Lifetime-aware Load Distribution Policies in Multi-core Systems: An In-depth Analysis

Cristiana Bolchini, Luca Cassano, Antonio Miele
Dip. Elettronica, Informazione e Bioingegneria - Politecnico di Milano, Italy
Email: cristiana.bolchini@polimi.it, luca.cassano@polimi.it, antonio.miele@polimi.it

***Abstract*—Dynamic Reliability Management solutions are often adopted in multi-core systems to mitigate aging and wear-out effects, by opportunely distributing the workload on the available cores. The efficiency of such solutions is generally evaluated by considering only the occurrence of the first core failure due to the computational complexity. In this paper we propose an in-depth analysis of such approaches by considering the occurrence of multiple subsequent core failures, thus offering a more precise estimation of the lifetime reliability. In particular, we analyzed two classical load distribution approaches: a load balancing strategy versus a strategy based on spare resources. Experimental results show benefits and limitations of the considered solutions in terms of lifetime reliability while fulfilling system performance.**

***Index Terms*—Aging, Dynamic Reliability Management, Lifetime Reliability, Multi-core Systems.**

I. Introduction

Dynamic Reliability Management (DRM) has become an attractive solution to deal with lifetime issues in multi-core systems [1]. In the last decades, handling *reliability* in digital systems has become a mandatory challenge due to the acceleration of the circuits' aging and wear-out trend caused by the aggressive downscaling of CMOS technologies. In fact, transistor shrinking has caused a considerable increase of power densities that exacerbates temperature-related wear-out mechanisms, such as time dependent dielectric breakdown (TDDB), negative bias temperature instability (NBTI), electromigration (EM) and thermal cycling (TC), drastically reducing system service life. Moreover, given the highly variable workload (in terms of applications' types, amount of data to be processed, requirements and arrival rates) that computing systems have to execute, a *dynamic* management of the system activity is also required.

DRM strategies for multi-core systems proposed in the last decade (e.g. [2], [3], [4]) mainly act on the load distribution, and secondarily on architectural knobs such as voltage/frequency settings and core power-gating, with the aim of balancing the stress caused by running applications. Moreover, they are generally devoted to the optimization of the trade-off between the lifetime reliability and some other parameters, such as the system's performance or the power consumption. Strategies available in literature are highly heterogeneous. In [2], [3], [5], [6] the stress is kept homogeneous among units to postpone the next core failure, while another strategy proposed in [2] reserves spare units to be employed at the occurrence of a failure of an active core; finally other strategies (e.g. [2], [7]) adopt a rotation-based policy to periodically vary the set of active cores. However, a limitation of several of the previous solutions (e.g. [7], [4]) is to consider the system to be out of service after the first core failure, while only few approaches prolong the service life by redistributing the workload on the remaining cores. Moreover, some approaches (e.g. [1], [3]) perform dynamic adaptation based on the Mean Time To Failure (MTTF) estimated on the basis of the previous history; however, in case of a highly variable workload this is not effective since no assumption can be done on the future.

Unfortunately, in such an ecosystem it is quite difficult to identify the best strategy since no systematic experimental analysis has yet been proposed, except for [4], [2]. However, these two works are limited in their results. In [4] a set of DRM strategies is evaluated, but only considering the system to fail after the first core failure. In [2], an accurate theoretical modeling of the effects of workload variation on system lifetime reliability is proposed. Yet, the experimental campaign is not systematic; e.g. the analysis is suspended after 4 subsequent failures over 32 available cores, not exploring the graceful degradation possibilities. Similarly, the number of possible spare cores is not sufficiently investigated.

Altogether, we argue that there is a lack of an in-depth analysis of how load distribution and spare usage policies affect the system lifetime, with respect to aging mechanisms, possibly supported by a flexible experimental framework. The contribution of this paper is to accurately analyse the effects on reliability of a set of classical policies by systematically varying the related parameters, such as the number of spare cores to be selected, in the whole value space. The presented analysis has been carried out by means of the framework for lifetime reliability analysis of multicore systems previously presented in [8]. The reported experimental results show that there is no silver bullet for all scenarios but at the opposite the peculiarities of both the architecture and the workload have to be taken into account in the selection of the most proper runtime policy.

The paper is organized as follows. System, load and reliability models are presented in the next section. Sections III and IV introduce the experimental framework and the adopted setup. A detailed discussion of the outcomes of the experimental campaigns is presented in Section V, while additional remarks are highlighted in Section VI. Section VII closes the paper presenting future work.

II. Preliminaries

A. System Model

The system model adoped in this work is similar to the ones considered in the recent literature [2], [9], [8]. The architecture is a homogeneous multi-core platform composed of n identical cores c_i with a rectangular shape and organized in a 2D mesh-grid floorplan. Each core can be either *active* or *spare* depending on the fact that the core is switched on or power-gated. Moreover, *active* cores may be *working*, when executing some elaborations, or *idle*, when waiting for new elaborations to be executed. For each core, the classical power model, consisting of static and dynamic contributions, is adopted:

$$P_i = P_i^s + P_i^d \tag{1}$$

In particular, $P_i^s > 0$ when the core is switched on (either active or idle), otherwise $P_i^s = 0$. $P_i^d > 0$ only when the core is active, and the actual value depends on the currently executed elaborations. Finally, the temperature of the various cores T_i is computed according to the commonly-adopted model [10] based on the equivalent circuit of thermal resistances and capacitances, able to capture the contributions of both self-activity, related to the actual power consumption of each core, and the heating of the adjacent cores. In particular, due to the fact that reliability is mainly affected by the long-term variations of the temperature, a steady-state temperature model is considered by not taking into account transients.

The system executes a workload composed of a set of applications. Each application a_x is single-threaded and processes input data in a sequence of chunks. More precisely, the application is modeled as a set of jobs $j_{x,y}$, each one characterized by an arrival time, a duration and a dynamic power consumption. In particular, arrival times $t_{x,y}$ and durations $d_{x,y}$ of the jobs of a single application are defined as a stochastic process characterized by an arrival rate γ_x and service rate μ_x. Moreover, given the homogeneous nature of the architecture, the dynamic power consumption value for each core c_i is only due to the specific elaborations; therefore, P_i^d is annotated to the specific job $j_{x,y}$, namely $P^d_{x,y,i}$. Finally, the system workload associated with application a_x is defined as $\rho_x = \frac{\gamma_x}{\mu_x}$ and the overall system load is the sum of the various contributions $\rho_s = \sum \rho_x$ (or $\%\rho_s = \frac{\rho_s}{n}$). It is worth noting that the workload execution request can be satisfied only if the overall ρ_s value is lower than the number of non-spare cores within the system. Figure 1 recaps the adopted system/workload characterization.

The execution model assumes the system to have a FIFO queue for storing new job requests arriving from the various applications, and a central controller dispatching the jobs in the queue according to a specific distribution policy (that is the focus of the proposed analysis) based on the availability of idle processing cores. The execution of a single job is assumed not to be preemptive.

System cores: c_i with $i \in [1, n]$; static power P_i^s; system load ρ_s
Applications: a_x with $x \in [1, m]$; associated system load ρ_x
Jobs in application a_x: $j_{x,y}$ with $y \in [1, p]$; arrival rate γ_x; service rate μ_x
Job $j_{x,y}$: arrival time $t_{x,y}$, duration $d_{x,y}$, dynamic power when executed on core c_i $P^d_{x,y,i}$

Fig. 1. System/workload characterization.

B. Reliability Model

The lifetime reliability of a system $R(t)$ is defined as the probability of a system to be operational until time t. When considering a single core c_i the lifetime reliability is modeled according to the classical Weibull distribution:

$$R_i(t) = e^{-\left(\frac{t}{\alpha_i}\right)^\beta} \tag{2}$$

where t is the current instant of time, α_i the scale parameter, or aging rate, and β the Weibull slope parameter. The α_i parameter is usually function of the operating conditions of the core, generally in terms of temperature, current, voltage, and power consumption. The actual formulation depends on the considered wear-out mechanisms, such as the EM, TDDB, or TC, that can be also considered together according to Sum Of Failure Rate (SOFR) approach.

Equation 2 assumes the core to be in a steady-state situation for the overall operational life; as a consequence, the α_i parameter is a constant. However, in a real-life scenario, the system experiences frequent changes in the operating conditions due to the evolving workload; therefore, Equation 2 needs to be extended as:

$$R_i(t) = e^{-\left(\sum \frac{\tau_h}{\alpha_h}\right)^\beta} \tag{3}$$

being τ_h the duration of each period of time with a steady-state operating condition in the core up to time t (i.e., τ_h periods are not overlapping and $t = \sum \tau_h$). From the above equation, it is also possible to derive the following formula to compute the average aging rate up to t, as:

$$\alpha_i = \frac{\sum \tau_h}{\sum \frac{\tau_h}{\alpha_h}} \tag{4}$$

To model the $R_s(t)$ function for a complex multi-core system, it is necessary to consider not only that each core may have variable operating conditions related to the workload distribution, but also that cores fail, thus requiring the redistribution of the activities on the remaining elements when such an event occurs. In particular, during its operational life the system may experience multiple subsequent core failures. f is defined as the minimum number of core failures such that the system cannot satisfy the workload request anymore. As shown in [8], the various sequences of failures may be represented in a tree structure. Therefore, the reliability model of a multi-core system is computed by the contributions of the cores by means of the formula of the total probability, and the one of the conditioned probability [8]. The former is used to decompose the $R_s(t)$ into various contributions, each one representing the status of the system after a given sequence of failures:

$$R_s(t) = R_s^{\#}(t) + R_s^{1}(t) + R_s^{1:2}(t) + \cdots + R_s^{1:2:\cdots:k}(t) \tag{5}$$

where $R_s^{\#}(t)$ corresponds to the situation when all cores are healthy, $R_s^1(t)$ when core c_1 has failed, $R_s^{1:2}(t)$ when core

c_1 failed, followed by core c_2, and so on. While $R_s^{\#}(t)$ can be computed as a series, i.e. as a product of the cores' reliabilities ($R_s^{\#}(t) = \prod R_i(t)$), to compute the other factors it is necessary to consider that each failure occurs in any instant of time before t. Therefore the conditioned probability formula needs to be used as show in the following, by computing the reliability of the system after core c_1 only has failed:

$$R_s^1(t) = \int_0^{\infty} f_1(t_1) \cdot \prod_{i=1, i\neq 1}^{n} R_i(t|\mathbf{t_1^1}) dt_1 \tag{6}$$

where $f_1(t)$ is the probability density function of $R_1(t)$, i.e., the probability that the failure occurred on core c_1 at a specific time t_1, and $R_i(t|\mathbf{t_1^1})$ is the conditioned reliability function of core c_i knowing that core c_1 failed at time t_1. For further details on the definition of the conditioned reliability function, please refer to [11]. Then, the contribution of the other operating scenarios reported in Equation 5 can be similarly computed by means of a f-dimensional integral, where f is the number of failures considered in the specific scenario.

Finally, the average lifetime of the system is estimated in terms of its Mean Time To Failure (MTTF), computed as the area underlying the specific $R(t)$ of the system:

$$MTTF = \int_0^{\infty} R(t) dt. \tag{7}$$

In a similar way, when considering that the system can tolerate up to f failures, it is also possible to define the Mean Time To the k-th Failure (MTTkF) where $k \in [1, f]$.

III. The Experimental Framework

An experimental framework has been defined by adapting and integrating a set of open source tools. The main module is Caliper [8], a tool based on Monte Carlo simulations for the estimation of the lifetime reliability of a complex multi-core system. In particular, Caliper has been integrated with an in-house event-based SystemC simulator, called `hpc_esim`, modeling the described multi-core system and supporting the specification of load distribution policies.

The simulator takes as input the specification of the architecture, of the workload and the load distribution policy, and computes the average α_i value for each core, representing the aging trend during the performed simulation. The architecture's area and power are characterized by means of McPAT [12], while the floorplan is manually computed by the user. Moreover, `hpc_esim` integrates Hotspot [10] to compute the steady-state temperature traces, and the aging models defined in [1]. The workload is generated in a preprocessing phase on the basis of the specified system load and of the chosen stochastic models.

Within each Monte Carlo run, Caliper calls `hpc_esim` to perform a characteristic simulation of the system w.r.t. the specified architecture, workload and load distribution policy. After the this simulation, Caliper 1) updates the reliability functions of the cores $R_i(t)$ by using the α_i value computed by `hpc_esim`, 2) randomly selects the next core to fail and the time of the failure by using the $R_i(t)$ functions, and 3) updates the status of the architecture to perform a new characteristic simulation. The execution ends when the remaining healthy cores cannot satisfy anymore the specified load, corresponding to the system failure. At the end of all Monte Carlo runs, Caliper aggregates the obtained times of system failures to build the global $R_s(t)$ and to compute the associated MTTF.

IV. The Experimental Setup

For the technological characterization of the architecture we borrowed parameters presented in [9]; in particular, we considered an Alpha 21264 processor synthesized at $45nm$, obtaining an area equal to $2.31 \times 2.31mm$. We considered a 3x3 architecture and a 4x4 one. An average P^d of $15W$ and $9W$ has been considered for the two architectures respectively to get a reasonable Thermal Design Power (TDP), while P^s is $0.14W$. For Hotspot default parameters have been used.

Different stochastic models have be considered to characterize the workload; in particular, for the arrival rates, we used i) a Poisson distribution, and ii) a periodic distribution characterized by a uniformly-distributed fluctuation. The former is classically used to represent the requests to a server's service, while the latter can be adopted to represent streaming applications [13]. Average job duration and related dynamic power consumption have been variated according to a uniformly-distributed fluctuation, to model the fact that job executions of the same application may vary according to the input data. We defined several workloads to stimulate the architecture with different stress intensities. For sake of space but without loss of generality, we report here only the two boundary ones, namely `Light` and `Heavy`, having $\%\rho_s$ equal to 33.3% and 66.6%, respectively, and for each of them we generated the job trace by using both distributions.

Electromigration (EM) has been considered as aging mechanism; the α is modeled according to Black's equation:

$$\alpha_{EM} = \frac{A_0 (J - J_{\text{crit}})^{-n} e^{\frac{E_a}{kT}}}{\Gamma\left(1 + \frac{1}{\beta}\right)} \tag{8}$$

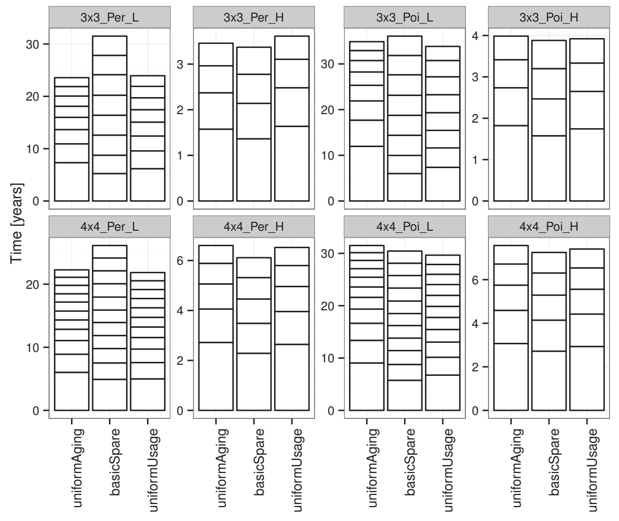
where A_0 is a process-dependent constant, J is the current density, J_{crit} is the critical current density for the EM effect to be triggered, E_a is the activation energy for EM (a constant value), k is the Boltzmann's constant, n is a material-dependent constant, and Γ is the gamma function. In our experiments, we borrowed parameter values from [8].

V. Experimental Results Discussion

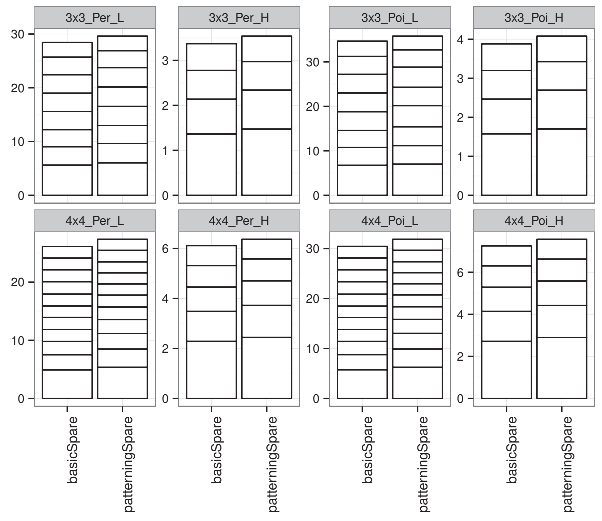
A. Preliminary Analysis

The first campaign aims at comparing the lifetime reliability achieved by some state-of-the-art load distribution policies for DRM, whose goal is to select the core where to map the new job $j_{x,y}$ to be executed, namely:

- `uniformAging`: uses all available cores and balances the cores' average aging factor α_i, by selecting the core with the maximum α_i value.
- `basicSpare`: uses a subset of the available cores and keeps the others as spare. The number of active cores



(a) First campaign: load distribution policies



(b) Second campaign: spare usage

Fig. 2. Comparative MTTkF and lifetime with respect to different core policies/usages.

is the minimum number needed to satisfy the workload. The policy selects a group of contiguous cores using a sequential order, starting from a corner of the grid.

- `uniformUsage`: that uses all the available cores and that balances their utilization, i.e. the ratio between the time the core is active and the total execution time. Balancing is obtained by selecting the core with the minimum utilization value.

Figures 2(a) presents the results achieved by the three policies for each architecture and workload pair. In particular each chart presents the stacked bars of the MTTkF experienced by the architecture until the complete system failure. In the 3x3 architecture, the system survives 7 core failures for `Light` workloads and 3 core failures for `Heavy` ones, respectively. In the 4x4 architecture, these values reach 10 and 4 core failures, respectively.

When the focus is on the last core failure (i.e., on the overall system failure), a first consideration is that balancing the aging factor or the utilization does not lead to significant differences (2.69% on average and 5.87% at most in experiment `4x4_Poisson_Light`). Therefore, we may conclude that – on the long term – when all cores are active the effect of the self-activity on the temperature, and consequently on the aging factor, is predominant w.r.t. the heating caused by the adjacent cores. `uniformAging` is (minimally) beneficial only for the `Poisson` workloads (3.23% on average and up to 5.86% in experiment `4x4_Poisson_Light`, but `basicSpare` in experiment `3x3_Poisson_Light` behaves even 3.50% better than `uniformUsage`). In conclusion, the system lifetime is practically the same for the two policies.

On the other hand, for `Periodic_Light` workloads, `basicSpare` achieves a significantly longer lifetime than the other two policies (25.56% on average up to 33.69% w.r.t. `uniformAging` in experiment `3x3_Periodic_Light`). In fact, the activity is concentrated in restricted regions thus saving a large number of cores from thermal interactions. At the opposite, for the `Periodic_Heavy` workloads, for which the minimum required number of active cores is high, the differences between the spare-based policy and the balancing policies are leveled out (5.74% on average up to 7.42% w.r.t. `uniformAging` for `4x4_Periodic_Heavy`) since the heating is almost spread over all the cores.

Finally, when analyzing intermediate MTTkF, it is possible to observe that the MTT01F represents only a minor contribution to the system lifetime, thus it is quite limiting to consider only such an event for the overall reliability estimation as considered in some works (e.g. [7]). In addition, it is worth noting that when considering only MTT01F, some policies may appear to achieve better results than others while this does not hold for the subsequent sequences of failures, as shown in `3x3_Periodic_Light` and `3x3_Poisson_Light` experiments. Moreover, the general intuition of balancing the aging rate or the utilization to maximize the system reliability holds only for the first few failures. In fact, due to the stress distribution among all the cores, both `uniformAging` and `uniformUsage` are able to postpone the first core failures at the cost of a higher degradation of all the remaining cores; thus the last core failures are very close in time. At the opposite, `basicSpare` concentrates the stress on a subset of cores thus accelerating their aging, while leaving spare cores *younger*. For this reason, all core failures (except the first one) are almost uniformly distributed in time.

A last consideration that may be drawn is related to the bigger difference in the results among the various policies for the `Periodic` workload with respect to the `Poisson` one. This is mainly due to the fact that the jobs in the former case are more uniformly distributed over time. As a result, temperature profiles have a reduced variability. On the other hand, jobs in the `Poisson` workload may be concentrated in bursts thus causing temperature profiles to exhibit higher peaks and valleys, leading to an accelerated aging.

B. Analysis of the effect of the active cores selection strategy

In a second experimental session, we investigate if the policy for the selection of the active cores may affect the system lifetime. We compare the previously presented `basicSpare` policy, with a new one, dubbed `patterningSpare`, that uses the same number of spare cores as `basicSpare`, but selects the active ones by means of a *patterning strategy* so that the geometric distance among them is maximized. This policy was inspired by the thermal-related considerations drawn in [9].

Results of this experiment are reported in Figure 2(b). The evidence is that `patterningSpare` behaves better than `basicSpare` in all the situations (the average lifetime improvement is 4.77% up to 7.04% in the `4x4_Periodic_Heavy` experiment). In fact, by maximizing the distance among the active cores, the patterning strategy allows to globally reduce the temperature on the chip. Moreover, it is worth noting that the two policies exhibit a similar trend in the time distribution of the MTTkF.

C. Analysis of the number of spare cores

In a third campaign, we investigate how the number of spare cores affects the lifetime reliability of the system. In particular, we consider the `patterningSpare` policy and vary the number of active cores from the total number of available cores to the minimum number of cores required by the load to be satisfied. Moreover, due to the long analysis time, we performed a coarser-grained analysis on the `4x4` architecture by varying the number of spare cores of 2 units per time.

From the results shown in Figure 3, it is possible to state that when the workload is `Light` having the largest possible number of spare cores allows to extend the system lifetime. More precisely, when the workload is `Periodic_Light` the improvement of having the largest possible number of spare cores w.r.t. not having spare cores is 32.24% in the `3x3` scenario and 23% in the `4x4` one. More limited benefits can be seen for the `Poisson_Light` workload (up to 6.09% in `3x3` scenario). On the other hand, when the workload is `Heavy`, varying the numbers of spare cores does not bring any significant difference in terms of system reliability. This is due to the fact that the `Heavy` workload requires a large number of active cores; therefore, there are very few spare cores surrounded by many active ones, causing an overall heating effect.

D. Analysis of the active cores usage strategy

A final campaign analyzes various possible strategies for using the active cores. In particular we compare the idea of using the active cores in a fixed sequential order for the job mapping (the strategy used in the `basicSpare` and in the `patteringSpare`) against a new strategy aimed at balancing the usage of the active cores (namely, `unformBasicSpare` and `uniformPatteringSpare`). Actually, the experimental results, that are not reported for sake of space, show that there is no relevant lifetime difference between the two approaches.

E. Analysis of the intermediate MTTkFs

The intuition for this last analysis is the evidence reported in Figure 2(a) where `uniformUsage`, which does not use spares, is able to obtain longer MTTkFs for the first failures than `basicSpare`, while for the subsequent MTTkFs the situation is reversed. Therefore, we compare intermediate MTTkF for the `patterningSpare` (the most efficient policy in the previous experiments) by considering the minimum number of active cores satisfying the workload (i.e. maximum number of spare cores) against zero spare cores. To better show the trend of the time distribution of subsequent MTTkFs, we report in Figure 4 the sequence of MTTTkFs. It may be observed that in almost all cases there is a trend inversion in favor of using spares when considering a larger number of MTTkFs. This is particularly true for `Periodic_Light` workloads, while for `Poisson_Light` the inversion can be observed only for the last MTTTkF.

VI. Final Remarks

The systematic experimental campaign we have carried out allowed us to draw the following remarks for multi-core system architects interested in DRM:

- In most cases, homogenizing the cores aging factor or utilization leads to the same mean-time-to-failure; thus it is preferable to adopt the simpler utilization balancing policy than the more complex aging factor balancing one.
- In case the load on the system is light, it is better to dispatch it on the minimum number of cores and to leave as much cores as possible as spares.
- When adopting a spare cores-based dispatching policy, it is preferable to chose the active cores in a "clever" way, e.g., maximizing the geometric distance among the cores, allows to reduce the overall chip temperature and thus to achieve longer mean-time-to-failure.
- If the load on the system strongly and periodically varies, having a dynamic dispatching policy that is able to adapt its behavior to the load itself may help in extending the lifetime of the system.

VII. Future Work

We plan to further investigate issues and aspects related to DRM in multi-core systems. In particular, while this work focuses on mapping policies, we aim at analyzing the impact of scheduling policies, that may give the possibility to further slow down aging phenomena by distributing jobs in time.

Furthermore, in real life scenarios, workload intensity may vary in time. Therefore we aim at investigating hybrid policies able to adapt to the characteristics of the workload as it changes in time.

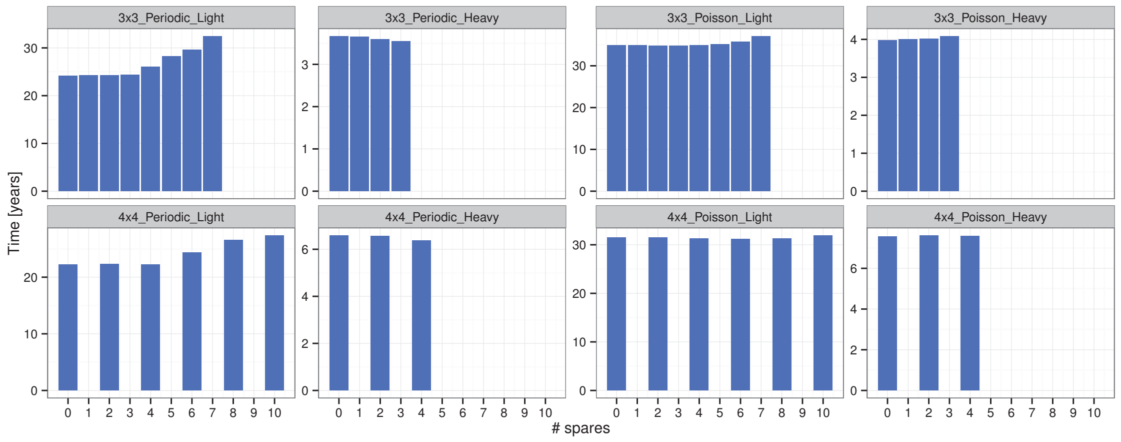


Fig. 3. Third campaign: exploration of the number of spare cores with `patterningSpare`.

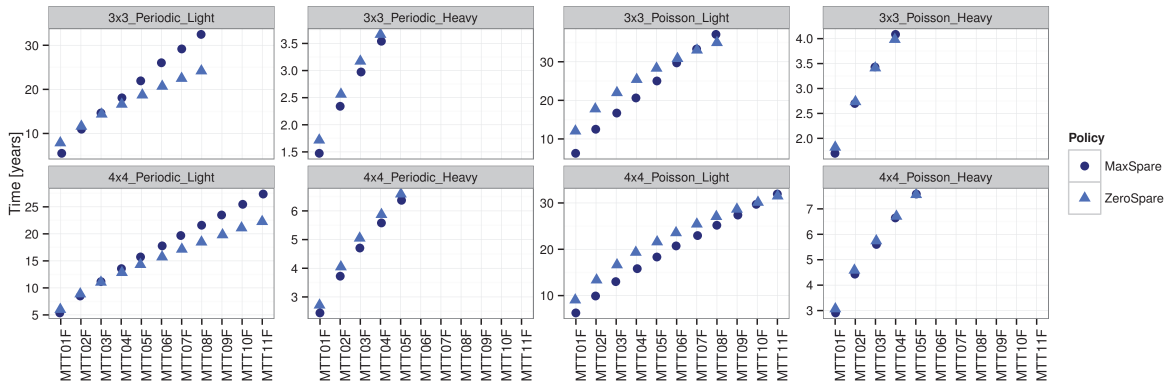


Fig. 4. Fourth experiment: comparison or MTTkF between `patterningSpare` with the maximum number of spare cores and zero spare cores.

Acknowledgements

This work has been partially supported by the European Union Seventh Framework Programme under grant agreement no. 610996 (SAVE).

References

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

[2] L. Huang and Q. Xu, "Characterizing the lifetime reliability of manycore processors with core-level redundancy," in *Proc. Int. Conf. on Computer-Aided Design*, 2010, pp. 680–685.

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

[4] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 1, pp. 169–180, Jun 2009.

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

[6] 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. Int. Symp. DFT*, 2013, pp. 58–64.

[7] A. Simevski, R. Kraemer, and M. Krstic, "Increasing multiprocessor lifetime by Youngest-First Round-Robin core gating patterns," in *Proc. NASA/ESA Conf. Adaptive Hardware and Systems*, 2014, pp. 233–239.

[8] C. Bolchini, M. Carminati, M. Gribaudo, and A. Miele, "A lightweight and open-source framework for the lifetime estimation of multicore systems," in *Proc. Int. Conf. on Computer Design*, 2014, pp. 166–172.

[9] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power - Efficient power budgeting for many-core systems in dark silicon," in *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis*, 2014, pp. 1–10.

[10] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

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

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *ACM Trans. on Architecture and Code Optimization*, vol. 10, no. 1, pp. 5:1–5:29, Apr 2013.

[13] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. on Networking*, vol. 6, no. 5, pp. 611–624, Oct 1998.