Thermal-Cycling-aware Dynamic Reliability Management in Many-Core System-on-Chip

Mohammad-Hashem Haghbayan[1], Antonio Miele[2], Zhuo Zou[3], Hannu Tenhunen[1], Juha Plosila[1]
[1]*Department of Future Technologies – University of Turku – Finland*
[2]*Dip. Elettronica, Informazione e Bioingegneria – Politecnico di Milano – Italy*
[3]*Fudan University – China*
{*mohhag, hannu.tenhunen, juplos*}*@utu.fi, antonio.miele@polimi.it, zhuo@fudan.edu.cn*

***Abstract*—Dynamic Reliability Management (DRM) is a common approach to mitigate aging and wear-out effects in multi-/many-core systems. State-of-the-art DRM approaches apply fine-grained control on resource management to increase/balance the chip reliability while considering other system constraints, e.g., performance, and power budget. Such approaches, acting on various knobs such as workload mapping and scheduling, Dynamic Voltage/Frequency Scaling (DVFS) and Per-Core Power Gating (PCPG), demonstrated to work properly with the various aging mechanisms, such as electromigration, and Negative-Bias Temperature Instability (NBTI). However, we claim that they do not suffice for thermal cycling. Thus, we here propose a novel thermal-cycling-aware DRM approach for shared-memory many-core systems running multi-threaded applications. The approach applies a fine-grained control capable at reducing both temperature levels and variations. The experimental evaluations demonstrated that the proposed approach is able to achieve 39% longer lifetime than past approaches.**

***Index Terms*—Lifetime reliability, Thermal cycling, resource management**

I. Introduction

The aggressive technological scaling of the last decade has allowed the integration of hundreds of cores in the same chip, thus leading to the design of many-core systems. The counterpart of such a progress has been an increase of the power densities and consequent heating within the device. ITRS reports [17] show that this trend has caused an acceleration of the device aging and wear-out. Aging mechanisms (e.g. electromigration, NBTI and thermal cycling) may cause delay errors and, eventually, device breakdowns [4]. Lifetime reliability models show device's Mean Time To Failure (MTTF) to have an exponential relationship with the temperature; a 10–15° increase may halve the expected lifetime [10].

DRM has been a widely investigated approach to tackle such an issue in multi-/many-core systems (e.g. [1], [9], [22]). Properly controlling at system-level the activity of a system executing a given workload allows to reduce the operating temperatures and, thus, the aging process. A feedback loop is generally exploited to monitor the aging status of the various architecture's cores and consequently take decision on the system behavior. Since aging sensors are not generally integrated in the devices, the common solution is to adopt stochastic reliability models [4] based on per-core temperature sensing. On the other hand, control decisions are actuated by tuning various application-level knobs, such as the mapping and scheduling, and architecture-level ones, such as DVFS and PCPG while the reliability is co-optimized in conjunction to other system-level requirements, such as the workload performance and power consumption.

Taking all these elements under control is an extremely complex task. We noticed that most of the previous approaches [1], [6], [7], [9], [15], [22] focus on aging mechanisms but thermal cycling. Thermal cycling is the only mechanism not only depending on temperature levels but also on the amplitude and the frequency of the temperature variations [4]; as a result thermal cycling is a critical reliability issue [2]. The few works addressing such an aging effect [2], [12] consider only the output of the reliability model as the driver of the control decisions; as we will show in this paper, this is an indirect feedback that does not suffice for lifetime extension purposes.

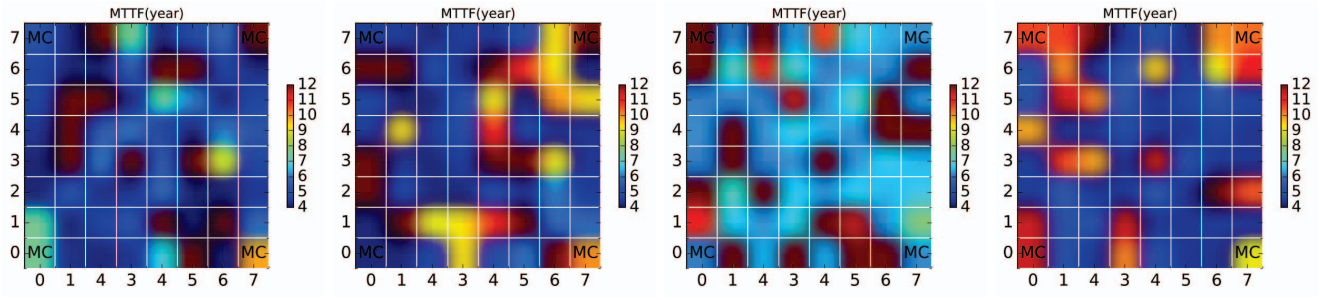
Given these motivations, we here propose a novel thermal-cycling-aware runtime resource management approach for many-core architectures. The novelty of the approach is a fine-grained direct control on both temperature levels and variations and long-term monitoring of aging thus capable at sensibly reducing the effects of thermal cycling w.r.t. the past approaches while guaranteeing same performance and not violating the power budget. The approach is based on state-of-the-art algorithms for mapping, scheduling and power management, each one of them enhanced with thermal cycling awareness. An experimental evaluation shows that the approach achieves 54% longer lifetime than the reliability-agnostic counterpart and 39% than the state-of-the-art techniques which address only in a partial way the considered issue.

The rest of the paper is organized as follows. Next section introduces the considered system architecture and reliability model and later presents a motivating example, while Section III discusses related work. The proposed thermal-cycling-aware resource management approach is described in details in Section IV and is later experimentally evaluated in Section V. Section VI draws conclusions and presents future work.

II. Background and Motivation

System Architecture. In this work we consider a classical many-core architecture integrating a large number of homogeneous cores connected through a Network-on-Chip (NoC) and organized in a mesh-based topology (Fig. 1). The system has a unified shared memory, accessible through a number of

978-3-9819263-4-7/DATE20/©2020 EDAA



(a) Nominal baseline (b) Reliability-aware Scheduling (c) Reliability-aware Mapping (d) Reliability-aware DPM

Fig. 2. The effect of reliability-awareness of the various units of the resource manager on cores' MTTF.

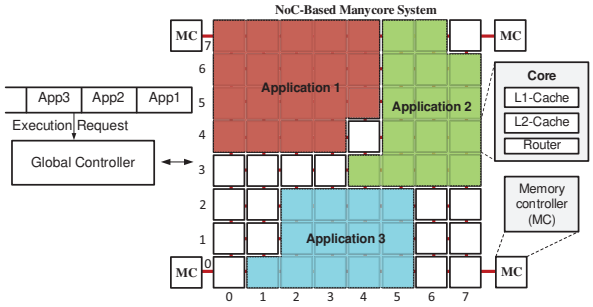


Fig. 1. The target system architecture.

memory controllers connected to the NoC, a shared L2 cache distributed all over the NoC and private L1 cache per each core. The cores are dynamically tunable in terms of DVFS and PCPG. Moreover the architecture exposes a per-system power sensor and per-core temperature sensors. A global controller is in charge of coordinating application's execution and hardware knob tuning. Such an architecture is commonly employed for accelerating data-intensive multi-programmed workloads, composed of multi-threaded applications entering and leaving the system with an unpredictable fashion.

Reliability Model. As a common practice in approaches for DRM (e.g. [1], [9], [22]), we here consider the classical stochastic reliability model [4], [21] to estimate the aging effects caused by the heating in the device. The only requirement is the architecture to be provided with per-core temperature sensors. In particularly, since we here consider Thermal Cycling, the expected lifetime of a single core working in a steady state situation is estimated in terms of the number of cycle to the failure by means of the Coffin-Mason equation:

$$N_{TC} = A_{TC}\left(\delta T - T_{th}\right)^{(-b)} e^{\frac{E_{a_{TC}}}{kT_{Max}}} \quad (1)$$

where A_{TC} is an empirically determined fitting constant, δT is the thermal cycle amplitude, 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 thermal cycling, and T_{Max} is the maximum temperature during the cycle. To consider a more realistic situation where various thermal cycles may occur with different amplitudes and maximum temperature, it is necessary first to extract the list of the cycles from a run characterizing the system activity, by means of the rainflow counting algorithm, and later, N_i of the various m identified cycles can be aggregated in an average value by means of the Miner's rule (as discussed in [21]):

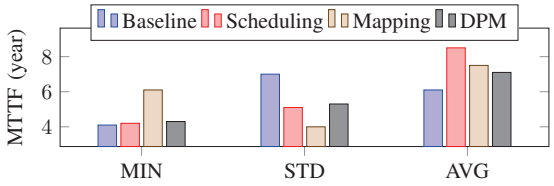


Fig. 3. Analysis of the results in Fig. 2: Minimum (MIN), standard deviation (STD), and average (AVG) MTTF values.

$$N_{TC\_avg} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_i}} \quad (2)$$

It should be noted that the formula assumes the cycles to be almost all of the same duration. To compute a weighted average, the constant term 1 should be replaced with the proper weight. Finally, the thermal cycling MTTF is calculated as:

$$MTTF_{TC} = \frac{N_{TC\_avg} \cdot \Delta T}{m} \quad (3)$$

where ΔT is the overall duration of the characteristic run.

Motivating Example Let's assume that the target platform is a 8×8 many-core as shown in Fig. 1 where applications enter and leave the system at run-time. We analyze the effect of thermal cycling awareness of each part of the resource management units on the system's overall MTTF. Fig. 2 and 3 compare fine-grained and overall MTTF of the various cores after certain time of system's activity and while considering thermal cycling in different resource management units. For each resource management unit, reliability is improved by deliberately considering the specific factors that have high contribution on thermal cycling w.r.t. that unit. For example the short-term history of thermal cycling frequency and average temperature is fed to Dynamic Power Management (DPM) unit to adopt it to be thermal cycling aware, and, mixture of long-term profiled MTTF and short-term temperature fluctuation is used for reliability-aware mapping unit. The overall results in Fig. 3 shows that by considering relevant factors in each management unit, we can partially improve the minimum, average, and balance of the MTTF separately. In this paper, we propose different separate techniques to consider the thermal cycling in different resource management units, i.e., mapping,

scheduling, and DPM. Moreover, we show how putting all the techniques together can lead to the best outcome w.r.t. the single individual techniques and w.r.t. state-of-the-art approaches.

III. Related Work

First works on DRM [10], [19] considered single-core systems and acted on architectural knobs such as DVFS to prolong MTTF. Indeed, they consider a simplistic reliability model. Later, various approaches focused on shared-memory multi-core systems [3], [11] thus acting on a larger set of knobs comprising task scheduling and, the second one also PCPG to satisfy a power budget. Unfortunately, they consider thermal cycling with a simplistic reliability model or neglect it.

When considering, NoC-based many-cores architectures, the picture becomes much more complex. In fact, applications frequently consist of multiple concurrent threads or a task-graph of pipelined tasks. Therefore, several approaches (e.g. [1], [6], [7], [9], [15], [22]) defined an advanced reliability-driven task mapping balancing performance and lifetime reliability. Among the approaches dealing with task-graph applications (e.g., [1], [9], [22]), one of the most comprehensive proposals is presented in [7], where the run-time policy concurrently maps incoming application task-graphs and acts on DVFS and PCPG to optimize performance while satisfying both the power constraints due to the dark silicon and the given lifetime target. In the scenario of shared-memory many-cores suffering dark silicon issues, it is worth mention the approach in [6] where aging status prediction is used to estimate the effects of a possible threads mapping decision. The technique is specifically tailed for NBTI. Finally, a similar approach using machine learning in the management policy is presented in [16]. Unfortunately, none of them considers thermal cycling. Moreover they cannot be adapted in a straightforward way to consider such an aging mechanism since it requires both temperature level and variations to be controlled.

The only works considering thermal cycling in the multi-/many-core scenario have been proposed in [2], [12]. In [2] the authors defined a run-time mapping policy for single-threaded applications aimed at optimizing various lifetime w.r.t several aging mechanisms including thermal cycling. The idea is to use the youngest core w.r.t. the various aging models. Unfortunately, as shown later in the experimental results, this does not suffice with thermal cycling, since it is necessary a more direct control on the amplitude and the frequency of the temperature variations. Nonetheless, the approach considers a quite simplified scenario where only task mapping is addressed. Finally, the approach in [12] considers almost the same simplified scenario. In particular, the variance in resource utilization is minimized so that indirectly temperature levels and variations are minimized. In conclusion, such proposals are weak in tackling thermal cycling and cannot be straightforwardly adapted to the considered scenario.

IV. Proposed Controller Architecture

The overall structure of the proposed reliability-aware run-time resource manager is depicted in Fig. 4. The unit is

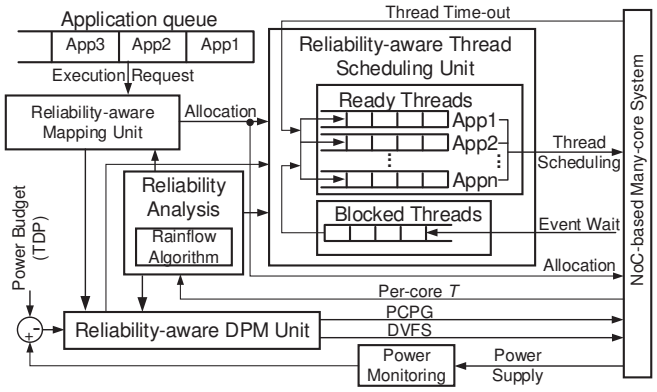


Fig. 4. The runtime thermal-cycling-aware resource manager.

hosted on the global controller and implements a feedback control loop with the many-core system. We borrowed a pretty standard internal organization (as in [14]) including an application mapping unit, a thread scheduling unit and a DPM unit. For such units state-of-the-art approaches has been employed and enhanced to consider reliability-related data in their decisions. Finally, a novel Reliability Analysis unit has been added to compute the aging status of the system to feed the other modules. The modules are discussed in the following.

Reliability Analysis unit. This unit analyzes the aging status of the various cores in the many-core system. It takes as input the temperature measures of each core in the system, gathered from the available sensors with a fixed sampling period (approximately 1 second). The unit first applies a low-pass filter to remove high-frequency oscillations in the temperature trace of each core, and then applies the rainflow count algorithm to extract the thermal cycles [21]. Each cycle is described in terms of the peak and valley temperatures. Both the two algorithms are applied on-line on the incoming values.

With a given long-term control period (lasting 1-2 hours), the unit analyzes the list of cycles detected for each core and computes various reliability-aware metrics, in particular $MTTF_{TC}$, based on the current activity from the beginning of the operational life. This value is obtained by means of the steps shown in Algorithm 1, which applies on each core the model discussed in Section II. Finally the unit identifies a list of critical cores, having a $MTTF_{TC}$ lower than a

Algorithm 1 Reliability Analysis of a single core

Inputs: $\mathbf{T}$: vector of temperature trace of a core for current control epoch t
t: number of epochs from the beginning
$N_{TC\_overall\_(t-1)}$: overall N_{TC} at the previous epoch $t-1$
Outputs: $MTTF_{TC}$: MTTF given the overall history
$N_{TC_overall\_t}$: overall N_{TC} at the current epoch t
Body:

1: $\mathbf{T_{low}} \leftarrow$ low_pass_filter($\mathbf{T}$);
2: $\mathbf{cycles} \leftarrow$ rainflow_count($\mathbf{T_{low}}$)
3: $\mathbf{N_i} \leftarrow$ coffin_masson($\mathbf{cycles}$)
4: $N_{TC\_avg\_i} \leftarrow$ miner_rule($\mathbf{N_i}$)
5: $N_{TC\_overall\_t} \leftarrow \frac{t}{\frac{1}{N_{TC\_avg\_i}} + \frac{t-1}{N_{TC\_overall\_(t-1)}}}$
6: $MTTF_{TC} \leftarrow$ compute_MTTF($N_{TC\_overall\_t}$)

given percentage threshold w.r.t. the average cores' status, thus requiring to be put in power-gating mode to slowdown the aging trend. Once, the metrics are computed, the list is emptied to proceed with the subsequent long-term period.

Reliability-aware Mapping unit. This unit is in charge at selecting a set of cores to be allocated for the execution of a newly incoming application. Allocation is here performed in a mutually-exclusive way; each single core can be at most assigned to a single application. The manager has a queue of applications to be executed. Thus, the unit is awoken when a new application enters the queue or a running application leaves the system and the queue is not empty.

The mapping policy selects the most suitable smallest squared region, if any available, capable at hosting the number of threads spawn by the application based on a reliability-aware affinity metric RAF. The affinity metric, assigned to the central node of the region, has to consider four different issues: 1) the application's threads have to be concentrated in the closest region possible to optimize communication overheads. 2) memory-intensive applications have to be mapped near to the memory controllers while other applications may be mapped farther. 3) it is necessary to reduce the overall aging. 4) it is necessary to reduce temperature variations. The former two issues are related to the nominal mapping, i.e. the considered baseline; while the two latter ones are related to thermal cycling, i.e. the core of our proposal. Thus, the affinity metric RAF has been defined by combining these four factors.

The defined mapping approach is an extension of a state-of-the-art policy for multi-task application mapping, called MapPro [8]. MapPro solves the first issue by defining an affinity metric called *vacancy factor*. Given the central node c with coordinates w, h identifying a square region with radius r, the metric is defined as

$$VF_c = \sum_{i=i-r}^{i+r} \sum_{j=j-r}^{j+r} I_{i,j} \times (r - d + 1) \tag{4}$$

being $I_{i,j} = 1$ when the core i, j is idle, otherwise 0, and d is the Mahanattan distance of the same core to the central node. It is worth mentioning that $I_{i,j} = 0$ also for cores in the critical list returned by the Reliability Analysis unit. If VF_c is 0, then the region cannot host the new application. The central node maximizing the metric represents the best candidate since it has the maximum number of idle nodes closer to itself.

The second issue is solved by means of another metric MF_c, borrowed from [5], which characterizes the affinity of each region to the type of application in terms of memory accesses (hits and misses), profiled at design time. Since the affinity is computed per single core, MF_c is defined for a region as the minimum value among the involved cores.

The reliability-related aspects are considered by computing two additional affinity metrics. The former, characterizing the aging AF_c, is computed from the cores' $MTTF_{TC}$. Each central node is assigned with the maximum value in the square region. In some extend, this factor is an extension of what the past approaches do in the single-thread scenario [2]. Finally, the last metric TF_c is obtained by analyzing the average temperature values returned by the Reliability Analysis unit to compute the average temperature variation in the pool of cores in the square region. Such a metric characterizes the fluctuations of temperature in that area. Selecting the region having the minimum value will reduce the number of thermal cycles, similarly to what they do in [13] with the utilization metric. Thus, choosing the minimum w.r.t. such metric will reduce the temperature variations.

Algorithm 2 Reliability-aware Mapping

Inputs: $appl$: new application
$type_{appl}$: pre-profiled type of the application w.r.t. memory accesses
$\mathbf{arch}$: vector of cores in the architecture
$\mathbf{T}$: vector of temperatures of the core in $mapList$
$\mathbf{MTTF_{TC}}$: vector of MTTF returned by the Reliability Analysis unit
Body:

```
1: r ← (√|appl| − 1)/2;
2: RAF_max ← −∞
3: c_map ← None
4: for all c ∈ arch do
5:     VF_c ← computeVF(c, r)
6:     coreList ← get_idle_cores(arch, c, r)
7:     MF_cores ← get_memory_affinity(coreList, type_appl)
8:     MF_c ← min(MF_cores)
9:     AF_c ← min(MTTF_TC[coreList])
10:    T_coreList ← T[coreList]
11:    T_avg ← compute_average(T_coreList)
12:    T_diff ← abs(T_coreList − T_avg) //vectorized operation
13:    TF_c ← avg(T_diff)
14:    if VF_c ≠ 0 then
15:        RAF ← (VF_c·AF_c)/(MF_c·TF_c)
16:        if c_map = None or RAF > RAF_max then
17:            c_map ← c
18:            RAF_max ← RAF
19: if c_map ≠ None then
20:    map(appl, c_map, r)
```

Algorithm 2 shows the steps performed on a newly incoming application. For each core c in the architecture, the four factors are computed (Line 5, Lines 6–8, Line 9, and Lines 10–13, respectively) and then combined in the RAF if the region can host the application (Lines 14–15). The algorithm selects the core c_map with the maximum RAF (Lines 16–18) and if exists, the application is mapped on the related region (Lines 19–20).

Reliability-aware Thread Scheduling unit. The unit schedules the threads of the running applications in the allocated region. Since, the mapping policy partitions the cores among the running applications, the scheduling is executed separately per each application. The unit features a policy based on a round-robin algorithm having the classical ready thread queue and blocked thread list as shown in Fig. 4.

In details, Algorithm 3 describes the scheduling policy that is awoken with a short-term control period (few milliseconds as in a common operating system). Receiving as input from the Reliability Analysis unit the current temperatures, the policy computes the average temperature on the cores allocated for the current application ($\mathbf{mapList}$) and the absolute deviation of the core's temperature w.r.t. the average value (Lines 1–3). Only idle cores are taken from $\mathbf{mapList}$ and the obtained queue is sorted based on the temperature deviation (Lines 4–

Algorithm 3 Reliability-aware Thread Scheduling

Inputs: $\mathbf{mapList}$: vector of cores allocated for the current application
$\mathbf{T}$: vector of temperatures of the core in $mapList$
$\mathbf{readyQueue}$: queue of ready threads for the current application
Body:

1: $\mathbf{T_{mapList}} \leftarrow \mathbf{T}[\mathbf{mapList}]$
2: $T_{avg} \leftarrow$ compute_average($\mathbf{T_{mapList}}$)
3: $\mathbf{T_{diff}} \leftarrow$ abs($\mathbf{T_{mapList}} - T_{avg}$) //vectorized operation
4: $\mathbf{coreQueue} \leftarrow$ sort($\mathbf{mapList}, \mathbf{T_{diff}}$)
5: $\mathbf{coreQueue} \leftarrow$ get_idle($\mathbf{coreQueue}$)
6: **for all** $thread \in \mathbf{readyQueue}$ **do**
7: **if** $\mathbf{coreQueue} \neq \emptyset$ **then**
8: $c \leftarrow$ pop($\mathbf{coreQueue}$)
9: schedule($thread, c$)
10: **else**
11: break

5). This strategy selects cores in a way such that temperature variations in the short term are minimized. Therefore, for each thread in the ready list, the policy selects the first idle core in the sorted $\mathbf{coreQueue}$ and starts its execution for the next scheduling epoch (Lines 6–9). The procedure ends when no more thread needs to be scheduled or $\mathbf{coreQueue}$ is empty (Line 11), thus leaving some threads in the ready queue.

It is worth noting that the mapping and scheduling units one works with two highly-different control periods. The former in a long term, thus it is capable at perceiving the slow variations in the aging metric $MTTF_{TC}$ and to consequently take decisions on that aspect, while the latter one in a short term, thus it cannot. For this reason, the scheduling policy only focuses on the minimization of the temperature variations.

Reliability-aware DPM. This last unit dynamically tunes DVFS of the various cores to satisfy a required power budget, in terms of Thermal Design Power (TDP). We here adopted and extended MOC [14], a multi-objective power management approach. In particular, the approach uses a PID controller in a first stage to identify the necessity of voltage/frequency scaling from the sensed power consumption. Then, an advanced multi-objective control concurrently tunes the knobs of the various cores to provide required computational power while not violating the TDP. We here extend the approach to consider among the optimization metrics also a thermal-cycling-related one, defined for each cores as the product between the temperature peak and amplitude of the last detected cycle: $avgAmplitude \times maxT$. Finally, the unit applies PCPG on each core that is idle or in the critical list returned by the Reliablity Analysis unit.

V. Experimental results

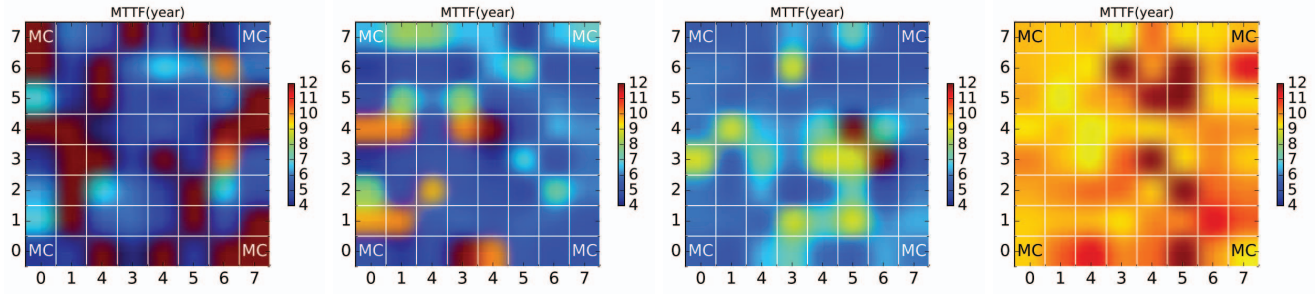
We experimentally evaluated the proposed approach in a simulation environment by using Noculator [5], a shared-memory NoC-based many-core simulator based on Intel PIN binary instrumentation tool. Each core is modeled as a Niagara2 processor with SPARC Instruction Set Architecture. Each core has a private local L1 cache and a shared L2 cache distributed all over the chip. The system has four memory banks each of them connected to the NQC through a separate memory controller. The architecture has been configured as in [5]. Physical scaling parameters and other characteristics such as power modeling and TDP were gathered from McPAT and Lumos [20] and for the steady-state thermal model Hotspot [18] has been integrated in the simulator. For the experiments, we characterized a realistic 8×8 architecture with a squared floorplan, a chip area of $109mm^2$ in $16nm$ technology, and TDP of 90W. Thermal cycling has been modeled as in [21] by characterizing parameters of Equations 1–3 as follows: $E_{a_{TC}} = 0.42$eV, $b = 2.35$, $T_{th} = 1°$C, $k = 8.62 \cdot 10^{-5}$eV/K, and we fitted 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 $m = 10$. The workload executed by the system is composed of multi-threaded PARSEC benchmarks that are randomly selected via a repository and issued into the system in runtime.

We compared our proposed approach with three different past works: 1) the *nominal baseline*, consisting in the basic management strategies we considered ([5], [8], [14]) without any reliability-enhancement, 2) *MOC* [14], a DPM optionally using MTTF in the decision process, 3) the *reliability-aware mapping* defined in [2], which adopts only MTTF to control thermal cycling by selecting the youngest core, applied to our considered nominal baseline. The simulation is lasted until completing 25 application executions randomly selected from the PARSEC multi-threaded suite. Fig. 5 shows a MTTF snapshot of the cores for the proposed approach and the considered references. As it can be seen the proposed approach significantly improves the MTTF in comparison to the other approaches. The main key point for such a significant improvement is to use different features in resource management to relax the negative effect of specific factors contributing on thermal cycling in each unit. Fig. 6 reports some statistics on the obtained results. According to this statistics the proposed approach improves the minimum MTTF of the core in the system by 54%, 51%, and 39% and average MTTF by 40%, 21%, and 20% w.r.t. the nominal baseline, MOC, and reliability-aware mapping, respectively. Moreover, the comparison of standard deviation shows that the proposed approach is able to better balance the MTTF than the state-of-the-art ones by 55%, 43% and 38% improvement of standard deviation, respectively. As a final note, comparing the Fig. 6 and Fig. 3 shows that considering different factors in different resource management units can complement each other for the sake of thermal cycling improvement.

Finally, in Fig. 7 we report the number of completed applications during the time for the proposed approach and reliability agnostic baseline. As it can be seen the trend of completing the applications for the proposed approach is very close to the nominal baseline; this demonstrates the proposed idea does not negatively affect the system performance.

VI. Conclusions

The paper presented a thermal-cycling-aware runtime resource management approach for many-core systems. The approach offers a fine-grained control on temperature variations in mapping, scheduling and power management thus allowing



(a) Nominal baseline (b) MOC [14] (c) Reliability-aware mapping [2] (d) Proposed approach

Fig. 5. Comparison of the proposed approach w.r.t. the past works in terms of cores' MTTF.

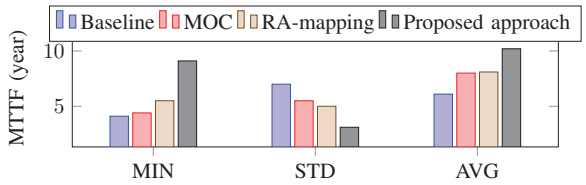


Fig. 6. Analysis of the results in Fig. 2: Minimum (MIN), standard deviation (STD), and average (AVG) MTTF values.

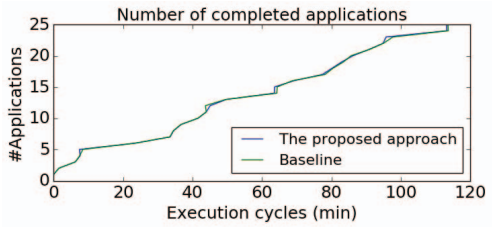


Fig. 7. Performance of the proposed approach w.r.t. the nominal baseline.

to reduce thermal cycling effects. Experimental results have shown the approach to outperform past work with a 39% lifetime improvement. Future work will focus on a more advanced temperature control to handle multiple aging mechanisms.

ACKNOWLEDGMENT

The work has been partially funded by the Academy of Finland project entitled "LARA: Learning and Assessing Risks for Enhancing Dependability of Autonomous Socio-Technical Systems", and by the Shanghai Research and Innovation Functional Platform Program under Grant 17DZ2260900.

REFERENCES

[1] 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)*, pages 58–64, 2013.

[2] 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)*, pages 1373–1378, 2013.

[3] 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*, pages 169–180, 2009.

[4] J. E. D. E. Council. Failure Mechanisms and Models for Silicon Semiconductor Devices. Technical Report JEP122G, Oct. 2011.

[5] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi. Application-to-core Mapping Policies to Reduce Memory Interference in Multi-core Systems. In *Proc. Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 455–456, 2012.

[6] 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)*, pages 1–6, 2015.

[7] 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*, 66(9):1599–1612, 2017.

[8] 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*, pages 1–8, 2015.

[9] A. S. Hartman and D. E. Thomas. Lifetime improvement through runtime wear-based task mapping. In *Proc. Intl. Conf. Hardware/software codesign and system synthesis (CODES)*, pages 13–22, 2012.

[10] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-Mechanism Reliability Modeling and Management in Dynamic Systems. *IEEE Trans. on VLSI Systems*, 16(4):476–487, 2008.

[11] 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)*, pages 13–22, 2012.

[12] 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*, 25(6):1895–1905, June 2017.

[13] Y. Ma, T. Chantem, R. P. Dick, S. Wang, and X. S. Hu. An on-line framework for improving reliability of real-time systems on "big-little" type MPSoCs. In *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*, pages 446–451, 2017.

[14] 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*, 25(2):427–440, Feb 2017.

[15] V. Rathore, V. Chaturvedi, A. K. Singh, T. Srikanthan, R. Rohith, S. Lam, and M. Shaflque. HiMap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In *Proc. Design, Autom. & Test in Europe (DATE)*, pages 991–996, 2018.

[16] 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)*, pages 179:1–179:6, 2019.

[17] Semiconductor Industry Association et al. International Technology Roadmap for Semiconductors. http://www.itrs2.net/, 2011.

[18] 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.*, 1(1):94–125, 2004.

[19] 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)*, pages 276–287, 2004.

[20] L. Wang and K. Skadron. Dark vs. Dim Silicon and Near-Threshold Computing Extended Results. In *University of Virginia Department of Computer Science Technical Report TR-2013-01*, 2012.

[21] 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)*, pages 297–306, 2010.

[22] A. Y. Yamamoto and C. Ababei. Unified reliability estimation and management of NoC based chip multiprocessors. *Microprocessors and Microsystems*, 38(1):53–63, 2014.