A Lightweight and Open-source Framework for the Lifetime Estimation of Multicore Systems

Cristiana Bolchini, Matteo Carminati, Marco Gribaudo, Antonio Miele
Dipartimento di Elettronica, Informazione e Bioingegneria – Politecnico di Milano, Milan – Italy
name.surname@polimi.it

***Abstract*—This paper presents a Monte Carlo-based framework for the estimation of lifetime reliability of multicore systems. Existing mathematical tools either consider only the time to the first failure, or are limited by their intrinsic complexity and high computational time. The proposed framework allows to compute quasi-exact results with a reasonable computational time, without adopting typical (and possibly misleading) simplifications that characterize the existing tools for computing Mean Time To Failure (MTTF). The paper describes the framework with all its mathematical details, assumptions and simplifications; it proves the correctness of the obtained results, by comparing them against the exact ones, and underlines the differences with the simplistic approaches, also discussing time overhead improvements.**

I. Introduction

Nowadays, reliability is playing a relevant role in the design of embedded systems due to the aggressive technology progresses that cause an increase of on-chip power and temperature densities and of other wear-out effects that negatively affect the device lifetime reliability [1]. Complexity of modern systems has pushed the focus of the design activities at system level; as a matter of fact, such systems are provided with a large number of (possibly heterogeneous) cores, and the executed workload is composed of many applications and may vary over time, both in the short period or with long-term phases. When considering reliability issues, system-level design choices (e.g., mapping and scheduling, operative frequency/voltage levels and further architectural knobs) have a macroscopic impact on the chip lifetime higher than circuit-level reliability enhancement techniques [2]. Thus, it is fundamental to be able to analyze the impact of each one of these aspects, not only on the single component but also on the overall system lifetime. This activity is particularly challenging when considering subsequent core failures in the architecture and when supporting runtime resource management strategies, whose effects on cores' aging and wear-out is a-priori unknown. We claim that the existing solutions ([2]–[5]) are either too simplistic (e.g., they consider the system to fail after the first core fails) or are limited in the features (e.g., homogeneous workload distribution and static design-time mapping), not allowing the designer to fully estimate the system's lifetime in a dynamic working scenario.

In this paper we present a novel lightweight and open-source framework for estimating the lifetime of multicore systems subject to a varying workload. The framework, dubbed CALIPER (monte CArlo LIfetime p reliability EstimatoR), supports: i) common failure mechanisms that model component failures as non-independent and non-identically distributed (non-i.i.d.) variables, ii) the computation of the lifetime of a system composed of n processors sharing a workload and able to tolerate a given number of k failures by means of load redistribution, and iii) the reliability curve analysis for long-period evolving workload scenarios. With respect to the existing solutions, the main contributions are:

- a complete mathematical background and the lifetime reliability evaluation framework, complementing what available in literature,
- a detailed formulation of the Monte Carlo based approach for computing the system's lifetime reliability,
- a comprehensive experimental validation of the proposed formulation, showing no accuracy loss, and
- open-source distribution[1].

The remainder of the paper is organized as follows. Section II introduces the related work on lifetime estimation and motivate this proposal. Section III sets the background for the presentation of the proposed framework and its main components, discussed in detail in the subsequent Sections IV-VI. An extensive validation based on different formulations is presented in Section VII. The last section draws some conclusions and discusses future extensions.

II. Related work

Devices aging and wear-out mechanisms, such as electromigration (EM), time dependent dielectric breakdown (TDDB), and negative bias temperature instability (NTBI), have been accurately studied and modeled [1]. These models usually consider a homogeneous device and assume a single steady-state operation mode at fixed (worst-case) temperature. These aspects constitute a significant limitation when taking into account modern systems composed of many processors, working in highly dynamic contexts, in terms of both the executed workload and its distribution on the cores, elements that highly affect system's aging rate.

One of the first system-level models for lifetime reliability evaluation is presented in [6]; it uses a sum-of-failure-rates (SOFR) approach to take into account several aging mechanisms. This work, as well as subsequent ones, considers a single core architecture [7] or adopts exponential failure distributions [4] that are though unable to capture accumulated aging effects. As a result, they are not suitable for multicore architectures supporting load-sharing.

When using Weibull or lognormal distributions to consider the aging history, as in [3] and [2], an important issue related to the computational complexity of the lifetime reliability evaluation arises. In fact, in each instant of time, the reliability

[1]The tool is available for download at: https://github.com/D4De/caliper

978-1-4799-6492-5/14/$31.00 ©2014 IEEE

of each component is a function of the current working conditions and the aging effects accumulated in the previous periods of activity. Actually, the simulation of the overall life of the system capturing the workload evolutions is a prohibitive time-consuming activity. Therefore, many approaches [2]–[4] simulate and trace only a subset of representative workloads for a reduced period of time with a fine granularity, then extrapolating the average temperature [4] or an average aging rate [2], [3] to be considered during the MTTF computation. However, while the former strategy is highly inaccurate (as shown in [3]), the latter is able to capture only workload changes in the short period of time considered for the simulation; they do not capture the evolution of the lifetime reliability curve due to changes that fall outside the simulation window.

Another issue in the computation of the lifetime reliability is the fact that the system may tolerate a given number of failures by remapping the workload from the failed cores to the healthy ones. As shown in some theoretical works [8]–[10], the system reliability formula is based on a multidimensional integral, whose dimension is given by the number of failures the system can tolerate. In practice, its analytical solution is unfeasible because the direct numerical computation is not affordable in terms of execution time. As a consequence, many approaches [3], [4], [11], [12] consider the entire system not to survive beyond the first failure, while others [13] compute the MTTF iteratively by selecting at each step the lowest MTTFs. Both solutions are inaccurate.

Finally, other approaches [2], [5], [9] adopt Monte Carlo simulation to quickly and accurately compute the multidimensional integral. However, they take into account only a "static" working scenario, with lack of generality and offering only a theoretical discussion of lifetime reliability estimation. In this work, we aim at overcoming such shortcomings.

III. Preliminaries

Reliability of a single system at time t, $R(t)$, is usually referred to as the probability that the system has been operational until t. As suggested by the JEDEC Solid State Technology Association [1], the lifetime reliability of a single digital component, such as a processor, can be modeled according to the Weibull distribution

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

being t the current instant of time (generally measured in hours), T the (steady-state) processor temperature (Kelvin degrees), β the Weibull slope parameter (considered to be independent of the temperature), and $\alpha(T)$ the scale parameter [14]. The lifetime of the processor is estimated in terms of its Mean Time To Failure (MTTF), defined as the area underlying the reliability function $R(t,T)$:

$$MTTF = \int_0^{\infty} R(t,T)dt. \tag{2}$$

The $\alpha(T)$ parameter formulation depends on the wear-out effects to be modeled, such as [1]: EM, TDDB, or NBTI. As a proof of concept, EM related wear-out failures will be considered in the experimental results; however, it is possible to integrate other effects either as standalone contributes or by using the SOFR approach for any combination of the above

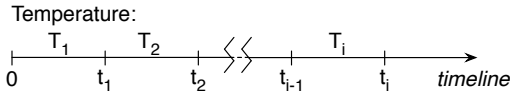

Figure 1. Model of a processor's temperature profile over time.

failure effects [6]. In the EM model, Black's equation is used to compute $\alpha(T)$:

$$\alpha(T)_{EM} = \frac{A_0(J - J_{\text{crit}})^{-n} e^{\frac{E_a}{kT}}}{\Gamma\left(1 + \frac{1}{\beta}\right)} \tag{3}$$

where A_0 is a process-dependent constant, J is the current density, J_{crit} is the critical current density for the EM effect to be triggered, E_a is the activation energy for EM, k is the Boltzmann's constant, n is a material-dependent constant, and $\Gamma()$ is to the gamma function.

To formulate $R(t)$ for the general scenario of a multicore system sharing a variable workload, the above formulas need to be revised to consider two specific aspects: i) temperature changes due to a variation in the workload, and ii) system composition in terms of its resources and its capability to survive beyond the first failure. These two aspects are here now briefly presented in relation to the scenario of the proposed approach; for a more precise formulation, it is possible to refer to [8], [9]. When the temperature changes as shown in Figure 1 (to keep the computation manageable, steady-state temperatures are considered), the exact computation of the $R(t)$ function needs to be accurately addressed by using Equation 1 and its inverse function, $R^{-1}(r,T)$, defined as:

$$R^{-1}(r,T) = \alpha(T) \cdot (-\log(r))^{1/\beta} \tag{4}$$

being r a reliability value. The $R^{-1}(r,T)$ function returns the elapsed time t, starting from time 0, when the system reaches the reliability value of r, under the influence of a fixed temperature T. In particular, the $R(t,T_1)$ function can be directly applied only on a new processor (i.e., when $t < t_1$, being t_1 the time at which a load redistribution takes place causing a change in the steady-state temperature). On the other hand, when $t \in (t_1, t_2]$, the processor reliability is equal to $R(t - t_1 + \hat{t}_1, T_2)$, where $\hat{t}_1 = R^{-1}(r_1, T_1)$ and r_1 is the reliability value in t_1 [8]. Similarly, the system's reliability for values of t in the next steps can be computed with a recursive approach by using the above-mentioned formulas. From a graphical point of view (shown in Figure 2), $R(t,T)$ is a curve starting from $t = 0$ (and $R(0,T) = 1$) with a varying shape that depends on the temperature T. Therefore, at each temperature change it is necessary to right-shift the new reliability curve in order for it to intersect the old reliability value.

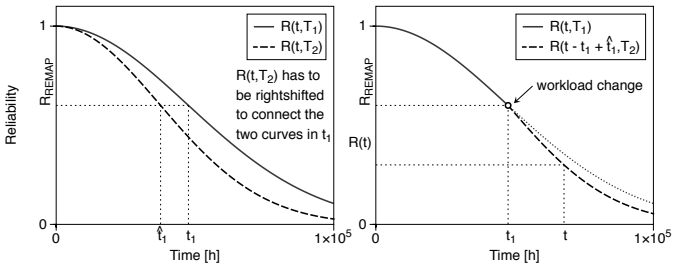

Figure 2. Reliability curve considering temperature changes.

Finally, the $R(t)$ formula considering a variable processor's temperature can also be rewritten in the following way [2]:

$$R(t) = e^{-\left(\sum_{j=1}^{i} \frac{\tau_j}{\alpha_j(T)}\right)^{\beta}}. \quad (5)$$

where τ_j represents the duration of each period of time with constant temperature T_j up to time t (i.e., $t = \sum_{j=1}^{i} \tau_i$).

All the equations discussed so far refer to the case of a system with a single processor. Things get way more complicated when a system consists of n processors, or cores, and is subject to a sequence of k failures, causing a redistribution of the workload after each failure occurrence. Such scenario is usually referred to as a load-sharing k-out-of-n:F system, made up of non-independent and non-identically distributed (non-i.i.d.) variables with arbitrary distribution. k ($\leq n$) is the minimum number of cores that have to fail before the entire system becomes out-of-service. The reliability of such system is computed by using the total probability formula:

$$R(t) = P_{no\_f}(t) + P_{1\_f}(t) + \cdots + P_{k-1\_f}(t). \quad (6)$$

$P_{no\_f}(t)$ is the probability that no failure occurred and is computed as the product of the reliability of all processors:

$$P_{no\_f}(t) = \prod_{i=1}^{n} R_i(t). \quad (7)$$

We call the MTTF to the first failure the integral between 0 and $+\infty$ of such formula; as discussed above, several works [3], [4], [11], [12] approximate the system MTTF to such value. $P_{1\_f}(t)$ is the sum of the probabilities of occurrence of n different cases where a single processor fails and its load is redistributed on the other working units:

$$P_{1\_f}(t) = \sum_{i=1}^{n} \int_0^{\infty} f_i(t_1) \cdot \prod_{j=1, j\neq i}^{n} R_j(t|\mathbf{t_1^i}) dt_1 \quad (8)$$

where $f_i(t)$ is the probability density function of $R_i(t)$, i.e., the probability that the failure occurred on processor i at a specific time t_1, and $R_j(t|\mathbf{t_1^i})$ is the conditioned reliability function of processor j knowing that processor i failed at time t_1. It is worth noting that conditioned reliability functions can be computed by using the formulas for load remapping, as shown in [8]. Similarly, $P_{2\_f}(t)$ is computed as:

$$P_{2\_f}(t) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \int_0^{\infty} \int_0^{t_2} f_i(t_1) \cdot f_j(t_2|\mathbf{t_1^i}) \cdot \cdot \prod_{m=1, m\neq i\neq j}^{n} R_m(t|\mathbf{t_1^i}, \mathbf{t_2^j}) dt_1 dt_2 \quad (9)$$

by using probability density functions and the reliability function conditioned by the previous sequence of failures, the first one occurred at t_1, the second one at t_2>t_1. Finally, the general scenario considering the sequence of $k-1$ failures is a $(k-1)$-dimensional integral that can be written recursively on the basis of Equations from (6) to (9).

This discussion shows that the direct numerical evaluation of the reliability of a k-out-of-n:F system subject to a variable workload is computationally feasible only for small values of k and, therefore, motivates once more the proposed approach.

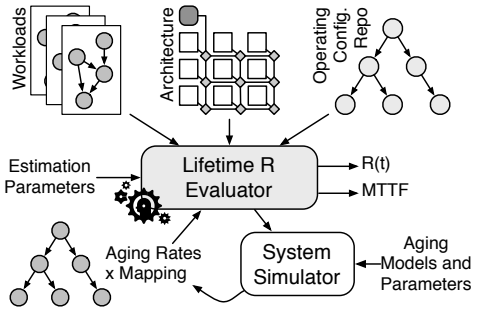

Figure 3. Workflow of the proposed framework.

IV. The proposed framework

The proposed framework, shown in Figure 3, takes as inputs the specifications of i) the system's architecture, ii) the workloads (made up of a set of applications), iii) the failure mechanisms of interest, and iv) the representative operating configurations. In particular, the architecture and application specifications are models that can be simulated, with the former one being characterized from a thermal point of view; moreover, the parameters characterization of the failure mechanisms is to be provided. The last input is the description of all possible working conditions of the architecture, corresponding to different healthy/failed processors with different workload distributions. These operating configurations are organized in a tree-based structure, called *operating configuration repository*, where the root represents the initial architecture and each path starting from it is a sequence of processor failures to be considered; therefore, first-level nodes correspond to single processor failures, i-level nodes to i failed processors. The depth of the tree is bounded by k failures, according to the designer's specifications. Each node (called *operating configuration*) in the tree is annotated with a set of mappings, each one describing how the workload is distributed on the healthy processors. There will be a single mapping in the scenario where changes in the workload distribution are only triggered by a processor's failure. Otherwise, when a dynamic workload scenario is envisioned, the time plan of the various workloads will be provided, by listing the set of considered mappings, each one with its own duration.

CALIPER is internally organized into two modules: the *lifetime reliability evaluator* and a supporting *system simulator*. The first module, the core of the proposed work, performs the lifetime reliability evaluation for the overall system by means of Monte Carlo simulations; more precisely, it estimates the system's reliability curve $R(t)$ and its expected lifetime, in terms of MTTF. To perform such computation, the evaluator module requires the aging characterization for the architecture's processors. This characterization must be computed under the influence of the executed workload in all operating configurations (and related mappings) considered in each specific Monte Carlo simulation. Therefore, the *system simulator* is invoked once for each considered mapping to estimate the thermal profile of the processors and to derive their aging rates $\alpha(T)$. The internals of the two modules are described in details in the upcoming Sections V and VI.

V. Aging rates estimation

The simulation engine is a service module used to characterize the lifetime reliability of the system at processor level; it

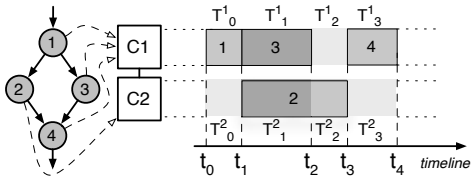

Figure 4. Example of application execution and thermal profile.

receives as input the operating configuration and the mapping, and produces as output the reliability characterization of each processor within the architecture for that scenario, in terms of the average aging rate. To this end, the main activity of this module is to estimate the thermal profile of each processor with respect to the given workload distribution.

Within the same operating configuration mapping, processor's temperature varies significantly from application to application and, at a finer granularity, with different applications' tasks; moreover, it is also affected by thermal interferences with neighbor units on the architecture floorplan. As an example, Figure 4 reports the thermal profile associated with the execution of a given application, modeled by a periodic task graph, with respect to a given mapping (containing also the scheduling information): the temperature of a processor depends also on the activity of the neighbor units, running their part of the application. Thus, architecture and application models have to capture all these aspects; more details will be provided in Section VII for the specific solution we adopted in the proposed framework prototype.

Since for each mapping in the operating configuration repository the workload is almost stationary, as shown in [2], it is not necessary to simulate the system for its complete life and to compute the reliability curve by means of Equation 5; such activity would be actually computationally non affordable. Indeed, it is possible to compute the average aging rate by tracing the system execution for a representative time period and by using the following formula:

$$\alpha = \frac{\sum_{i=0}^{p} \tau_i}{\sum_{i=0}^{p} \frac{\tau_i}{\alpha_i(T)}} \tag{10}$$

where τ_i represents the duration of the p atomic steps (with constant temperature on each core) within the simulated period. Then, the reliability evaluator uses such values as input parameters in Equations 1 and 4 to derive the lifetime reliability of the overall system. As a final note, even if simulated times may span from few microseconds to several hours, the result is the average aging per time unit, since Equation 10 is a ratio between the accumulated aging rates and the time window; considering the units of measure of the various parameters, the α value is generally referred per hour.

VI. Reliability computation

The second module of the framework, the main innovative contribution of this work, is devoted to the evaluation of the lifetime reliability curve $R(t)$ of the entire system and the related MTTF. Since numerical evaluation of the model presented in Section III is computationally unsustainable for more than 4 subsequent failures (as shown in Section VII), this activity is performed by means of Monte Carlo simulations to greatly reduce the computational time while obtaining accurate results. In the following subsections, the basic approach is discussed in details, and two improvements are then presented: i) the automatic tuning of the number of simulations needed to obtain a certain confidence in the obtained results and ii) the handling of variable workloads.

A. System reliability estimation for constant workloads

The basic scenario we consider (non-i.i.d. load-sharing) is characterized by a change in the workload distribution on the healthy cores only when a core fails. Multiple subsequent failures are considered: the whole system fails when the number of healthy processors is below a threshold specified by the designer. The pseudo-code for the lifetime computation is shown in Algorithm 1 and here briefly explained. Before getting into the details, for readability's sake, it is worth redefining the two functions $R()$ and $R^{-1}()$, describing the reliability of a single core, to take as input the α parameter, computed by means of the system simulator, in place of the temperature T. The inputs of the algorithm are: i) the number `n` of cores in the architecture, ii) the number of failures `k` that causes the entire system to become out-of-service, and iii) the number of simulations to be performed `#iter`.

A single Monte Carlo test consists of a simulation of a randomly-chosen sequence of processor failures' events; the goal is to compute the time to failure (TTF) of the overall system during that specific simulation. To compute the lifetime reliability, the same procedure must be iterated `#iter` times. After an initialization phase (Lines 2–5), the random walk step takes place and is repeated until the number of healthy cores reaches `n-k` (Lines 7–27). Starting from a completely healthy architecture, for each iteration a specific subset of healthy cores with a given workload distribution is selected and a failing event happens; the core failing at time t is randomly chosen among the healthy ones. This failure triggers the update of the system reliability based on the ones of the remaining healthy cores. To compute the α values for each step, the `getAlpha()` function is used; it invokes the simulation engine by specifying the current operating configuration `currConfig`. For all the healthy cores `i`, a new reliability value `randR` is randomly chosen within $[0, currR[i]]$ (Line 11): it represents the reliability at which that core will fail. The corresponding time instant `t` (all times are measured in hours) for such failure is computed (Line 12). Moreover, the time period between the current failure and the subsequent one needs to be computed, by evaluating the equivalent time `eqT` and subtracting it to `t` (Lines 13–14, please refer to Section III for the theoretical details). After having computed the failure times for all healthy cores, the smallest one is selected to identify the next event, i.e., the next core to fail; the simulation is thus advanced until such `t`, the core is removed from the list of healthy ones (Lines 21–22), and the reliability values of all remaining cores are updated (Line 24), to begin with a new iteration. At the end of each simulation, the value called $\texttt{MTTF}_\texttt{m}$ is updated by computing the sample mean of the system's TTF values experienced in the last m iterations (Line 29).

When all the simulations have been performed, collected data are summarized by generating the system reliability curve (Lines 32–39); for each time t corresponding to a failure, the value of R is computed as the ratio between the number of simulations with time greater than the current time and the

Algorithm 1 Monte Carlo-based reliability computation

Input: n, k, $\#iter$
Output: $MTTF_m$, R plot
1: **for** $m = 1$ **to** $\#iter$ **do**
2: $currConfig = getInitConfig()$
3: $healthy = \{0, 1, ..., n-1\}$
4: $currR = \{1, 1, ..., 1\}$
5: $totalTime = 0$
6: $TTFAccumulator = 0$
7: **while** size$(healthy) > n - k$ **do**
8: $failingCore = -1$
9: $alpha = getAlpha(currConfig)$
10: **for all** i **in** $healthy$ **do**
11: $randR =$ random$() \cdot currR[i]$
12: $t = R^{-1}(randR, alpha[i])$
13: $eqT = R^{-1}(currR[i], alpha[i])$
14: $t = t - eqT$
15: **if** $failingCore == -1$ **or** $failTime > t$ **then**
16: $failingCore = i$
17: $failTime = t$
18: $prevEqTime = eqT$
19: **end if**
20: **end for**
21: $totalTime = totalTime + failTime$
22: $healthy.remove(failingCore)$
23: **for all** i **in** $healthy$ **do**
24: $currR[i] = R(failTime + prevEqTime, alpha[i])$
25: **end for**
26: $currConfig = updateConfig(failingCore)$
27: **end while**
28: $TTFAccumulator = TTFAccumulator + totalTime$
29: $MTTF_m = TTFAccumulator/m$
30: $stats[totalTime] + +$
31: **end for**
32: $currHealthyIter = \#iter$
33: $precT = 0$
34: **for all** t **in** $stats.keys()$ **do**
35: $currHealthyIter = currHealthyIter - stats[entry]$
36: $currRvalue = currHealthyIter/\#iter$
37: $precT = t$
38: **print** t, $currRvalue$
39: **end for**
40: **return** $MTTF_m$

total number of performed simulations. Finally, the algorithm displays the reliability curve and returns the $\texttt{MTTF}_\texttt{m}$, representing the MTTF of the system updated to the last iteration.

B. Evaluating the iterations number

Since Monte Carlo simulations are a stochastic approach, it is necessary to assess a certain level of accuracy in the obtained results. Thus, the basic algorithm has been enhanced to automatically tune the number of tests `#iter` to be performed to obtain a given confidence level. The program iterates the simulations until either a specified percentage $c\%$ of the width of the confidence interval for the $MTTF_m$ value is less than a given threshold th, or the maximum number of iterations `#maxiter` is reached (to avoid starvation). The first condition is expressed as:

$$2 \cdot \Psi^{-1}\left(\frac{1+c\%}{2}\right) \cdot \frac{c_v[TTF_m]}{\sqrt{m}} \leq th \qquad (11)$$

where m is the number of performed tests, $c_v[TTF_m]$ is the coefficient of variation of the m TTF samples considered (i.e., the same ones used to estimate $MTTF_m$), computed as $\frac{\sqrt{Var[TTF_m]}}{MTTF_m}$, and $\Psi^{-1}(x)$ is the inverse of the cumulative normal distribution. $c\%$, th and `#maxiter` are received as input in place of `#iter`.

Algorithm 2 Computation of the next core to fail for dynamic workload changes

Input: $currConfig$, $randR$, $currR[i]$, $alpha[i]$, $totalTime$
Output: t
1: $currMapping = getCurrMapping(currConfig, t)$
2: $period = getTimeToNextChange(currConfig, t)$
3: $t = totalTime$
4: $alpha = getAlpha(currConfig, currMapping)$
5: $eqT = R^{-1}(currR[i], alpha[i])$
6: $testR = R(eqT + period, alpha[i])$
7: **while** $testR > randR$ **do**
8: $t = t + period$
9: $currR[i] = testR$
10: $currMapping = getCurrMapping(currConfig, t)$
11: $period = getTimeToNextChange(currConfig, t)$
12: $alpha = getAlpha(currConfig, currMapping)$
13: $eqT = R^{-1}(currR[i], alpha[i])$
14: $testR = R(eqT + period, alpha[i])$
15: **end while**
16: **if** $currR[i] > randR$ **then**
17: $eqT = R^{-1}(currR[i], alpha[i])$
18: $lastT = R^{-1}(randR, alpha[i])$
19: $t = t + lastT - eqT$
20: **end if**
21: **return** t

C. Reliability estimation for varying workloads

Frequent workload remappings can be approximated by using average aging rates as demonstrated in literature [3]; however, when considering long-term workload changes, such method may not be accurate enough. To this purpose, as discussed in Section IV, the framework supports the specification of a sequence of mappings for each operating configuration, each one provided with its duration. Algorithm 1 has been slightly revised to support such aspect; more precisely, the computation of the new reliability value (Line 12) is replaced with the code snippet listed in Algorithm 2: it performs a step-by-step update of the reliability value and related elapsed time by using $R()$ and $R^{-1}()$ functions (as already described in Section III) based on the specified workload time plan. Do note that `getCurrMapping()` and `getTimeToNextChange()` functions return the mapping and the related period for the current operating scenario.
Similarly, the reliability computation of the remaining healthy cores upon a core failure (Line 24 in Algorithm 1) needs to be adapted as well, with a similar strategy (the pseudo-code for this function is omitted due to lack of space).

VII. Experimental evaluation

A SystemC discrete event simulator (with a structure similar to [3], [12], [15], [16]) has been developed, while HotSpot [17] has been used to characterize the architecture's thermal profile. Do note that CALIPER can be integrated in a state-of-the-art framework, such as the one in [18].
The system considered in the experimental sessions is a mesh-based System-on-Chip architecture with $N \times M$ processors, spanning from 2×1 to 3×4. The system executes a workload that is uniformly distributed among the healthy cores, such that all cores have a 50% occupation in the initial healthy architecture. The workload is then redistributed after the occurrence of each failure and/or a given period. In the experiments,

we consider the system to be out-of-service when, after a remapping, the workload per core is higher than 100%. Finally, the EM-related parameters have been borrowed from [12].

A. Approach Validation

The approach has been validated against the exact formulation presented in Section III based on multidimensional integrals. The exact model has been solved by means of numerical methods (step discretization and rectangles). The discretization step has been set to 500 hours, causing an overall negligible error, since the $R()$ curve is almost equal to zero at $400,000$ hours, in a 1-out-of-2:F scenario. Due to the high computational complexity of exact approaches, validation against them has been feasible for architectures with at most 2×3 cores, tolerating up to 3 failures. In all the performed tests, the observed difference between the MTTF computed by means of exact approaches and the one computed by means of CALIPER is lower than 1%; do consider that such error may also be related to the discretization step. All the scalability experiments discussed in this section have been run with a constant workload throughout the whole system lifetime.

The main scalability problem of the exact method is related to the $(k-1)$-dimensional integral to be solved for a system failing after k failures (Equation 6). Such an integral has an asymptotic complexity $O(t^k)$, being t the overall simulation duration. In fact, when considering a 2×3 architecture, the exact method took less than 3 seconds for computing the MTTF for $k = 1$ or $k = 2$ failures; the computation time increased to 1 minute and 10 seconds for 3 failures, while requiring more than 2 hours for 4 failures. On the other hand, the execution complexity of CALIPER scales almost linearly w.r.t. the number of performed iterations and the number of simulated failures, as shown in Figure 5, which considers a 3×4 architecture. We measured the same error ($< 1\%$) in the experiments for which we were able to perform the validation. Do consider that aging rates have been precomputed and tabulated in an input file to perform an analysis of the performance of the Monte Carlo simulations approach only.

It is also interesting to analyze how the number of iterations for a fixed width of the confidence interval ($c\% = 95\%$) and threshold parameter ($th = 0.1$) changes w.r.t. to the considered architecture and the number of failures k. Figure 6 shows that the number of iterations is almost constant if k is fixed, regardless of the considered architecture's dimension and topology, and more important, it decreases as k grows. This could seem to be a surprising result, but the explanation is actually pretty intuitive: as it can be seen in Equation 11 the width of the confidence interval (i.e., the left side of the inequality) is proportional to the coefficient of variation $c_v[TTF_m]$. Increasing the number of failures decreases the coefficient of variation, and thus decreases the number of required iterations for the algorithm to converge. Figure 7 empirically proves this trend. A theoretical proof of this behavior can be easily provided for the exponential case, since the corresponding distribution becomes hypoexponential; the same idea holds for the Weibull case.

B. Constant workload scenarios: comparison

CALIPER has been compared against some state-of-the-art approaches assuming a constant workload scenario:

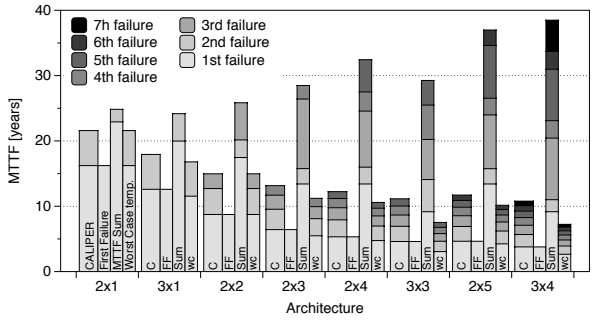

Figure 8. Comparison vs. past works for constant workloads.

- the Mean Time To First Failure (*FF*), adopted in [3], [4], [11], [12], where the whole architecture is considered to be out-of-service when the first core fails. In this case the reliability is computed by exploiting Equation 6;
- the sum of MTTF (*Sum*), proposed in [13]: in this case, at each step, the core selected to fail is the one with the lowest MTTF, the reliability for the healthy cores is updated and a new step is computed, until the architecture cannot reach the required performance;
- the worst case temperature (*wc*) [1]: at each step, the worst case temperature in the architecture is considered for computing the system aging.

Figure 8 reports the obtained results. While all the considered approaches have comparable execution times (same trend for *FF* and *wc* and around 1 second for *sum*), estimated lifetimes are very different. The simplification considering only the first failure clearly underestimates the real estimated lifetime of the architecture. As expected, the error increases as the number of core failures k increases: from 33.1% (1-out-of-n:F) to 185.4% (6-out-of-n:F). The *sum* approach proves to be even less precise, by highly overestimating the evaluated system lifetime. Again, the error range depends on k, reaching 256.9% for the 6-out-of-n:F case. These two approaches show trends in the successive failures that are not proportional at all; this means that the results produced by the *sum* approach are affected by a considerable non-systematic error. *wc* is the approach that provides the closest results to the CALIPER ones. The lifetime is here underestimated as well, but the error is significantly smaller, at an average of 14.2%. It is worth noting that this approach obtains exact results when the worst case temperature corresponds to the actual one: this is usually true for very small architectures (namely, 2×1 and 2×2 in our experiments), since, when the workload is uniformly distributed, temperatures of all cores are almost the same.

In conclusion, as a side note, it is interesting to highlight how, given the same starting workload for each core in each architecture, smaller architectures have longer lifetimes. This is due both to the higher thermal interferences that take place in bigger architectures and to the fact that the actual overall workload is bigger; in fact, if the same workload had been used for all the experiments, bigger architectures would have had longer lifetime than smaller ones.

C. Varying workload scenarios: comparison

The aim of the last experimental session is to show how CALIPER is able to provide more accurate estimations of

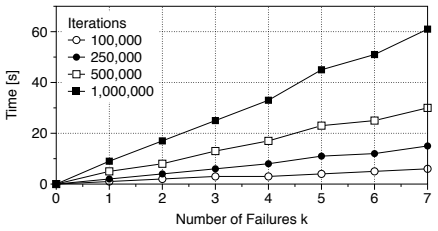

Figure 5. Simulations execution times w.r.t. number of failures k.

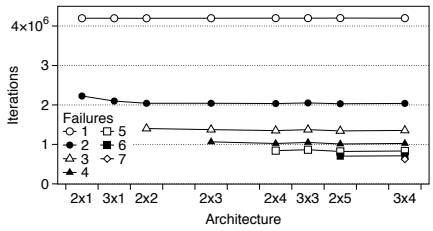

Figure 6. Iterations number w.r.t. different architectures topologies.

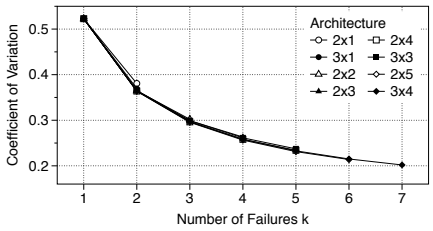

Figure 7. Coefficient of variation w.r.t. number of failures k.

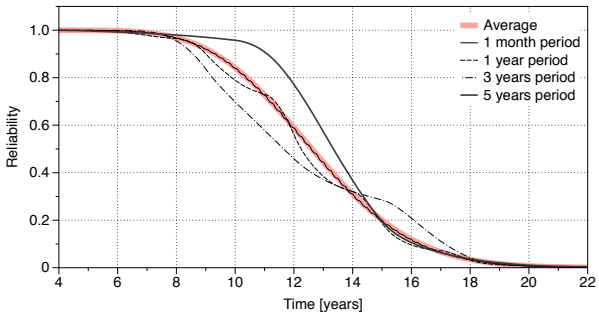

Figure 9. Reliability curve for different workload change's periods.

lifetime reliability than state-of-the-art techniques in scenarios considering varying application workloads. Here a 3×4 architecture is considered, with 5 as a maximum number of tolerated failures and a uniformly distributed workload periodically changing among 60%, 40%, and 15% single core utilization of the initial architecture. Figure 9 shows the reliability curves for various workload-change period spanning from 1 month to 5 years and compares them against average aging rates computed based on Equation 10 [2], [3]. From the figure it is possible to note how the approach in [2], [3] is able to accurately approximate workload changes with periods up to some months (curve oscillations are actually imperceptible), but it fails with longer periods. Obtained MTTFs slightly differ w.r.t. the average approximation up to 9% with the 5-year period; however, the maximum difference in the reliability curve with the average approximation spans from 0.07 of the 1-year period, to 0.15 of the 3-year period, to 0.19 of the 5-year period (over the $(0, 1]$ reliability range). Do note that having a precise estimation of the shape of the reliability curve is fundamental, for example, in computing which are the most probable failure periods, beside the estimated MTTF. In practical scenarios, the higher accuracy of the proposed tool has a considerable relevance in the overall evaluation of systems expected to experience highly different working phases during their operational life, with long-term periods with intensive workloads and periods of resources' underutilization.

VIII. Conclusions

This paper presented CALIPER, a framework for the estimation of the lifetime reliability for multicore systems, based on Monte Carlo simulations approach. The proposed framework allows to compute quasi-exact results with a reasonable computational time, without adopting typical extreme (and usually misleading) simplifications that characterize the existing tools for computing MTTF. Extensive experimental campaigns proved the effectiveness of the approach.

References

[1] JEDEC Solid State Tech. Association, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122G*, 2010.

[2] Y. Xiang, T. Chantem, R. Dick, X. Hu, and L. Shang, "System-level reliability modeling for MPSoCs," in *Conf. Hardware/Software Codesign and System Synthesis*, 2010, pp. 297–306.

[3] L. Huang and Q. Xu, "AgeSim: A simulation framework for evaluating the lifetime reliability of processor-based SoCs," in *Conf. Design Autom. & Test in Europe*, 2010, pp. 51–56.

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

[5] A. Hartman and D. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Int. Conf. Hardware/software codesign and system synthesis*, 2012, pp. 13–22.

[6] J. Srinivasan, S. Adve, P. Bose, and J.A.Rivers, "The case for lifetime reliability-aware microprocessors," in *Int. Symp. Computer Architecture*, 2004, pp. 276–287.

[7] A. Coskun, T. Simunic, K. Mihic, G. D. Micheli, and Y. Leblebici, "Analysis and optimization of MPSoC reliability," *J. Low Power Electronics*, pp. 56–69, 2006.

[8] H. Liu, "Reliability of a load-sharing k-out-of-n:G system: non-iid components with arbitrary distributions," *Trans. Reliability*, vol. 47, no. 3, pp. 279–284, 1998.

[9] L. Huang and Q. Xu, "Lifetime reliability for load-sharing redundant systems with arbitrary failure distributions," *Trans. Reliability*, vol. 59, no. 2, pp. 319–330, 2010.

[10] E. Salehi, M. Asadi, and S. Eryilmaz, "Reliability analysis of consecutive k-out-of-n systems with non-identical components lifetimes," *J. Stat. Planning and Inference*, vol. 141, no. 8, pp. 2920–2932, 2011.

[11] T. Chantem, Y. Xiang, X. Hu, and R. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Conf. Design Autom. & Test in Europe*, 2013, pp. 1373–1378.

[12] L. Huang, F. Yuan, and Q. Xu, "On task allocation and scheduling for lifetime extension of platform-based MPSoC designs," *Trans. Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2088–2099, 2011.

[13] Z. Gu, C. Zhu, L. Shang, and R. P. Dick, "Application-specific MPSoC reliability optimization," *Trans. VLSI Systems*, vol. 16, no. 5, pp. 603–608, 2008.

[14] L. Huang, R. Ye, and Q. Xu, "Customer-aware task allocation and scheduling for multi-mode MPSoCs," in *Design Autom. Conf.*, 2011, pp. 387–392.

[15] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Conf. Design Autom. & Test in Europe*, 2013, pp. 689–694.

[16] 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 *Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2013, pp. 58–64.

[17] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *Trans. Architecture Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[18] S. Corbetta, D. Zoni, and W. Fornaciari, "A temperature and reliability oriented simulation framework for multi-core architectures," in *Int. Symp. on VLSI*, 2012, pp. 51–56.