Run-time Resource Management in CMPs Handling Multiple Aging Mechanisms

Hashem Haghbayan, *Member, IEEE,* Antonio Miele, *Senior Member, IEEE,* Onur Mutlu, *Member, IEEE,* Juha Plosila, *Member, IEEE,*

Abstract—Run-time resource management is fundamental for efficient execution of workloads on Chip Multiprocessors. Application- and system-level requirements (e.g. on performance vs. power vs. lifetime reliability) are generally conflicting each other, and any decision on resource assignment, such as core allocation or frequency tuning, may positively affect some of them while penalizing some others. Resource assignment decisions can be perceived in few instants of time on performance and power consumption, but not on lifetime reliability. In fact, this latter changes very slowly based on the accumulation of effects of various decisions over a long time horizon. Moreover, aging mechanisms are various and have different causes; most of them, such as Electromigration (EM), are subject to temperature levels, while Thermal Cycling (TC) is caused mainly by temperature variations (both amplitude and frequency). Mitigating only EM may negatively affect TC and vice versa. We propose a resource orchestration strategy to balance the performance and power consumption constraints in the short-term and EM and TC aging in the long-term. Experimental results show that the proposed approach improves the average Mean Time To Failure at least by 17% and 20% w.r.t. EM and TC, respectively, while providing same performance level of the nominal counterpart and guaranteeing the power budget.

Index Terms—Lifetime Reliability, Many-core Architectures, Mapping, Run-time Resource Management

1 Introduction

The fast technology scaling we have witnessed during the last decades has led to integration of from few tens to several hundreds of cores on a single Chip Multiprocessor (CMP). CMPs are currently the backbone in computing systems for high-performance execution of multi-programmed workloads. However, the end of Dennard scaling has implied that the supply voltage has not followed the same exponential scaling experienced in transistor miniaturization, leading to *dark silicon* [1]. In particular, physical limits in device packaging and cooling technology put a constraint on peak power consumption and peak power density, making it impossible to power on an entire chip at a nominal voltage/frequency (VF) level at the same time. From a practical point of view, the Thermal Design Power (TDP) is generally defined in a conservative way by designers to avoid excessive heating potentially damaging transistor junctions. Thus, to satisfy the TDP, only a fraction of the available on-chip cores can be operated at a nominal VF level at any given moment. Alternatively, a larger portion of cores can be operated at a reduced VF level. International Technology Roadmap for Semiconductors (ITRS) Projections in 2013 [2] have shown that the percentage of dark silicon for a chip in 22nm technology is around 50% while at 8nm it will increase to 70%. As a consequence, on-chip power management is becoming an increasingly relevant issue.

TDP control ensures avoidance of chip failures due to extreme power densities; similarly, temperature control ensures avoidance of thermal peaks. Sometimes, TDP is considered too conservative, and more advanced power management strategies, such as Thermal Safe Power (TSP) [3], have been defined to achieve higher performance. However, all these approaches cannot handle the gradual effects of high temperatures experienced by the chip over time. As reported by the ITRS in 2011 [2], long-term high temperature profiles, even if within design guard bands, lead to an acceleration of aging and wear-out process of the chips manufactured in very small technology nodes. Thus, phenomena such as Electromigration (EM), Negative-Bias Temperature Instability (NBTI) or Thermal Cycling (TC), cause delay errors and, eventually, device breakdowns [4]. Moreover, it has been shown how failure mechanisms are exponentially dependent on the temperature [5]; a $10 - 15°C$ difference in the operating temperature may result in a $2\times$ difference in the device lifetime.

Run-time resource management plays a key role in the efficient execution in CMPs since they use to experience variable workloads composed of several multi-threaded applications entering and leaving the system with an unknown trend. Dynamic Reliability Management (DRM) has been widely investigated to enhance run-time resource management to handle also aging issues (e.g., [6], [7], [8], [9], [10]). Indeed, properly controlling the activity of a CMP executing a variable workload allows at the same time to co-manage performance and power consumption in the short term, and lifetime reliability in the long term. The feedback loop in DRM is exploited also to monitor the aging status of the single cores to take decision on the workload distribution and architecture tuning by acting on various knobs at application level, such as the mapping and scheduling, and at architecture level, such as Dynamic Voltage/Frequency Scaling (DVFS) and Per-Core Power Gating (PCPG). However, the largest part of the literature addresses aging issues only with a partial view. This is particularly due to the fact that there are two different classes of aging mechanisms: i) the ones that are mainly subject to the

• *Hashem Haghbayanand and Juha Plosila are with the Autonomous Systems Laboratory, Department of Computing, University of Turku, Turku, Finland. Antonio Miele is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy.Onur Mutlu is with ETH Zürich, 8092, Zürich, Switzerland.*
E-mail:mohhag@utu.fi, antonio.miele@polimi.it, omutlu@gmail.com, juplos@utu.fi

temperature levels, such as EM or NBTI, and ii) TC, that depends also on the amplitude and the frequency of the temperature variations [4]. Thus, the literature proposes approaches acting only on one of the two classes (e.g., [6], [7], [8], [9], [10] vs. [11], [12]) and, most of the time, strategies that are beneficial for first class of aging mechanisms have negative effects on the other one and vice versa.

Given these motivations, we propose a novel comprehensive lifetime-reliability-aware run-time resource management approach for CMPs addressing multiple aging mechanisms, i.e., EM and TC. The approach aims at optimizing the system performance (in terms of completed applications per unit of time) while guaranteeing the given power budget (specified as TDP or TDP), and it is based on state-of-the-art strategies for mapping, scheduling and Dynamic Power Management (DPM), that have been extended to take into account also the aging status of the cores in the decision process in order to prolong the system lifetime. In more detail, the main contributions can be expressed as follows:

- Proposing a new run-time framework capable of observing the short-term power and performance status of a CMP as well as analyzing its long-term aging history. It facilitates a unified control and organization of resources based on current and predicted future of the systems status.
- Proposing a multi-objective run-time resource management approach to simultaneously control short-term effects, avoiding violation of the power budget and maximizing system performance; and long-term effects, keeping EM and TC under control to maximize system lifetime reliability.
- Evaluating the proposed resource management approach in various working scenarios against state-of-the-art work and demonstrating an improvement of lifetime about 17% and 20% w.r.t. the EM and TC, respectively, while maintaining similar performance and power consumption profile.

The paper is organized as follows. Sections 2, 3 and 4 discuss the preliminaries, a motivating example, and related work, respectively. Section 5 describes in details the proposed reliability-aware resource management approach. Section 6 presents the experimental evaluation and Section 7 draws conclusions.

2 Preliminaries

2.1 System Architecture

This work targets Chip Multiprocessors (CMPs) having an architecture similar to the one depicted in Figure 1; it is the classical multi-core system organized in a mesh grid of homogeneous cores interconnected by means of a Network-on-Chip (NoC) (as in several past works, e.g., [13], [10], [14], [9]). The CMP has a memory hierarchy comprising a unified off-chip main memory accessible through a set of Memory Controllers (MCs) placed on the sides of the chip and accessible from the routers at the corners of the NoC, a shared L2 cache distributed all over the NoC, and private per-core L1 caches. Each core provides individual HW knobs for DVFS and PCPG, and sensors for temperature and power measures [10], [9].

The CMP loads an Operating System (OS) acting as a controller which monitors the state of the system, coordinates the application execution and tunes the HW knobs (as in [1], [9]). Considered applications are multi-threaded and data-intensive; the workload is composed of several applications entering and leaving the system with an unknown trend. Applications are dynamically distributed and managed by three cooperating OS components:

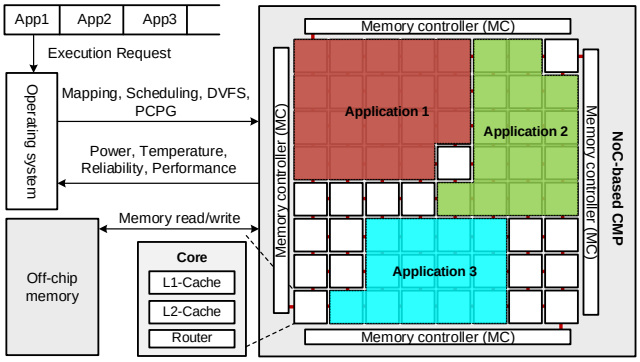


Fig. 1. The target system architecture.

- *Mapping unit* which identifies a set of idle cores in the architecture to execute each incoming applications, enqueued in an execution request list.
- *Scheduling unit* which schedules the execution of the threads belonging to each running application on the allocated cores.
- *Dynamic Power Management (DPM) unit* which controls the overall power consumption based on a given budget.

Baseline solution. Since this work focuses on managing threats due to multiple aging mechanisms, the approach we present here is an extension of a full-fledged nominal run-time resource management solution, obtained by integrating various state-of-the-art strategies. In particular, the mapping has been divided in a region selection step, automated by MapPro algorithm [15] and a thread mapping one, automated by CoNA algorithm [16]; MapPro also exploits the memory affinity metric proposed in [13] to distribute applications based on the memory traffic. The scheduling has been implemented by means of the standard single-queue multiprocessor scheduling algorithm using the Round-Robin policy [17], and the DPM approach by means of the approach in [18]. The three selected units cooperate to maximize performance, in terms of completed applications per unit of time while not violating the given power budget. In particular, the mapping unit tries to maximize the number of concurrently running applications on the cores' grid and to give the largest possible number of cores to run concurrently threads of each application. Then, since the mapping policy partitions the cores among applications, the popular Round-Robin scheduling policy allows all application's threads to advance at the same time, thus minimizing application execution time. Finally, the DPM unit tunes VF of cores assigned to each application to run it as fast as possible without violating the power budget. The goal of this work is to enhance this scheme to consider also lifetime reliability.

2.2 Reliability Model

Aging sensors are not generally available in commercial chips; thus, a common practice in DRM (e.g. [19], [9], [12]) is to adopt the classical stochastic reliability model based on the Weibull distribution [4]. The reliability of a single core acting at a reference worst-case temperature T is estimated as:

$$R(t) = e^{-\left(\frac{t}{\alpha(T)}\right)^{\beta}} \quad (1)$$

where t is the current time (in hours), β the Weibull slope parameter, and $\alpha(T)$ the scale parameter, or *aging rate*, depending on the considered aging mechanism and to the fixed temperature

T. The reliability model is extended, as discussed in [5], to consider temperature variations as follows:

$$R(t) = e^{-\left(\sum_{j=1}^{i} \frac{\tau_j}{\alpha_j(T)}\right)^{\beta}} \quad (2)$$

where τ_j represents the duration of each time period with constant steady-state temperature T_j in the core up to time t (i.e., $t = \sum_{j=1}^{i} \tau_i$). Moreover, when interested in analyzing the average aging trend, Equation 2 is also simplified by computing an average aging rate by tracing the system execution for a representative time period and by using the following formula:

$$\alpha_{avg} = \frac{\sum_{i=0}^{p} \tau_i}{\sum_{i=0}^{p} \frac{\tau_i}{\alpha_i(T)}} \quad (3)$$

where τ_i represents the duration of the p steps within the considered period. Thus, α_{avg} can be applied in both Equations 1 and 2. Finally, the overall expected lifetime of the core is computed in terms of Mean Time To Failure (MTTF):

$$MTTF = \int_0^{\infty} R(t)dt. \quad (4)$$

When the system works in steady-state conditions for the entire operating life, as we will consider in the experimental sessions of this work, MTTF can be also computed as:

$$MTTF = \alpha_{avg} \cdot \Gamma\left(1 + \frac{1}{\beta}\right) \quad (5)$$

where Γ is the statistical gamma function.

In this paper we consider two different aging mechanisms, that are Electromigration (EM) and Thermal Cycling (TC); each one of them is characterized by a specific formula for the aging rate $\alpha(T)$ parameter to be used in the above lifetime reliability model, described in the following.

The EM aging rate is derived from Black's equation [4] as:

$$\alpha_{EM} = \frac{A_{EM}(J - J_{\text{crit}})^{-n} e^{\frac{E_a}{kT}}}{\Gamma\left(1 + \frac{1}{\beta}\right)} \quad (6)$$

where A_{EM} is a process-dependent constant, J and J_{crit} the current density and the critical value activating EM, E_a the constant EM activation energy, k the Boltzmann's constant, and n a material-dependent constant. Almost all the other aging mechanisms, but TC, have the α based on a formula similar to the Black's equation.

TC aging rate depends on both temperate peak T_{Max} and the amplitude of the thermal cycle δT as stated by the Coffin-Mason's equation [4], [5], which estimates the overall lifetime in terms of expected number of cycles:

$$N_{TC} = A_{TC}\left(\delta T - T_{th}\right)^{(-b)} e^{\frac{E_{a_{TC}}}{kT_{Max}}} \quad (7)$$

where A_{TC} is an empirically determined fitting constant, T_{th} is the temperature where the inelastic deformation begins, b is the Coffin-Mason exponent constant, $E_{a_{TC}}$ is the activation energy for TC. From Equation 7, the TC aging rate to be used in Equation 1 is derived as follows:

$$\alpha_{TC} = \frac{N_{TC}\Delta t}{\Gamma\left(1 + \frac{1}{\beta}\right)} \quad (8)$$

where Δt is the duration of a single cycle. It is worth noting that the slope parameter β may assume different values for the two aging mechanisms.

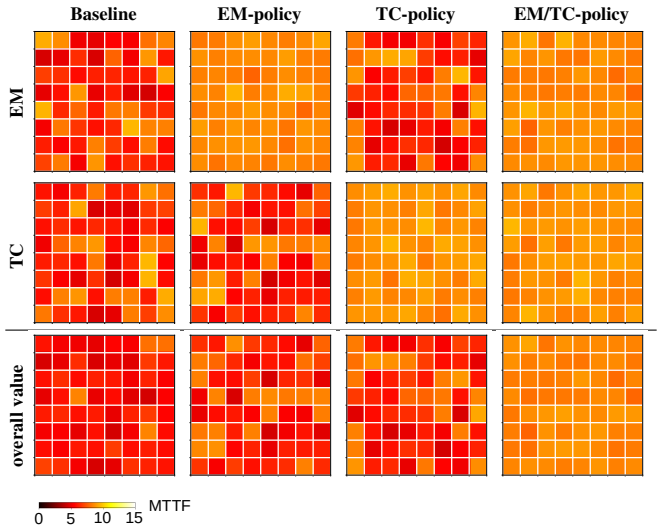


Fig. 2. MTTF of the cores in the CMP w.r.t. EM, TC or both of them (i.e. overall value, computed per each core as the minimum of the two precedent values) for the four considered approaches: baseline, EM-policy, TC-policy, EM/TC-policy.

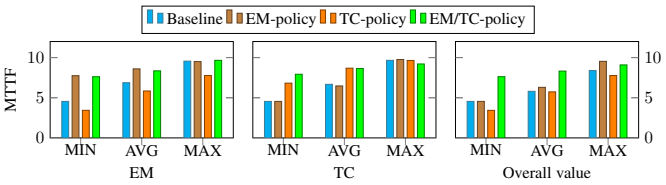


Fig. 3. Statistical analysis on results in Fig. 2.

The Coffin-Mason equation assumes, as Black's one, an ideal steady state situation. To consider a more realistic situation comprising various thermal cycles having different amplitudes and maximum temperature, Equation 2 can be applied on the sequence of occurring cycles. Such cycles can be extracted at run-time by means of the rainflow count algorithm [20]. Finally, in the case the system experiences a steady state situation with a list of cycles periodically occurring, N_i of the various m identified cycles can be aggregated in an average value by means of the Miner's rule [5]:

$$N_{TC\_avg} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_i}} \quad (9)$$

In such a situation, Δt in Equation 8 should represent the average duration of the various cycles.

3 Motivating Example

Let us consider a system architecture composed of 8×8 cores, as the one in Figure 1, executing a variable workload[1]. We performed four runs with the same workload and considering different sets of alternative policies for the same run-time resource management approach, with the aim at investigating the long-term effects on EM and TC. In particular, the first considered approach is the defined baseline (described in Section 2.1), representing the basic reliability-unaware approach. Such an approach has been enhanced by either introducing EM-awareness or TC-awareness. Both the two reliability-aware approaches act on the various knobs in mapping, scheduling and DPM to balance performance and the

1. Details on the system configuration are provided in Section 6.

considered aging mechanism. In particular, in each of the three phases, decisions are taken by prioritizing cores with an higher reliability w.r.t. one of the two aging mechanisms (i.e., cores are sorted w.r.t. their R_{EM} or R_{TC} value). Finally, the last analyzed approach features both EM-awareness and TC-awareness, in an orchestrated way as discussed in this paper.

Figure 2 reports the results of such preliminary experiments. Each plot represents with colors the MTTF of each core in the CMP under specific working conditions. In particular, each row of plots refers to an aging mechanism, i.e., EM, TC and both of them (computed as the minimum of the two separate values as in [9]); while each column of plots represents results for one of the considered approaches, i.e., baseline, EM-policy, TC-policy and EM/TC-policy. As expected, the reliability-unaware baseline, which acts on mapping and scheduling of threads, DVFS, and PCPG to trade off between performance and power consumption in the short term, has a deleterious effect on aging in the long term. This demonstrates that the DPM policy controlling the proper distribution of the power budget by means of DVFS avoids temperature peaks but it does not suffice in maintaining a low temperature profile and reducing temperature fluctuations.

When considering the reliability-aware policies (two central columns), we can notice that each policy is beneficial only for the aging mechanism it is aware of but not for the other one. Indeed, EM and TC are caused by different aspects, temperature levels and variations respectively, that are conflicting in the reliability-aware policies. The classical EM-policy reduces the overall temperature levels by neglecting how many temperature variations it may cause, and, conversely, the TC-policy minimizes the number of temperature variations without any attention to the average temperature experienced by the various cores. The final effect, reported in the last row, is that both the two approaches offers very poor results in terms of MTTF values. We may conclude that to be effective on both EM and TC, it is necessary to act in the long term on both the two causes of the aging in synergy. As shown in the last column of plots, an EM/TC-aware policy, as the one proposed in this paper, succeeds in this purpose.

To better analyze these results, Figure 3 reports the minimum (MIN), maximum (MAX), and average (AVG) reliability values. We can notice that joint management of EM and TC allows to increase both minimum and average reliability for the two aging mechanisms at least of 19% and 22% w.r.t. the baseline respectively. The minimum reliability value represents the worst case situation, i.e., the most aged core, while the average value shows the mean status of the overall architecture. At the same time, standard deviation in reliability is minimized, indicating that the joint management avoids unbalanced aging in all regions of the system. We may notice that EM-policy and TC-policy provides merely better average results for their specified reliability type, but at the same time this has a considerably negative effect on the other type of reliability not targeted in the policy; thus, the overall reliability values for both policies are worse than the EM/TC-policy, that is capable at facing with both the two aging mechanisms. In conclusion, we believe that resource management for CMPs needs an efficient multi-objective feedback-based dynamic approach focusing on the both aging mechanisms at the same time of power consumption and performance.

4 Related Work

DRM has been first proposed in [21], [22] to deal with aging in single-core architectures by means of a proper DVFS tuning. These first works address several aging mechanisms, such as EM, NBTI and TC, but they consider a simplistic reliability model based on an exponential failure distribution to ease MTTF computation; this model is not realistic for modern systems. Moreover, the single-core architecture does not represent modern multi-core architecture and DVFS is only a single knob among various available ones in the overall system.

Subsequent approaches (e.g., [23], [24]) target shared-memory multi-core systems and actuate on DVFS and task scheduling. Simplistic reliability models not including TC are still adopted. Moreover, these studies consider single-threaded applications; more complex applications composed of multiple threads or a task-graph of pipelined tasks, executed by modern multi-core systems, cannot be managed by these schemes. Further works [25], [26] extend to multiple parallel applications and more accurate aging models, while other ones [27], [28] consider heterogeneous architectures. However, No one of these approaches defines a proper reliability-aware mapping strategy, that is highly relevant in aging management as the number of architectural cores increases.

Later contributions (e.g., [6], [7], [19], [9]) propose reliability-driven run-time workload distribution and resource management in CMPs to find a trade off between performance and lifetime reliability. The central element of all these proposals is the mapping strategy; in fact, to achieve high application performance and prolong the system lifetime it is fundamental to identify the group of processing cores among the large availability of resources to be used to execute the various threads/tasks of each application in an optimal way. More precisely, the topology of the selected cores has a relevant effect both on the performance, due to the intensive communications required by the several tasks/thread in the same application and on the overall thermal stress on the various cores due to the heating exchange among neighbour elements. The approaches in [6], [7] periodically migrate applications' tasks from elder cores to younger ones; aging is computed according accurate models for EM or NBTI. The mapping algorithm in [8] performs an almost-exhaustive solution space exploration to balance aging. The technique in [29] defines a zoning strategy to select the younger area of the resource grid to deploy the arrived applications. The approach presented in [30] uses aging status prediction to drive possible thread mapping decisions. The technique is specifically tailored for NBTI. In [19], machine learning is used for similar purposes in the management policy. All the previous approaches address only partially the overall run-time resource management problem, in particular frequently neglecting power management. In this perspective, [9], [10] present two comprehensive approaches for mapping incoming application task-graphs and actuating on DVFS and PCPG to optimize performance while guaranteeing both the power budget and the given lifespan. Similarly to all previous approaches, they only consider EM. In conclusion, none of the discussed methods addresses TC nor can be easily extended or adapted in that direction.

Apart [21], few other works address also TC in multi-core systems. A machine learning strategy is used in [31] to select the proper DVFS governor to trade performance and aging; the approach is not effective on CMP with a large number of cores since providing a very limited control on DVFS. The run-time mapping policy proposed in [32] addresses all aging mechanisms for single-threaded applications. The idea is to use the youngest core w.r.t. the various aging models without any DVFS tuning. As shown in [12], this strategy is not efficient for TC; in fact, it requires a more direct control of the amplitude and the frequency of the tem-

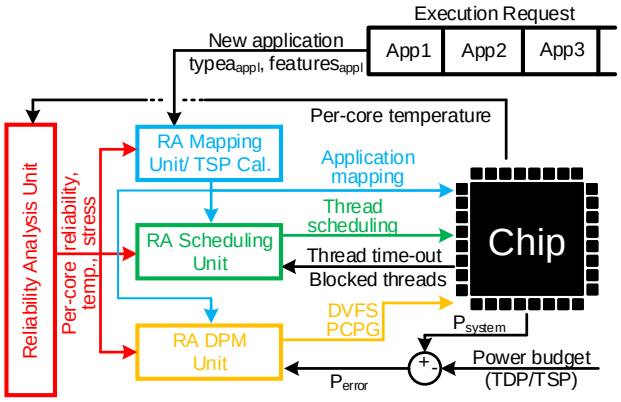


Fig. 4. The run-time reliability-aware resource manager.

perature variations. Furthermore, the simplistic mapping method is not suitable for multi-threaded applications. The approach in [11] considers in principle the same base idea as the previous one and therefore shares its limitations. In particular, the management aims at minimizing the variance in resource utilization so that temperature levels and variations are indirectly minimized. The approach in [33] focuses on heterogeneous architectures, integrating both CPU and GPU. Indeed, the mapping strategy for single-threaded applications is very similar to the previous ones. The novelty is a wear balancer which redistributes the applications based on the aging status of the various heterogeneous cores; to limit the number of thermal cycles, thus not incurring in the same weakness of [32], the balancer avoids to perform relocations between cores with a high temperature difference. Indeed, this is an interesting idea, even if a bit immature w.r.t. the approach we here propose. A more advanced approach addressing TC in a systematic way in the scenario of CMPs running a variable workload of multi-threaded applications has been proposed in [12]. The approach features policies for mapping applications on the grid of cores, scheduling applications' thread on the allocated cores and tuning DVFS to trade performance and excessive heating withing the available power budget. On the other hand, the work considers TC but not EM.

As a conclusion, to the best of our knowledge, no work in the literature addresses entirely the aging management of both EM (or any other temperature-level-based aging mechanism) and TC in the complex scenario of CMPs running multi-threaded applications. Our aim is therefore to restart from our previous experiences in reliability-aware run-time resource management against EM [9] and TC [12], that are incomplete proposals w.r.t. the overall discussed scenario and cannot be extended in that direction, in order to design a completely-new comprehensive and full-fledged approach for both EM and TC.

5 Proposed Controller Architecture

Figure 4 presents the overall architecture of the proposed lifetime-reliability-aware run-time resource manager. Such a controller integrated in the OS running on top of the CMP implements a feedback loop with the system to monitor and control workload execution and architecture status. As discussed in the preliminaries, the controller is internally organized into three basic modules: i) Mapping unit, ii) Scheduling unit, and iii) DPM unit. In addition,

Algorithm 1 Reliability Analysis of a single core

Inputs:
- t_i: overall system operational life
- $\mathbf{T}$: core temperature trace (a vector) for current control period t_i

Outputs:
- R_{EM} : reliability w.r.t. EM at time t_i
- R_{TC} : reliability w.r.t. TC at time t_i
- $status_{EM}$: core status w.r.t. EM at the present control period t_i
- $status_{TC}$: core status w.r.t. TC at the present control period t_i

Internal registers:
- A_{EM}: accumulated EM aging rate
- A_{TC}: accumulated TC aging rate

(both registers are initialized to 0 at the beginning of the system operating life)

Constants:
- Δt: duration of the long-term control period

Body:

1: $\mathbf{T_{low}} \leftarrow$ low_pass_filter($\mathbf{T}$);
2: $\alpha_{EM\_avg} \leftarrow$ compute_α_{EM_avg}($\mathbf{T_{low}}$);
3: $A_{EM} = A_{EM} + \frac{\Delta t}{\alpha_{EM\_avg}}$
4: $R_{EM} = e^{-\left(\frac{t_i}{A_{EM}}\right)^{\beta}}$
5: $\mathbf{cycles} \leftarrow$ rainflow_count($\mathbf{T_{low}}$)
6: $\mathbf{N_i} \leftarrow$ coffin_masson($\mathbf{cycles}$)
7: $N_{TC\_avg} \leftarrow$ miner_rule($\mathbf{N_i}$)
8: $\alpha_{TC\_avg} \leftarrow \frac{N_{TC\_avg} \cdot (\Delta t / |\mathbf{cycles}|)}{\Gamma\left(1+\frac{1}{\beta}\right)}$
9: $A_{TC} = A_{TC} + \frac{\Delta t}{\alpha_{TC\_avg}}$
10: $R_{TC} = e^{-\left(\frac{t_i}{A_{TC}}\right)^{\beta}}$
11: $status_{EM} =$ classify(R_{EM})
12: $status_{TC} =$ classify(R_{TC})

we here introduce a new *Reliability Analysis unit*, estimating the aging status of the various cores based on continuous temperature measures and the stochastic lifetime reliability model. The Reliability Analysis unit provides to each one of the three above modules the aging status of the various cores. The receiving modules feature state-of-the-art policies, enhanced to consider also reliability in decision making together with performance and power consumption; for these reasons they are referred to as *Reliability-aware* units (RA in the figure).

5.1 Reliability Analysis Unit

The Reliability Analysis unit estimates the reliability status of each core in the architecture at the current time w.r.t. the two considered aging mechanisms, referred to as R_{EM} and R_{TC}. The module receives in input the temperatures, measured on each core by the available temperature sensors at a fixed sampling frequency[2] (about 2 Hz), and collects them into a temperature buffer.

At each long-term control period Δt (about 1-2 hours), the temperature trace collected for each core is processed to update the reliability values at the current time, R_{EM} and R_{TC} according to Algorithm 1. First, a pre-processing of the temperature trace of each core is carried out by means of a low-pass filter, implemented with a moving average (Line 1); the aim is to remove high-frequency small-amplitude oscillations that generally characterizes temperature traces. Then, the computation of the updated R_{EM} values is executed based on Equation 2 (Lines 2–4); in particular, in such an equation the sum in the exponent is unrolled to separate the aging rate affecting the system in the last control period from the previous aging history. In detail, the average aging rate α_{EM_avg} characterizing the current control period is

2. Control periods should be tuned w.r.t. the specific working scenario. Values reported in this paper have been experimentally defined with the setup described in Section 6.

computed by means of Equation 3 (Line 2) and it is summed up to the accumulation of the average aging rates of the entire previous history of the system saved in A_{EM}, i.e., an internal register of the unit (Line 3). In such a way, we obtain the value of the exponent in Equation 2 referred to the current instant of time and, consequently, the updated R_{EM} is derived by means of the classical formula of the Weibull distribution (Line 4).

Regarding to TC (Lines 5–10), the pre-processed temperature trace is first given in input to the rainflow count algorithm [20] which extracts the list of cycles in the last long-term control period; each cycle is a pair of peak and valley temperature values. Then, Equations 8 and 9 are used to compute the average TC aging rate for the current control period, α_{TC_avg}. In this computation we assume all cycles to have the same duration; therefore, at Line 8 the duration of each single cycle is approximated by the length of the control period divided by the number of identified cycles. Finally, A_{TC} is updated similarly to EM counterpart and R_{TC} is computed.

After the update of the reliability values, the module analyzes the stress status of the core (Lines 11–12); in particular, it computes the stress factor of each core w.r.t. EM, defined as $s_{EM} = 1/R_{EM}$, and the average stress of all the cores in the architecture, i.e., s_{EM_avg}. Thus, cores are classified according to the stress experience as:

$$status_{EM} = \begin{cases} normal & \text{if } s_{EM} \leq s_{avg} \\ aged & \text{if } s_{avg} < s_{EM} \leq s_{avg} + s_{th} \\ critical & \text{if } s_{EM} > s_{avg} + s_{th} \end{cases} \quad (10)$$

being s_{th} a predefined threshold. Same computation is performed for TC.

Once all computations are carried out, the temperature buffers are emptied to proceed with the subsequent control period. A_{EM} and A_{TC}, that are initialized to 0 at the beginning of the system's operational life, are saved within the unit so that can be used as previous values in the subsequent period.

5.2 Reliability-aware Mapping Unit

The Reliability-aware Mapping unit identifies a *region* of cores that are proximal in the grid to be used for the execution of a new application. The activity is performed whenever there is a new application waiting in the request queue and some idle core is available. Otherwise, it is postponed until some running application terminates and releases occupied cores, making them available for an application waiting in the execution request list.

The mapping algorithm is based on the following principles:

- Reduce EM and TC stress of the region and neighborhood of the region in the floorplan.
- Concentrate the application threads in the closest region, according to the network base, to optimize communication overheads inside the network.
- Map memory-intensive applications near to the MCs, while other ones may be located farther.

The mapping algorithm, shown in Algorithm 2, takes in input the first application in the execution request queue and the architecture reliability status. It is divided in two steps: i) region selection, and ii) cores reservation. The first step (Lines 1–18) has been inspired from the MapPro algorithm [15], re-adapted to use a new set of affinity metrics. The algorithm ranks all the candidate core regions based on three affinity metrics and selects the most suitable one where to map the application. A region on the cores grid is

Algorithm 2 Reliability-aware Mapping

Inputs:
- $appl$: application to be mapped
- $type_{appl}$: pre-profiled application type w.r.t. memory accesses
- $features_{appl}$: pre-profiled application features
- max_DoP: maximum degree of parallelism for application execution
- $\mathbf{R_{EM}}$, $\mathbf{R_{TC}}$: vectors of cores reliability
- $\mathbf{status_{EM}}$, $\mathbf{status_{TC}}$: vectors of cores status

Body:

```
1: #cores ← max_DoP
2: repeat
3:     r ← ⌈(√#cores − 1) /2⌉
4:     aff_max ← −∞
5:     c_map ← None
6:     for c ∈ arch do
7:         reg ← get_idle_cores(c, r)
8:         reg ← remove_critical_cores(reg, status_EM, status_TC)
9:         if |reg| ≥ #cores then
10:            SF ← compute_SF(reg, R_EM, R_TC, features_appl)
11:            VRF ← compute_VRF(c, r, SF)
12:            MF ← compute_MF(type_appl, c)
13:            aff ← VRF/MF
14:            if aff > aff_max then
15:                aff_max ← aff
16:                c_map ← c
17:    #cores ← #cores − 1
18: until c_map = None and #core ≥ 1
19: if c_map ≠ None then
20:    map(appl, c_map, r)
```

defined with a squared shape; it is identified by the central core c and has a radius r. The algorithm computes the radius r of the squared region, based on the number of concurrent threads spawn by the application (Line 3); in the algorithm we refer to the number of threads as $\#cores$ since we desire to execute each thread on a separate core to maximize application performance. The application comes with the pre-profiled information about the maximum degree of parallelism, max_DoP, that is the maximum number of threads that can be spawned concurrently to parallelize application execution (this value is determined by means of a scalability analysis as the one in [34]). Thus, the region selection algorithm starts the research by setting $\#cores$ equal to max_DoP (Line 1) and decreases the value by 1 at each iteration until a region is not found and $\#cores$ is positive (Lines 17–18); in this way, the algorithm finds the largest region where to accommodate the concurrent threads of the application, thus achieving the largest level of parallelism for the application based on the currently available processing resources. It is worth noting that, if necessary, it is possible to specify also a minimum level of parallelism, min_DoP, to be used as a lower bound for $\#cores$, thus stopping the research earlier with a failure.

Then, the algorithm scans all the cores c in the architecture to analyze the corresponding region with radius r; in particular, the region is candidate only if the number of idle non-*critical* cores is enough to accommodate the application (Lines 7–9). For each candidate region, the algorithm computes the affinity metrics (Lines 10–13). According to the above principles, the adopted affinity metrics are: i) Stress Factor (SF), ii) Vacancy Reliability Factor (VRF), and iii) Memory Factor (MF). First two ones are new metrics here proposed, which computation is detailed in the next subsections, while the latter is the state-of-the-art metric introduced in [13] and here integrated in the selected baseline solution. SF is a new metric here proposed estimating the impact on reliability of each core caused by the mapping of the application on the candidate region; it exploits the pre-profiled

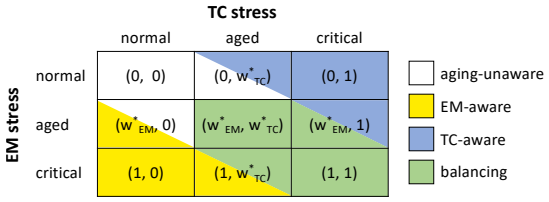


Fig. 5. Look-up table to compute w_{EM} and w_{TC} weights.

application features to predict the stress the new application will cause. VRF is an extension of the Vacancy Factor [15] aimed at measuring the dispersion of the idle cores in the candidate region; it incorporates SF of the various cores to balance dispersion with the core aging. Finally, MF measures the affinity of the region to the application based on the its type, that may be memory intensive, normal or non-memory intensive [13]; the application type is a pre-characterization received by the mapping algorithm as input parameter. Such metrics are then combined in the overall affinity aff. The region associated with core c presenting the maximum aff value, if any, is selected (Lines 13–16).

In the second step (Line 19–20), the module performs the actual reservation of $\#cores$ cores within the candidate region, if one has been identified. The mapping works in an exclusive way (i.e., at most one application can run on a single core); thus, cores already assigned to other applications are not considered. Similarly, *critical* cores are discarded not to negatively affect reliability. This activity is performed by means of the CoNA algorithm [16] aimed at minimizing network congestion caused by the internal communication among the threads; the algorithm has been enhanced to use SF to sort cores.

Stress Factor (SF) Computation. SF is a per-core metric estimating the impact on the aging of each single core that would be caused by the execution of the newly incoming application. SF is computed by assuming that a specific mapping solution is applied for the new application in the current working context of the system and, consequently, by estimating the steady-state temperature the core would experience. To take into account both the considered aging mechanisms, the metric for a single core is defined as the weighed sum of the reliability variations w.r.t. EM and TC assuming the new working conditions will last for a specific time period:

$$SF = w_{EM} \cdot \Delta R_{EM} + w_{TC} \cdot \Delta R_{TC} \tag{11}$$

In the formula, ΔR_{EM} and ΔR_{TC} represent the reliability variations w.r.t. the single aging mechanism caused by the new temperature T' applied for expected execution time of the new application Δt_{appl}. w_{EM} and w_{TC} are the two weight parameters, that are dynamically computed as discussed in the next.

The formula to compute ΔR_{EM} is:

$$\Delta R_{EM} = R_{EM} - e^{-\left(\frac{t_i}{A_{EM}} + \frac{\Delta t_{appl}}{\alpha_{EM}(T')}\right)^{\beta}} \tag{12}$$

where R_{EM} and A_{EM} are the current reliability and accumulated aging rate returned by the Reliability Analysis unit. Moreover the second operand in the subtraction represents the new reliability value caused by the new temperature T'; such a formula is derived from Equation 2. A run-time estimation model has been adopted to predict the new temperature T' caused by the new application w.r.t. to a selected mapping solution. In particular, we adopted a state-of-the-art solution [35] based on a multi-layer perception (MLP) and predicting the new temperature from the current measured temperature of the various cores in the system and a list of extracted application features, such as the number of integer/floating point instructions, the number of L1/L2 cache accesses and misses, etc. Such features and the average application execution time Δt are assumed to be known in advance, for instance, due to online profiling performed during the previous runs of the same application; indeed, such data are input parameters for Algorithm 2. It is worth noting that the temperature estimator is an utility module plugged in the proposed approach; the adopted state-of-the-art approach [35] can be replaced by any newer approach providing a fast steady-state temperature estimation at the granularity of the single core.

The weight parameters w_{EM} and w_{TC} are computed at run-time to adapt mapping decisions to the current aging conditions by means of the look-up table shown in Figure 5 where the two aging status parameters of each core, $status_{EM}$ and $status_{TC}$ (previously defined with Equation 10) are used as input. w_{EM} (and w_{TC}) of a single core may be set to 0, w^*_{EM} (and w^*_{TC}) or 1 depending on the current core status, being normal, aged or critical, respectively, where w^*_{EM} (and w^*_{TC}) represents a value between 0 and 1 defined at design time, such that $w^*_{EM} + w^*_{TC} = 1$. This aspect has two main consequences:

- It allows to dynamically modify the relevance of an aging mechanism w.r.t. to the other one in Equation 11 based on the predicted absolute reliability values (ΔR_{EM} and ΔR_{TC}), thus taking into account only the actual aging issues. For instance, when $w_{EM} = 1$ and $w_{TC} = 0$, SF is computed only based on EM.
- It is possible to dynamically modify the relevance of SF in the metrics aggregation (Line 13 of Algorithm 2). For instance, when a core is in a normal status w.r.t. both the aging mechanisms ($w_{EM} = 0$ and $w_{TC} = 0$), VRF formula becomes equal to the state-of-the-art Vacancy Factor, which considers only the cores distribution; thus the overall affinity metric will be computed only based on cores distribution and memory affinity. At the opposite, if $w_{EM} = 1$ and $w_{TC} = 1$, due to the critical stress experienced by the core, SF will be predominant on MF and the cores distribution.

Vacancy Reliability Factor (VRF) Computation. This metric is inspired from the Vacancy Factor proposed in [15], aimed at estimating the dispersion of the idle cores in a square region; the lower the dispersion, the better the region. Such a metric is here extended to incorporate SF to balance the cores dispersion with the aging of the various involved cores.

Given the central node c with coordinates w, h identifying a square region with radius r, VRF is defined as:

$$VRF_c = \sum_{i=w-r}^{w+r} \sum_{j=h-r}^{h+r} I_{i,j} \cdot \lambda^{-SF_{i,j}} \cdot (r - d + 1) \tag{13}$$

where $I_{i,j}$ states whether the core with coordinates i, j is idle (1) or not (0), and d is the Manhattan distance of the same core to the central node. Then, SF of each core is combined in terms of an exponentiation with a predefined base λ (5 in our case). In such a way, the relevance of each core is reduced based on its aging status. In the computation of Equation 13, for each core tagged as critical for any of the two aging mechanisms, $I_{i,j}$ is set to 0. Thus, the central node maximizing the metric represents the best

Algorithm 3 Reliability-aware Scheduling

Inputs:
- $\mathbf{mapList}$: vector of cores allocated for the current application
- $\mathbf{readyQueue}$: queue of ready threads for the current application
- $\mathbf{R_{EM}}$, $\mathbf{R_{TC}}$: vectors of cores reliability
- $\mathbf{status_{EM}}$, $\mathbf{status_{TC}}$: vectors of cores

Constants:
- n: threshold distinguishing warm and cold cores
- $n_{timeout}$: timeout threshold

Body:

```
1: SF_c ← compute_SF(mapList, R_EM, R_TC, status_EM, status_TC)
2: mapList ← sort(mapList, SF_c)
3: timeoutQueue ← ∅
4: warmQueue ← ∅
5: coldQueue ← ∅
6: for c ∈ mapList do
7:     use ← compute_previous_activity(c)
8:     if use > t_timeout then
9:         push(timeoutQueue, c)
10:    else if use > n then
11:        push(warmQueue, c)
12:    else
13:        push(coldQueue, c)
14: for t ∈ readyQueue do
15:    if timeoutQueue ≠ ∅ then
16:        c ← pop(timeoutQueue)
17:        schedule(t, c)
18:    else if warmQueue ≠ ∅ then
19:        c ← pop(warmQueue)
20:        schedule(t, c)
21:    else if coldQueue ≠ ∅ then
22:        c ← pop(coldQueue)
23:        schedule(t, c)
24:    else
25:        break
```

candidate since it has the maximum number of idle cores closer to itself and not considerable aged.

5.3 Reliability-aware Scheduling Unit

The Reliability-aware Scheduling unit is responsible for executing the applications' running threads while balancing and reducing the EM and TC stress. The unit takes in input the metrics produced by the Reliability Analysis unit and the current mapping of the running applications. The unit behavior has been designed as a standard single-queue multiprocessor scheduling algorithm [17]: 1) a thread balancing policy distributes in a uniform way the load among the various cores, and 2) a first-in-first-out ready queue is used for each core to implement a Round-Robin policy among application threads to fairly assign the same time slice to each of them in a periodic way. The novelty of our approach is in the employment of reliability-aware metrics to decide which core to prefer in the threads distribution. In particular, the scheduler will prioritize the usage of less stressed cores, especially in the case the number of running threads is lower than the number of cores reserved to the application (i.e., the application is not running its parallel section); otherwise, it will behave exactly as the standard approach. Moreover, since each core is assigned at most to a single application and application's threads have the same relevance, no priority mechanism is used in the Round-Robin scheduling.

The scheduler workflow is shown in Algorithm 3; it is awaken with a short time quantum (100 ms). It receives the queue of the ready threads in the application, i.e. all threads except the blocked ones, the list of cores assigned to the application, and the reliability metrics. First, cores are sorted based on a stress factor SF_c (Lines 1–2) computed with the current reliability values:

$$SF_c = w_{EM} \cdot R_{EM} + w_{TC} \cdot R_{TC} \tag{14}$$

Then, cores are divided in three queues according to an analysis of the temperature (Lines 3–13):

- *warm cores*, that have been used during the previous n scheduling epochs,
- *cold cores*, that have not been used during the previous n scheduling epochs, and
- *timeout cores*, that have not been used for a period larger than $n_{timeout}$ scheduling epochs (with $n_{timeout} \gg n$).

The scheduling policy prefers the use of warm cores rather than cold ones since this would reduce thermal cycles. In fact, if a thread is assigned to an warm core, its temperature will not vary considerably as well as if a cold core continues not being used. Indeed, since the scheduling quantum is shorter that the transitory in temperature variation, we have to consider n subsequent epochs to distinguish between warm and cold cores. n should be tuned based on the transitory duration. Based on this policy, cold cores may never be used; for this reason, when a timeout $t_{timeout}$ expires, such cores are put on the third list that will be considered with the highest priority in the scheduling activities.

The scheduling is performed by iterating on the ready threads until there is an available core in the three core queues which are used in a sequence (Lines 14–25). The fact that each queue is sorted based on SF_c allows to use less aged cores first, in case the number of threads is less than the number of cores. Finally, if threads are more than the available cores, the algorithm ends when all queues are empty (Line 25) and the scheduling of the exceeding threads is postponed to the next time quantum.

It is worth noting how the scheduling algorithm takes decisions by using primarily the temperature (the queue of *warm cores* is used before the one of *cold cores*) and secondarily the reliability metric (since cores in each queue are sorted according to SF_c). This is due to the fact that the temperature varies with a very short time horizon, equal to the scheduling quantum while the overall reliability very slow in the time. Therefore, keeping core at the same high temperature for several scheduling epochs will not have a perceivable effect on the reliability, and in particular on EM, while making its temperature to fluctuate will add more thermal cycles, thus negatively affecting TC. In conclusion, the scheduling unit mainly mitigate temperature fluctuations while the reliability balancing is mainly addressed by the mapping unit.

5.4 Reliability-aware DPM Unit

The Reliability-aware DPM unit controls the chip power consumption by acting on per-core DVFS and PCPG; in our proposal the Reliability-aware DPM unit performs a specific DVFS control based on the aging status of the cores. To take decisions, DPM unit receives in input the current mapping decisions, the aging status of the cores and the difference between the current power consumption and the given power budget. The power budget may be expressed by means of any approach, such as TDP or TSP [3].

The unit is awaken when there is a relevant variation in the consumed power, or when there is a variation in the available power budget (for instance when TDP is used). PCPG is simply actuated on all the cores not allocated to any running applications so that the system saves power and consequently reduces cores' temperatures and stress. Then, for the used cores, DVFS is tuned at the granularity of the single application by means of an heuristic approach aimed at maximizing the power budget utilization and distributing it by balancing applications' performance and cores' stress. To avoid performance bottleneck, DPM unit tunes the

same VF level to all cores allocated for a single application [36]. The selection of the candidate application(s) which cores will be subject to a DVFS tuning is based on a combination of four metrics considering performance and reliability aspects: i) applications' network intensity [18], ii) congestion in their mapping area [18], performance-power ratio [37], and a DVFS reliability cost here defined. To compute the first two metrics, the network interface of each core c in the architecture is assumed to be provided with hardware counters to measure the traffic, and in particular, the number of injected and deflected flits per unit of time, namely flit injection rate, inj_rate_c, and flit deflection rate $defl\_rate_c$, respectively. Thus, the network intensity of an application is computed (as in [18]) by averaging the value of inj_rate_c of all the cores where the application is mapped on:

$$int_{appl} = \frac{\sum_{c \in appl} inj\_rate_c}{|appl|} \tag{15}$$

Similarly, the network congestion of an application is computed (as in [18]) by averaging the the value of $defl\_rate_c$ of all the cores where the application is mapped on:

$$cong_{appl} = \frac{\sum_{t \in appl} defl\_rate_c}{|appl|} \tag{16}$$

The performance-power metric [37] estimates the average performance loss (gain) when decreasing (increasing) power consumption of the set of cores executing an application when the VF of the cores is varied from the current value $freq_c$ to a new value $freq_c^*$. The metric is computed as:

$$D^{appl}_{perf-power} = \frac{\sum_{c \in appl} \frac{perf_c^* - perf_c}{power_c^* - power_c}}{|appl|} \tag{17}$$

where $power_c$ is the current power of a core c, $perf_c$ is the current performance of core c, estimated by multiplying the average core utilization by the core frequency, $freq_c$. The future power and performance values for each core, $perf_c^*$ and $power_c^*$, respectively, are estimated as:

$$perf_c^* = perf_c \cdot \frac{freq_c^*}{freq_c} \tag{18}$$

$$power_c^* = power_c \cdot \frac{freq_c^*}{freq_c} \cdot \left(\frac{V^*_{dd\_c}}{V_{dd\_c}}\right)^2 \tag{19}$$

where V_{dd_c} and $V^*_{dd\_c}$ are the voltage levels associated to the two considered frequency levels of core c.

Finally, the reliability aspect is taken into account by means of a DVFS reliability cost RC^{VF}_{app} computed on the basis of the effects of a DVFS variation on the cores' SF, as follows:

$$RC^{VF}_{appl} = \sum_{c \in appl} SF^{VF}_c \tag{20}$$

where SF^{VF}_c is the SF of each core allocated to the application based on Equation 11, computed on the basis of the temperature due to DVFS change. The overall cost function is computed by combining the four metrics for each application $appl$ and summing up the results for all the running applications as follows:

$$VF_{cost} = \sum_{appl} \left[\frac{D^{appl}_{perf\_power}}{cong_{appl} \cdot int_{appl} \cdot RC^{VF}_{appl}} \right]^{sign(P_{error})} \tag{21}$$

Algorithm 4 Reliability-aware DPM

Input:
- P_{error}: power error
- P_{budget}: power budget
- $\mathbf{VF_{in}}$: current VF levels of the cores
- $\mathbf{appls}$: list of running applications
- $\mathbf{features_{appls}}$: list of pre-profiled applications features
- $\mathbf{R_{EM}}$, $\mathbf{R_{TC}}$: vectors of cores reliability

Internal registers:
- P^{t-1}_{error}: contains P_{error} of the previous control period

Constants:
- P_{th}: power threshold

Body:

1: **if** $|P_{error} - P^{t-1}_{error}| > P_{th}$ **then**
2: $\quad\mathbf{VF} \leftarrow \mathbf{VF_{in}}$
3: $\quad$**repeat**
4: $\quad\quad VFchanged \leftarrow false$
5: $\quad\quad\mathbf{VF_{last\_iter}} \leftarrow \mathbf{VF}$
6: $\quad\quad$**for** $appl \in \mathbf{appls}$ **do**
7: $\quad\quad\quad VF_{cost} \leftarrow$ estimate_VF$_{\text{cost}}$($\mathbf{VF}$, $\mathbf{features_{appls}}$, $\mathbf{R_{EM}}$, $\mathbf{R_{TC}}$)
8: $\quad\quad\quad$**if** $P_{error} > 0$ **then**
9: $\quad\quad\quad\quad\mathbf{VF^{new}} \leftarrow$ down-scale($appl$, $\mathbf{VF}$)
10: $\quad\quad\quad$**else**
11: $\quad\quad\quad\quad\mathbf{VF^{new}} \leftarrow$ up-scale($appl$, $\mathbf{VF}$)
12: $\quad\quad\quad VF^{new}_{cost} \leftarrow$ estimate_VF$_{\text{cost}}$($\mathbf{VF^{new}}$, $\mathbf{features_{appls}}$, $\mathbf{R_{EM}}$, $\mathbf{R_{TC}}$)
13: $\quad\quad\quad$**if** $VF^{new}_{cost} < VF_{cost}$ **then**
14: $\quad\quad\quad\quad\mathbf{VF} \leftarrow \mathbf{VF^{new}}$
15: $\quad\quad\quad\quad VFchanged \leftarrow true$
16: $\quad\quad P^{predicted}_{error}$ = estimate_power($\mathbf{VF}$, $\mathbf{features_{appls}}$) $- P_{budget}$
17: $\quad\quad$**if** $P_{error} > 0$ **then**
18: $\quad\quad\quad$**if** $P^{predicted}_{error} \leq 0$ **then**
19: $\quad\quad\quad\quad$EXIT $\leftarrow true$
20: $\quad\quad\quad$**else if** $VFchanged = false$ **then**
21: $\quad\quad\quad\quad$kill_an_app()
22: $\quad\quad\quad\quad$EXIT $\leftarrow true$
23: $\quad\quad$**else**
24: $\quad\quad\quad$**if** $P^{predicted}_{error} \leq 0$ **and** $VFchanged = false$ **then**
25: $\quad\quad\quad\quad$EXIT $\leftarrow true$
26: $\quad\quad\quad$**else if** $P^{predicted}_{error} > 0$ **then**
27: $\quad\quad\quad\quad\mathbf{VF} \leftarrow \mathbf{VF_{last\_iter}}$
28: $\quad\quad\quad\quad$EXIT $\leftarrow true$
29: $\quad$**until** EXIT $= false$
30: $\quad$apply_VF($\mathbf{VF}$)
31: $P^{t-1}_{error} \leftarrow P_{error}$

In the formula, P_{error} represents the difference between the current power consumption and the power budget (refer to Figure 4). Thus, $sign(P_{error}) = 1$ if $P_{error} \geq 0$ and $sign(P_{error}) = -1$ otherwise. As a result, this sign function makes the cost function value be reversed when $P_{error} < 0$, i.e., in the *up-scaling* process that is performed when the current power consumption of the system is below the given power budget.

Algorithm 4 shows the proposed reliability-aware DPM policy. The algorithm is executed at each fixed time interval (500ms) to check if there is a significant variation in the power consumption or in the power budget (in case TSP is used), and, therefore, to avoid performing DVFS changes every time there is a fluctuation in the power consumption. To do so, the absolute difference between the current P_{error} and the one at the previous instant of time P^{t-1}_{error} is checked to be greater than a given threshold (Line 1). Starting from the current cores VF levels (Line 2) a hill-climbing algorithm is used to iteratively select an application on which DVFS is changed. This heuristic allows to find a near optimal solution in a reduced time by performing selections based on the estimated VF_{cost}. At each hill-climbing iteration, all running applications are analyzed (Lines 6–15). For each application, the algorithm first estimates the VF_{cost}. Then, it computes new

TABLE 1
Baseline Processor, Cache, Memory, and Network Configuration

Processor spec	16nm technology, Core type: Niagara, TDP=90W, 2 GHz processor, 128-entry instruction window
Fetch/Exec/Commit width	2 instructions per cycle in each core; only 1 can be a memory operation
Memory Management	4KB physical and virtual pages, 512 entry TLBs, CLOCK page allocation and replacement
L1 Caches	32KB per-core (private), 4-way set associative, 64B block size, 2-cycle latency, split I/D caches, 32MSHRs
L2 Caches	256KB per core (private), 16-way set associative, 64B block size, 6-cycle bank latency, 32MSHRs, directories are co-located with the memory controller
Main Memory Network	16GB DDR3-DRAM, up to 16 outstanding requests per-core, 160 cycle access, 4 on-chip Memory Controllers
Network Router	2-stage wormhole switched, virtual channel flow control, 4 VCs per Port, 4 flit buffer depth, 4 flits per data packet, 1 flit per address packet. Network Interface: 16 FIFO buffer queues with 4 flit depth
Network Topology	8x8/10x10/12x12 mesh, 128 bit bi-directional links (32GB/s)
Chip area	$109mm^2$ (8x8), $170mm^2$ (10x10), $245mm^2$ (12x12)
Power budget	TDP={90W, 110W, 130W} or TSP

candidate VF levels of the application cores by means of a one level of *down-scaling*, if a violation of the power budget occurred, or *up-scaling*, if the power budget is underutilized (Lines 7–11). If the cost function VF_{cost}^{new} estimated for the new $\mathbf{VF^{new}}$ list is less than the cost function in the previous step of the iteration, $\mathbf{VF}$ is replaced with that new list (Lines 12–15). Then, the power of the selected VF levels is estimated and its difference with the power budget is computed, namely $P_{error}^{predicted}$ (Line 12). Based on $P_{error}^{predicted}$, the algorithm decides whether to continue the hill climbing or not. In case of *down-scaling* (Lines 17–22), the algorithm exits when the newly predicted error is lower than zero or if it is still larger than zero but no VF tuning has been performed; in the latter case, one of the applications (the younger one) is killed to recover from the power violation. In case of *up-scaling* (Lines 23–28), the algorithm exits when no further VF change is done or during the last iteration of the algorithm $P_{error}^{predicted}$ becomes greater than 0; in the latter case, VF changes are undone to the values of the previous iteration (Line 27). Finally, after the loop, the selected VF levels are applied.

6 Experimental Results

The approach has been experimentally evaluated in a software framework based on Noculator [14], a shared-memory NoC-based CMP simulator using Intel PIN tool. The CMP has been modeled based on a Niagara2 processor with SPARC architecture as reported in Table 1. Physical scaling parameters and power modeling and TDP were gathered from McPAT [38] and Lumos [39] and for the steady-state thermal model Hotspot [40] has been integrated in the simulator. EM has been modeled as in [9] by fitting A_{EM} to have a $MTTF_{EM} = 10$ years in a steady state condition at $T = 60°C$. Similarly, TC has been modeled as in [12] by characterizing A_{TC} to have a $MTTF_{TC} = 10$ years in a steady state condition with $\delta T = 20°C$, $T_{Max} = 70°C$, $\Delta t = 1$ hour and $m = 10$. Finally, we defined three different architectures: 8x8, 10x10 and 12x12, respectively. Larger single-chip architectures would not be realistic given the considered technological parameters. For each architecture we considered four different power budgets: TDP=$\{$90W, 110W, 130W$\}$ and TSP.

The executed workload is composed of several instances of the thirteen applications included in the multi-threaded PARSEC benchmark suite [41] that are randomly selected and issued into the system at run-time. We generated three different workloads, namely *heavy*, *normal*, and *light*, based on the rate of incoming applications. For heavy workload, the rate of incoming application is set in a way such that approximately more than 85% of the cores are busy at the moment. Similarly, for normal and light workloads, the application entering rate is adjusted to have approximately 60% and 40% of cores busy at the moment, respectively.

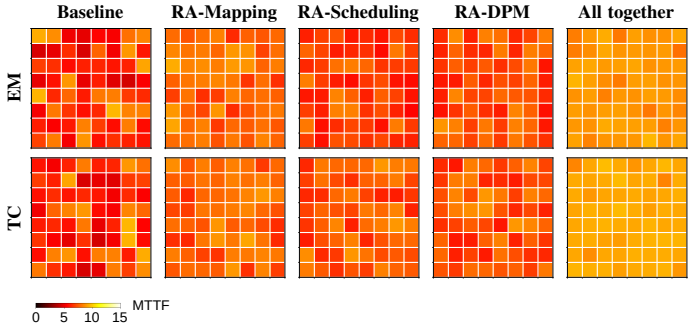


Fig. 6. Effect of the different reliability-aware units on core MTTF w.r.t. EM and TC for the 8x8 architecture.

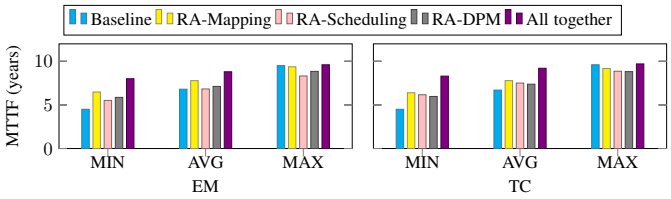


Fig. 7. Statistical analysis of the effect of the different reliability-aware units on core MTTF w.r.t. EM and TC for the 8x8 architecture.

The described tool has been used to simulate the system activity for approximately 20 hours in the various described configurations. Then, the MTTFs have been computed for each simulation according to the reliability model described in Section 2.2 by assuming the working conditions to be stationary for the entire lifetime (as assumed in various past works, e.g. [32], [9], [19]). More in detail, for EM the average aging rate of the entire simulation has been computed during the simulation by means of Equation 3, and Equation 5 is used to compute MTTF; similar computations are carried out for TC. Results of the experimental sessions are discussed in the following sections.

6.1 Effect of each reliability-aware units on MTTF

In a first experiment we analyzed the beneficial effect of each single reliability-aware units on system lifetime. To do so, we ran the experiment on the 8x8 architecture with the heavy workload and $TDP = 90W$ various times enabling a single unit at time. For the sake of completeness we also ran the baseline approach and the overall proposed approach, i.e., with all reliability-aware units disabled and enabled, respectively.

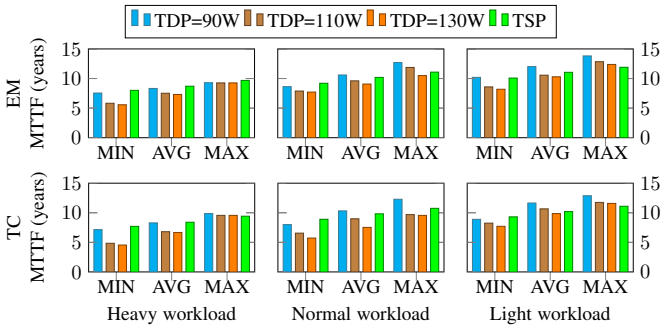


Fig. 8. MTTF for 8x8 architecture considering different workload setups and power budgets.

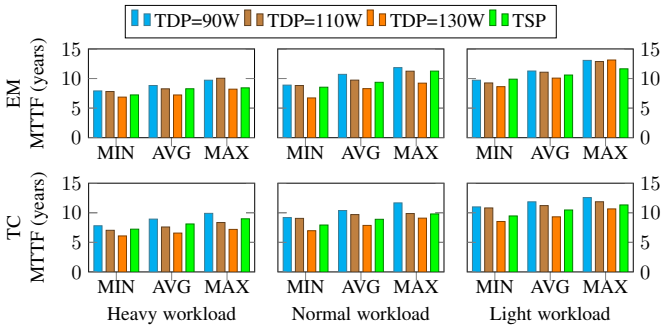


Fig. 9. MTTF for 10x10 architecture considering different workload setups and power budgets.

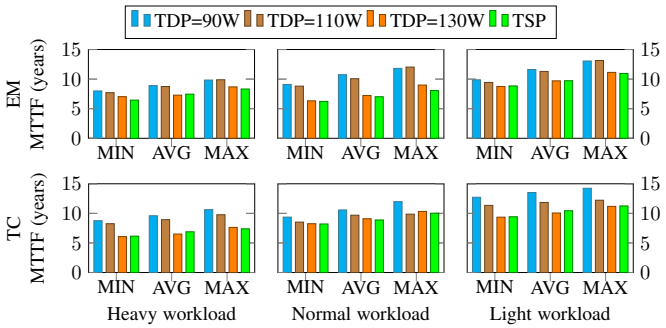


Fig. 10. MTTF for 12x12 architecture considering different workload setups and power budgets.

Figure 6 reports the MTTF of all the cores w.r.t. EM and TC, while Figure 7 shows the corresponding statistics. As it can be seen, each proposed reliability-aware unit can separately improve the overall EM and TC reliability comparing to the baseline. The reliability-aware mapping has a significant contribution to improve both EM and TC. The effect of DPM on balancing both EM and TC is milder in comparison with mapping. The reason is that without selecting the appropriate core that is happening in mapping process, DPM cannot contribute in balancing the reliability well. The effect of scheduling on balancing TC stress is more noticeable than their effect on balancing EM stress, in contrast with the effect of mapping. This is due to the fact that scheduling acts with a more fine-grain approach devoted to reducing the temperature fluctuations on each single core than the mapping and DPM that are more coarse-grained strategy. However, such short-term event does not affect EM stress much.

6.2 Analysis for different workloads and TDPs/TSP

In a second experiment, we evaluated how the proposed approach behaves with different workload intensities and power budges on the three considered architectures. The statistical analysis of the core MTTFs for 8x8, 10x10, and 12x12 architectures are reported in Figures 8, 9, and 10, respectively, for both EM and TC. Plots are not reported for the sake of space.

By decreasing the workload from heavy to light for all configurations and TDPs the EM and TC reliability improves. For example for 8x8 configuration and TDP=90 the EM and TC reliability (in particular the minimum cores' MTTF) improves 25% and 20%, respectively. The improvement of EM gets worse for lighter workloads compared with TC. For example for 8x8 configuration, obtained EM improvement is 25%, 18%, and 18% for heavy, normal, and light workload respectively. However, TC improvements does not follow as such and are 20%, 29%, and 31%. The same trend can be seen for 10x10 and 12x12 configurations. The reason for this is that with lighter workloads, reliability-aware units are not proportionally free enough to manipulate the temperature of the chip but are performing well to manage temperature fluctuation due to existence of vacant areas and possible temperature fluctuations in comparison against the baseline (without reliability-aware management).

By increasing TDP from 90W to 130W EM and TC reliability for 8x8 architecture decreases 12% and 18%, respectively, for the heavy workload; 14% and 27% for normal workload; and 10% and 15% for light workload. Similar trend can be observed for 10x10 and 12x12 architectures. Even though with higher TDP the aging rate exponentially increases, we noted that by increasing the TDP TC reliability gets better improvement comparing to EM. In fact, with higher TDP the degree of freedom for mapping decreases due to higher core utilization. This gives less freedom to the mapping unit to select different vacant areas to balance the reliability. However, due to high overall power consumption, the DPM unit is more triggered to decrease temperature fluctuations. Moreover, by increasing the TDP, the effect of the scheduler on balancing TC is more than EM since the throttling policy of power management reduces the temperature fluctuations that happens during the scheduling process.

Results related to TSP shows that for the 8x8 architecture of cores the aging is slightly slow; in fact, TSP is adjusted based on number and location of active cores in run-time. On the other hand, the 12x12 architecture ages faster when using TSP due to higher dedicated power budgets, leading at the same time to higher performance. For example, comparing to the obtained results for heavy workload, TDP=90W, in 8x8 configuration, using TSP gives slightly similar average EM and TC aging while with the similar settings of TDP and workload, for 10x10 and 12x12 architectures the EM (TC) aging gets worse by 6% (9%) and 25% (28%).

It is also interesting to analyze how MTTF varies in relationship with the number of architecture cores at a fixed TDP. When TDP is low, larger architectures age slower; in fact, the strict power budget forces a larger number of cores to be idle, causing at the same time a performance loss. For example comparing TDP=90W case for heavy workload experiment, obtained average values for EM (TC) for 10x10 and 12x12 configurations gets better 6% (8%) and 22% (13%) respectively comparing to 8x8 configuration. However, with the larger value for power budget, e,g., while using TDP=130W or TSP, the configuration with larger number of cores

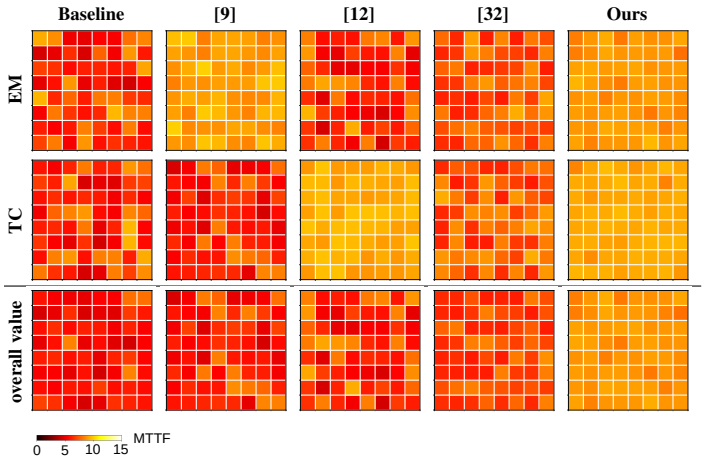


Fig. 11. MTTF comparison against past approaches for 8x8 architecture.

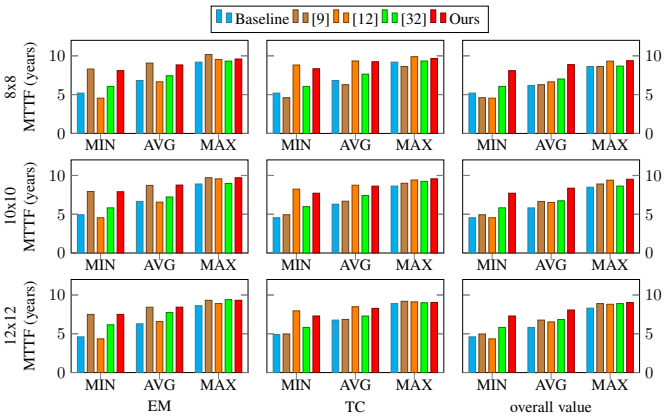


Fig. 12. Statistical analysis of MTTF comparison against past approaches for different architectures.

slightly ages faster (around 1-2%). This is mainly due to the higher effect of neighbor temperatures in 2D mesh environment for larger number of cores; the ratio of cores with less than four neighbors to all the cores is higher value for smaller number of cores.

6.3 Comparison against state-of-the-art approaches

In a third experiment, we compared the proposed approach against the most representative state-of-the-art works: i) the EM-aware approach in [9], ii) the TC-aware approach in [12], and iii) the EM/TC-aware approach in [32]. The baseline is also included for the sake of comparison. Experiments have been carried out on the three considered architectures running the heavy workload. First two rows of Figures 11 show the cores' MTTF for the various approaches w.r.t. EM, TC, and the last row the overall value, respectively, for the 8x8 architecture. The overall MTTF has been obtained for each core as the minimum between the two contributions (as in [9]). Figure 12 shows the statistical analysis for the results for all the three architecture. MTTF plots for 10x10 and 12x12 architectures have been omitted for the sake of space.

Results show that the proposed approach outperforms the other techniques by improving MTTF for both EM and TC. For the 8x8 architecture, the proposed approach presents an improvement of the minimum MTTF w.r.t. the reliability-agnostic baseline by 35% for the only EM, 37% for the only TC, and 35% for both types of aging mechanisms together, respectively. Statistics on average MTTF are 22%, 26%, and 30%, respectively. It can be seen that even through TC-aware technique in [12] results in a better MTTF for the only TC, the weak EM reliability improvement in this technique significantly affects the overall MTTF value. Also it can be noticed that the proposed approach outperforms in both EM and TC in comparison with EM-aware technique in [9]. The reason for improvement in EM based MTTF is that in [9] only EM reliability is considered partially in mapping and DPM units while the scheduling unit features no reliability-aware strategy. Thus, it may be noticed from the third line of plots that [9], [12] have no actual improvement of the overall MTTF w.r.t. the baseline.

In comparison with [32], the only one considering both EM and TC, our approach applied to the 8x8 architecture gains 25%, 27%, 25% minimum MTTF based on EM, TC, and both aging mechanisms together, respectively. Such improvements slightly decrease when the architecture size increases; in the 12x12 architecture, minimum MTTF improves by 17% for EM and 20% for TC. The approach in [32] is quite simplistic and considers only the thread scheduling without handling reliability-aware multi-threaded application mapping or DPM. Moreover, in the thread scheduling, it only performs basic core discrimination with the current reliability and temperature; at opposite, our approach performs an advanced decision marking based on both short-term and long-term analysis. Due to such weaknesses, this approach obtains worse results in EM mitigation than [9] and in TC mitigation than [12]. Finally, results reported in Figure 12 for the 10x10 and 12x12 architectures confirm all the drawn conclusions.

6.4 Effect of the proposed approach on performance

We analyzed the impact of the proposed reliability-aware approach on system performance. Figure 13 shows the normalized throughput of the proposed approach in comparison with the selected baseline for different workloads, architectures and power budgets. The performance penalty is actually negligible; this confirms that the proposed approach orchestrates decisions by properly taking into account at the same time reliability aspects and performance ones, by co-optimizing metrics referred to both aspects. The other reason is that considering the reliability metric is not necessarily in contrast with performance metric, but is sorts of metrics that in long-term shapes and can be considered as a variable that is independent from the performance metrics. Another important fact is that the penalty of the reliability-aware resource management slightly increases when the workload is intensive or the TDP is higher. In these cases the system experiences more stress; consequently, the proposed reliability approach puts some additional limitations slightly reducing the action space for performance optimization w.r.t. the baseline. Similar scenario occurs when TSP is used in 12x12 architecture since it sets a power budget on average larger than the maximum considered TDP (i.e., 130W).

6.5 Performance overhead of the proposed approach

Finally, since the proposed management approach is assumed to run on the same controlled CMP, we measured its overhead on the system activity. Practically, we used the described simulator to run also the four units composing the approach and measure execution times; average results for the three considered architectures are reported in Table 2. The overall overhead can be thus computed in terms of CPU utilization, i.e., the ratio between the controller execution time and the invocation period in percentage, since

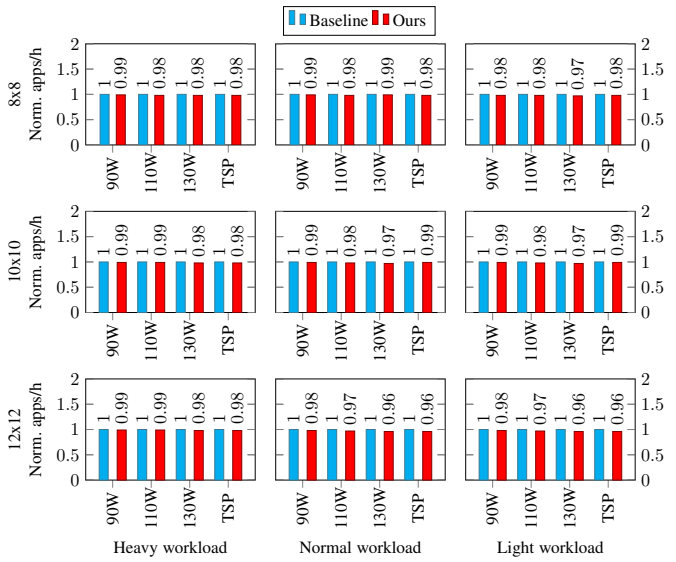


Fig. 13. Throughput analysis for the different workloads and TDPs/TSP.

this represent the percentage of compute power "stolen" to the workload. The frequency of invocation of each unit is reported in the second column of Table 2; for the Reliability Analysis unit and the Reliability-aware Scheduling one the call period is fixed and the obtained utilization is therefore equal to 0%.

Regarding to the other two units which call is event-triggered, the invocation frequency depends on the workload, and, in particular, on the average execution time of the various applications, and secondarily, the Reliability-aware DPM unit on the possible execution phases of the single applications. In the experiments, the execution times of the applications were on average equal to 30 minutes and on average 6/7 applications were running concurrently; in this configuration, we measured the Reliability-aware Mapping unit to be invoked every 5 minutes and the Reliability-aware DPM unit every 30 seconds/1 minute. Thus, the CPU utilization of the two units is close to 0%.

As a final note, even if Table 2 shows that execution times grow with the increase of the architecture size, the compute power of the architecture increase due to the larger number of cores as well. This allows us to confirm that the overhead of the approach, in terms of CPU utilization, is negligible.

7 Conclusions

This paper has presented a novel reliability-aware run-time resource management approach for CMPs aimed at improving architecture lifetime reliability w.r.t. multiple aging mechanisms, while maintaining same performance level of the state-of-the-art reliability-unaware counterpart and satisfying the power budget. The approach offers a fine-grained control aware of the aging status of the various cores in mapping, scheduling and power management, capable at dealing with both EM, mainly caused by high temperature levels, and TC, whose main driver is the temperature variations. Experimental results have shown the approach to outperform past works focusing on either a single aging mechanism or both the two ones at least by 17% and 20% w.r.t. EM and TC stress, respectively. Future work will extend the approach to work with heterogeneous system architectures.

TABLE 2
Execution time of the various units of the proposed approach.

Unit	Call frequency	Execution time (us) 8x8	10x10	12x12
Reliability Analysis	Periodic (1h)	9.6	15.0	21.6
RA Mapping	Event based (execution request)	512.28	1116.6	2301.48
RA Scheduling	Periodic (100ms)	32.2	53.8	77.9
RA DPM	Event based (power variation $>$ 2.5W)	557.0	1127.8	2150.46

Acknowledgment

The work has been financially supported by the Academy of Finland funded projects 335512 - ADAFI (Adaptive-Fidelity Digital Twins for Robust and Intelligent Control Systems) and 330493 - AURORA (Autonomous Performance Management in Digital Manufacturing), and by Nokia Jorma Ollila Grant.

References

[1] A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen, *The Dark Side of Silicon*, 1st ed. Springer, Switzerland, 2016.
[2] Semiconductor Industry Association *et al.*, "International Technology Roadmap for Semiconductors," http://www.itrs2.net/, 2011.
[3] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal Safe Power (TSP): Efficient Power Budgeting for Heterogeneous Manycore Systems in Dark Silicon," *IEEE Trans. on Computers*, vol. 66, no. 1, pp. 147–162, 2017.
[4] JEDEC, "Failure Mechanisms and Models for Silicon Semiconductor Devices," Tech. Rep. JEP122G, 2011.
[5] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, "System-level reliability modeling for MPSoCs," in *Proc. Conf. Hardware/Software Codesign and System Synthesis (CODES)*, 2010, pp. 297–306.
[6] A. Y. Yamamoto and C. Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors," *Microprocessors and Microsystems*, vol. 38, no. 1, pp. 53–63, 2014.
[7] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli, "Run-Time Mapping for Reliable Many-Cores Based on Energy/Performance Trade-offs," in *Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotech. Systems (DFT)*, 2013, pp. 58–64.
[8] A. S. Hartman and D. E. Thomas, "Lifetime improvement through run-time wear-based task mapping," in *Proc. Intl. Conf. Hardware/software codesign and system synthesis (CODES)*, 2012, pp. 13–22.
[9] M. Haghbayan, A. Miele, A. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era," *IEEE Trans. on Computers*, vol. 66, no. 9, pp. 1599–1612, 2017.
[10] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan, and M. Shafique, "Longevity Framework: Leveraging Online Integrated Aging-Aware Hierarchical Mapping and VF-Selection for Lifetime Reliability Optimization in Manycore Processors," *IEEE Trans. on Computers*, vol. 70, no. 7, pp. 1106–1119, 2021.
[11] Y. Ma, T. Chantem, R. P. Dick, and X. S. Hu, "Improving System-Level Lifetime Reliability of Multicore Soft Real-Time Systems," *IEEE Trans. on VLSI Systems*, vol. 25, no. 6, pp. 1895–1905, 2017.
[12] M. H. Haghbayan, A. Miele, Z. Zou, H. Tenhunen, and J. Plosila, "Thermal-Cycling-aware Dynamic Reliability Management in Many-Core System-on-Chip," in *Proc. Conf. Design, Automation & Test in Europe (DATE)*, 2020, pp. 1229–1234.
[13] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-core mapping policies to reduce memory system interference in multi-core systems," in *Proc. of Intl. Symp. on High Performance Computer Architecture (HPCA)*, 2013, pp. 107–118.
[14] R. Ausavarungnirun, C. Fallin, X. Yu, K. K.-W. Chang, G. Nazario, R. Das, G. H. Loh, and O. Mutlu, "A case for hierarchical rings with deflection routing: An energy-efficient on-chip communication substrate," *Parallel Computing*, vol. 54, pp. 29–45, 2016.
[15] M. H. Haghbayan, A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip," in *Proc. Intl. Symp. Networks-on-Chip (NOCS)*, 2015, pp. 1–8.

[16] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "CoNA: Dynamic application mapping for congestion reduction in many-core systems," in *Proc. of Intl. Conf. on Computer Design (ICCD)*, 2012, pp. 364–370.

[17] R. Arpaci-Dusseau and A. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. CreateSpace Independent Publishing Platform, 2018.

[18] A. M. Rahmani, M. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era," *IEEE Trans. on VLSI Systems*, vol. 25, no. 2, pp. 427–440, 2017.

[19] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan, and M. Shafique, "LifeGuard: A Reinforcement Learning-Based Task Mapping Strategy for Performance-Centric Aging Management," in *Proc. of Design Automation Conf. (DAC)*, 2019, pp. 179:1–179:6.

[20] S. Downing and D. Socie, "Simple rainflow counting algorithms," *Intl. Journal of Fatigue*, vol. 4, no. 1, pp. 31–40, 1982.

[21] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," in *Proc. of Intl. Symp. on Computer Architecture (ISCA)*, 2004, pp. 276–287.

[22] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Multi-Mechanism Reliability Modeling and Management in Dynamic Systems," *IEEE Trans. on VLSI Systems*, vol. 16, no. 4, pp. 476–487, 2008.

[23] A. K. Coskun, R. Strong, D. M. Tullsen, and T. S. Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors," in *Proc. Intl. Conf. Measurement and Modeling of Computer Systems*, 2009, pp. 169–180.

[24] K. Ma and X. Wang, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *Proc. Intl. Conf. on Parallel Arch. and Compil. Techniques (PACT)*, 2012, pp. 13–22.

[25] T. Kim, B. Zheng, H.-B. Chen, Q. Zhu, . V. Sukharev, and S. X.-D. Tan, "Lifetime Optimization for Real-time Embedded Systems Considering Electromigration Effects," in *Proc. of Intl. Conf. on Computer-Aided Design (ICCAD)*, 2014, pp. 434–439.

[26] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing, "WARM: Workload-Aware Reliability Management in Linux/Android," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1557–1570, 2017.

[27] A. Baldassari, C. Bolchini, and A. Miele, "A dynamic reliability management framework for heterogeneous multicore systems," in *Proc. of Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2017, pp. 1–6.

[28] Y. Ma, J. Zhou, T. Chantem, R. P. Dick, S. Wang, and S. X. Hu, "Online Resource Management for Improving Reliability of Real-Time Systems on "Big–Little" Type MPSoCs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 88–100, 2020.

[29] J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda, "Workload Assignment Considering NBTI Degradation in Multicore Systems," *Journal Emerg. Technol. Comput. Syst.*, vol. 10, no. 1, pp. 4:1–4:22, 2014.

[30] 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 *Proc. Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[31] A. Das, R. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proc. of Design Automation Conf. (DAC)*, 2014, pp. 170:1–170:6.

[32] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing Multicore Reliability through Wear Compensation in Online Assignment and Scheduling," in *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*, 2013, pp. 1373–1378.

[33] Y. Ma, J. Zhou, T. Chantem, R. P. Dick, S. Wang, and S. X. Hu, "Improving Reliability of Soft Real-Time Embedded Systems on Integrated CPU and GPU Platforms," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2218–2229, 2020.

[34] G. Southern and J. Renau, "Analysis of PARSEC workload scalability," in *Proc. Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 133–142.

[35] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine learning-based temperature prediction for runtime thermal management across system components," *IEEE Trans. on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 405–419, 2018.

[36] K. Chang, R. Ausavarungnirun, C. Fallin, and O. Mutlu, "HAT: Heterogeneous Adaptive Throttling for On-Chip Networks," in *Proc. Intl. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2012, pp. 9–18.

[37] K. Ma and X. Wang, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *Proc. Intl. Conf. on Parallel Arch. and Compilation Techniques (PACT)*, 2012, pp. 13–22.

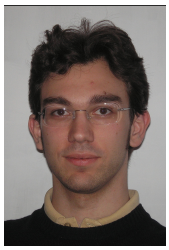
[38] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proc. Int. Symp. on Microarchitecture*, 2009, pp. 469–480.

[39] L. Wang and K. Skadron, "Implications of the Power Wall: Dim Cores and Reconfigurable Logic," *IEEE Micro*, vol. 33, no. 5, pp. 40–48, 2013.

[40] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware Microarchitecture: Modeling and Implementation," *ACM Trans. on Arch. Code Optim.*, vol. 1, no. 1, pp. 94–125, 2004.

[41] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, 2011.

Hashem Haghbayan (Member, IEEE) is an Adjunct Professor (docent) in embedded intelligent systems with the Department of Computing, Faculty of Technology, University of Turku (UTU), Finland. He received his M.Sc. in computer architecture from the University of Tehran, Iran, and Ph.D. with honors from the University of Turku, Finland. His research interests include machine learning, autonomous systems, high-performance energy-efficient architectures, and on-chip/fog resource management.



Antonio Miele (Senior Member, IEEE) is an Associate Professor at Politecnico di Milano, Italy. He holds a M.Sc. in Computer Engineering from Politecnico di Milano and a M.Sc. in Computer Science from the University of Illinois at Chicago. In 2010 he received a Ph.D. degree in Information Technology from Politecnico di Milano. His main research interests cover design of reliable computing systems, run-time resource management in heterogeneous multi-/many-core systems and FPGA-based systems design.

Onur Mutlu received the BS degree in computer engineering and psychology from the University of Michigan, Ann Arbor, and the MS and PhD degrees in electronics and communications engineering from the University of Texas at Austin. He is currently a professor of computer science with ETH Zürich. He is also a faculty member with Carnegie Mellon University, where he was the Strecker Early Career professor. His research interests include computer architecture, systems, hardware security, and bio-informatics.

Juha Plosila (Member, IEEE) is a Professor at the University of Turku (UTU), Finland, leading the Robotics and Autonomous Systems Unit at the Department of Computing. He received a Ph.D. degree in electronics and communication technology from UTU in 1999. His current research interests include intelligent adaptive multi-processing platforms, autonomous multi-robot systems, fog/edge computing, machine-learning, and optimization.