with $\sigma \leq 0.5\%$.

V. Results

Figures 3 and 4 present the results for each benchmark class considering the geometric mean of all applications within each class. They show each PPI running with a different number of threads/processes (1, 2, 3, 4, and 8), the result found by BALDER, and the best result found by the exhaustive search.

TABLE I: Main characteristics of each benchmark

Benchmarks		Operations to exchange data				Input size
		2	3	4	8	
HC	Game of Life (GL)	414	621	1079	1625	4096 x 4096
	Gauss-Seidel (GA)	200004	200006	200008	20016	2048 x 2048
	Gram-Schmidt (GS)	3009277	4604384	6385952	12472634	2048 x 2048
	Jacobi (JA)	4004	6006	8008	16016	2048 x 2048
	Turing Ring (TR)	16000	24000	32000	64000	2048 x 2048
LC	DFT	4	6	8	16	32768
	Dijkstra (DJ)	4	6	8	16	2048 x 2048
	Dot Product (DP)	4	6	8	16	15 billions
	Matrix Mult. (MM)	4	6	8	16	2048 x 2048
	Histograms Simil. (HS)	4	6	8	16	1920 x 1080

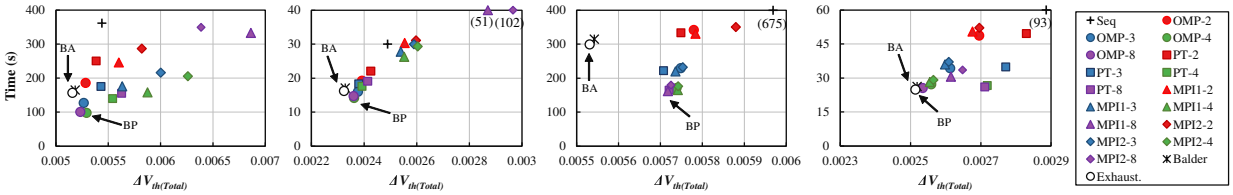
Fig. 3 depicts the $\Delta V_{th(total)}$ (*x-axis*) and the execution time (*y-axis*) for each configuration on each benchmark class and platform (BP means the configuration that delivers the best performance while BA, the best aging), while Fig. 4 shows the number of times that the application set may execute until the processor's end-of-life (*x-axis*) and its respective lifetime (*y-axis*). We consider that the end-of-life of a processor happens when the V_{th} is increased by 10%. As an example, if HC applications are executed with OpenMP (2 threads) on the Rasp. Pi (Fig. 4a.), it will be capable to complete 0.41×10^6 executions before its end-of-life, estimated in 2.44 years.

For HC applications, the characteristics of each PPI play an essential role in the performance and $\Delta V_{th(total)}$ due to the way synchronization and communication are performed (Fig. 3a and 3b). Overall, applications implemented with OpenMP are capable of executing more times until the processors' end-of-life (Fig. 4a and 4b). The difference between OpenMP and PThreads is related to the impact of context switching imposed by the use of mutexes to perform synchronization and communication among threads. Because of the impact of context switching on the duty-cycle of each core, PThreads implementations were slower and presented a higher $\Delta V_{th(total)}$ than the OpenMP. As for MPI implementations, the higher the number of processes, the worst the trade-off between performance and $\Delta V_{th(total)}$ due to excessive amount of send/receive operations. On the other hand, for LC applications (Fig. 3c, 3d, 4c, and 4d), on average, the PPIs have a similar behavior due to the CPU intensive nature of the applications, which amortizes the impact of each PPI on communication and synchronization.

However, even though OpenMP implementations provide the best results on the average of each benchmark class, they are not capable of maximizing the number of times an application may execute until the device's end-of-life for all situations (Table II).

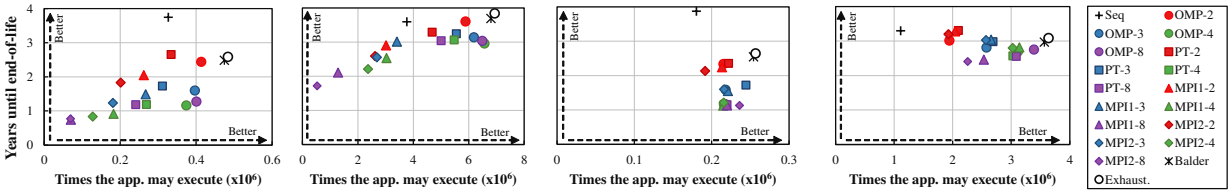
Therefore, by using BALDER to find an ideal or near-ideal combination of PPI-TLP for each application, the number of times that an application may execute until the processor's end-of-life can be significantly increased. As an example, if BALDER is used to find the best combination for HC applications on the Raspberry Pi (Fig. 4a) instead of the configuration that delivers the best result (i.e., all applications running with OMP-2), the applications may be executed 1.14 times more while keeping a similar end-of-life.

BALDER is capable of reaching a result that is very close to the one found by the exhaustive search (Fig. 3 and 4). However, as BALDER employs the learning algorithm over



a) HC applications – Raspberry Pi b) HC applications – Jetson c) LC applications – Raspberry Pi d) LC applications – Jetson

Fig. 3: Execution time and $\Delta V_{th(Total)}$ for each combination of PPI and number of threads (gmean of each bench. class)



a) HC applications – Raspberry Pi b) HC applications – Jetson c) LC applications – Raspberry Pi d) LC applications – Jetson

Fig. 4: The number of executed applications until the end-of-life of each processor per combination of PPI and degree of TLP

TABLE II: Best pair – PPI (#Threads)– and the learning time of Balder and exhaustive search for each benchmark

		Best Configuration		Learning of Balder		Learning of Exh. Search	
		Pi	*Jetson*	*Pi*	*Jetson*	*Pi*	*Jetson*
HC	***GA***	MPI-1(3)	MPI-2(4)	54s	128s	12961s	1742s
	GL	OMP(4)	PT(4)	34s	108s	2403s	395s
	GS	OMP(8)	OMP(2)	87s	25s	3474s	452s
	JA	OMP(2)	OMP(2)	106s	21s	4239s	689s
	TR	OMP(2)	PT(4)	13s	18s	2660s	450s
LC	***DFT***	OMP(3)	MPI-1(8)	51s	16s	3392s	1384s
	DJ	OMP(2)	MPI-1(4)	81s	18s	5967s	550s
	DP	MPI-2(8)	OMP(8)	85s	18s	3181s	941s
	HS	PT(3)	PT(8)	29s	14s	1645s	171s
	MM	PT(2)	PT(4)	22s	20s	13442s	2943s

the application with its smaller input set, the learning time of BALDER is significantly reduced (Table II): only 6.7% and 2.6% of the time spent by the exhaustive search for HC and LC applications on the Jetson platform. Furthermore, there is also an implicit overhead of BALDER regarding (*i*) the database; and (*ii*) the time to access it. BALDER occupies *7.9 Kb* of space and that each hash with the best combination stored adds *136 bytes*. The time to update the database is *0.001s* while the time for searching a combination and read it is *0.002s*.

VI. Conclusion and Future Work

We have presented BALDER, a framework capable of finding the pair of PPI and the number of threads so that the number of times an application may execute until the processor's end-of-life is maximized. BALDER is transparent to both designer and end-user: given different application binaries already compiled, it optimizes the application execution without any code changes. As future work, we will enhance BALDER to consider heterogeneous architectures and to combine with state-of-the-art approaches.

References

[1] S. Kamil, J. Shalf, and E. Strohmaier, "Power efficiency in high performance computing," in *IEEE IPDPS*, April 2008, pp. 1–8.

[2] S. Corbetta and W. Fornaciari, "Nbti mitigation in microprocessor designs," in *ACM Great Lakes Symp. on VLSI (GLSVLSI)*, ser. GLSVLSI '12. New York, NY, USA: ACM, 2012, pp. 33–38.

[3] B. Chapman, G. Jost, and R. v. d. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[4] D. Butenhof, *Programming with POSIX Threads*, ser. Addison-Wesley professional computing series. Addison-Wesley, 1997.

[5] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core*, 2nd ed. Cambridge, MA, USA: MIT Press, 1998.

[6] A. Lorenzon and A. Filho, *Parallel Computing Hits the Power Wall: Principles, Challenges, and a Survey of Solutions*, ser. SpringerBriefs in Computer Science Series. Springer, 2019. [Online]. Available: https://books.google.com.br/books?id=sFy7zQEACAAJ

[7] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta, "Argo: Aging-aware gpgpu register file allocation," in *IEEE/ACM/IFIP CODES+ISSS*. NJ, USA: IEEE Press, 2013, pp. 30:1–30:9.

[8] M. Shafique, M. U. K. Khan, and J. Henkel, "Content-aware low-power configurable aging mitigation for sram memories," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3617–3630, 2016.

[9] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, Duo Sun, and J. Henkel, "Hayat: Harnessing dark silicon and variability for aging deceleration and balancing," in *ACM/EDAC/IEEE DAC*, 2015, pp. 1–6.

[10] V. Hanumaiah and S. Vrudhula, "Temperature-aware dvfs for hard real-time applications on multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, 2012.

[11] H. Khdr, H. Amrouch, and J. Henkel, "Aging-aware boosting," *IEEE Transactions on Computers*, 2018.

[12] A. Rahimi, L. Benini, and R. K. Gupta, "Aging-aware compiler-directed vliw assignment for gpgpu architectures," in *DAC*. NY, USA: ACM, 2013, pp. 16:1–16:6.

[13] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli, "Thermal balancing policy for multiprocessor stream computing platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1870–1882, 2009.

[14] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," *IEEE Trans. on VLSI Systems*, vol. 19, no. 10, pp. 1884–1897, 2011.

[15] H. Lee, M. Shafique, and M. A. Al Faruque, "Aging-aware workload management on embedded gpu under process variation," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 920–933, 2018.

[16] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan, and M. Shafique, "Life guard: A reinforcement learning-based task mapping strategy for performance-centric aging management," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[17] F. Oboril and M. B. Tahoori, "Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, 2012, pp. 1–12.

[18] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the nbti effect for reliable design," in *IEEE Custom Integrated Circuits Conference 2006*, 2006, pp. 189–192.

[19] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, 2014, pp. 478–485.

[20] A. F. Lorenzon, M. C. Cera, and A. C. S. Beck, "Investigating different general-purpose and embedded multicores to achieve optimal trade-offs between performance and energy," *JPDC*, vol. 95, pp. 107–123, 2016.