Improving System-Level Lifetime Reliability of Multicore Soft Real-Time Systems

Yue Ma, *Student Member, IEEE*, Thidapat Chantem, *Member, IEEE*, Robert P. Dick, *Member, IEEE*, and X. Sharon Hu, *Fellow, IEEE*

***Abstract*—This paper studies the problem of maximizing multicore system lifetime reliability, an important design consideration for many real-time embedded systems. Existing work has investigated the problem but has neglected important failure mechanisms. Furthermore, most existing algorithms are too slow for on-line use, and thus cannot address run-time workload and environment variations. This paper presents an on-line framework that maximizes system lifetime reliability through reliability-aware utilization control. It focuses on homogeneous multicore soft real-time systems. It selectively employs a comprehensive reliability estimation tool to deal with a variety of failure mechanisms at the system level. A model predictive controller adjusts utilization by manipulating core frequencies, thereby reducing temperature and an on-line heuristic adjusts the controller sampling window length to decrease the reliability effects of thermal cycling. Experiments with a real quad-core ARM processor and a simulator demonstrate that the proposed approach improves system mean time to failure by 50% on average and 141% in the best case, compared to existing techniques.**

***Index Terms*—System-level reliability optimization; Dynamic voltage and frequency scaling; Online control; Real-time embedded system**

I. Introduction

MULTICORE systems offer good performance with reasonable power consumption and are widely used in many real-time applications such as automotive electronics, industrial automation, and avionics. Ongoing increases in device densities and operating frequencies increase microprocessor power density and temperature. This increased chip temperature accelerates the aging of devices, resulting in permanent faults and reducing operating lifespans. Many embedded systems, e.g., automotive infotainment, operate in environments with varying temperatures, as well as varying task execution times and task periods. On-line reliability optimization approaches have advantages for such systems.

Manuscript received Aug. 29, 2016; revised Dec. 8, 2016; accepted Jan. 22, 2017. This work was supported in part by NSF under awards CNS-1319904, CNS-1319718, and CNS-1319784 and by Sandia National Laboratory.

Y. Ma and X. S. Hu are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: yma1@nd.edu; shu@nd.edu

T. Chantem is with the Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Arlington, VA 22203 USA (e-mail: tchantem@vt.edu).

R. P. Dick is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: dickrp@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

System reliability is related to the reliability of each transistor, the number of transistors, the distribution of permanent failures, and the mechanisms of failure propagation [1]. For a given transistor, lifetime is temperature dependent. Generally, a higher temperature increases the effects of the gradual displacement and mass transportation of the atoms in metal wires, and the deterioration of the gate dielectric layer. Changes in temperature over time also influence reliability. When integrated circuit (IC) materials with differeing thermal expansion coefficients abut, thermal fluctuation produces mechanical stress, which leads to failures [2]. Reliability modeling and optimization techniques should consider these divergent wear processes.

Most existing work on temperature management focuses on improving lifetime reliability by minimizing chip's peak temperature [3]–[6] or increasing the quality-of-service (QoS) given a temperature constraint [7]–[10]. However, minimizing peak temperature sometimes implies increasing thermal cycling, so it may, or may not, have a net positive impact on reliability [11]. The lifetime reliability maximization problem has its own unique characteristics and requires an in-depth analysis of the temporal temperature profile, including its peaks and valleys.

There exist several efforts that directly address increasing lifetime reliability, measured by mean time to failure (MTTF), via task mapping, scheduling [12]–[15], and dynamic voltage and frequency scaling (DVFS) [16], [17]. However, the complexity of the reliability and thermal models (often based on Monte Carlo simulations [1], [18]–[20]) result in high execution overheads, making them impractical for online use. In order to reduce overhead, a common strategy is to use device-level MTTF model to directly replace the system-level model [12]–[15], [21]. This, however, is a potentially inaccurate approximation as the temporal fault distributions of individual devices and million-device systems differ greatly. Only system-level models can accurately determine system-level MTTF.

In this paper, we introduce an on-line framework called reliability-aware utilization control (RUC) to improve the lifetime reliability in the presence of wear caused by failure mechanisms that strongly depend on temperature and temperature variation, such as electromigration, time-dependent dielectric breakdown, and stress migration and thermal cycling. We pay special attention to reducing the complexity of reliability control and lifetime reliability estimation, enabling on-line, in-system reliability estimation. Our main contributions follow.

1) Based on the observation that increasing utilization via

decreasing core frequencies reduces core temperature, we designed a model predictive controller (MPC) that keeps system utilization at a desired value called the utilization set point. Our MPC algorithm incorporates various design considerations including real-time constraints and MTTF dependency on peak temperature and load balancing.

2) We developed a heuristic algorithm to dynamically adjust the sampling window length of the MPC, thereby influencing (i) system reliability via peak temperature and thermal cycling and (ii) the MPC computational overhead. Our heuristic algorithm adjusts sampling window length to balance these considerations.
3) We implemented our on-line framework both in a simulation environment and on a hardware board (NVidia's Jetson TK1 board containing a Tegra K1 processor with four Cortex-A15 ARM cores [22]). The simulator's parameters are calibrated based on the measurement from TK1 board.

We conducted a large set of experiments on the hardware board and the simulator to validate our approach and compared it with two existing control methods: temperature-aware (TA) and utilization control (UC). For 10 MiBench benchmarks [23], RUC improves MTTF by 50% on average, and by up to 141% for processors with high power density. Improvements are small for very low power density processors.

The rest of the paper is organized as follows. We review related work in Section II. Section III introduces hardware and software models and describes the system-level reliability model. Section IV formulates the problem and provides an overview of the RUC framework. Section V describes RUC in detail. Sections VI and VII describe our experimental setup and results. Section VIII concludes the paper.

II. Related Work

Researchers have modeled system lifetime reliability in several ways. An architecture-level model was designed to calculate a processor's lifetime due to electromigration (EM), stress migration (SM), time dependent dielectric breakdown (TDDB), and thermal cycling (TC) [24]. The model was used to analyze the effects of CMOS technology scaling on system lifetime [2]. A statistical model and simulation methodology were used to determine multi-core system-on-chip reliability [25]. System lifetime has also been estimated in the presence of simulated sequences of failures [18]. A fast and accurate Monte Carlo based modeling framework that integrates device-, component-, and system-level models was proposed [1].

Several papers have described using the above models and hardware to improve lifetime reliability through offline optimization strategies. For periodic tasks running on a multiprocessor system-on-chip (MPSoC), Huang et al. proposed an analytical model to estimate the lifetime reliability of MPSoCs and a task mapping and scheduling algorithm to guard against aging effects [12]. Based on the same model, Das et al. extended the work to improve the lifetime of network-on-chips and also solve the energy-reliability tradeoff problem for multimedia MPSoCs [14], [15]. Based on a system-level lifetime reliability model, Zhou et al. presented a unified metric combining system-level lifetime reliability and soft-error reliability, and maximized the soft-error reliability under the lifetime reliability constraint [20]. These approaches can improve lifetime reliability but require that tasks and their timing and power characteristics be known at design time, and remain static. Our work supports tasks with characteristics that are unknown at design time and change during system operation, using dynamic voltage and frequency scaling as a mechanism.

Several dynamic reliability management (DRM) approaches have been developed. Hartman et al. describe a run-time task mapping technology to maximize system-level lifetime reliability by utilizing wear sensors [19]. However, wear sensors are very frequently unavailable or inaccessible to system software. Bolchini et al. dynamically determined the most effective mapping of tasks to minimize network-on-chip energy consumption and maximize the lifetime [26]. Mandelli et al. proposed a runtime distributed energy-aware mapping technique that balances thermal distribution and improves reliability [27]. Bolchini et al. analyzed how load distribution strategies influence system lifetime reliability [28]. Das et al. proposed a machine learning based algorithm to handle inter- and intra-application variations and reduce peak temperature and thermal cycling [21]. All the above work improves lifetime reliability and captures variations in workload and run-time environments. However, none of them use a system-level reliability model, potentially undermining accuracy and fidelity. In this paper, we control a core's frequency and dynamically determine the length of the control period to improve the system-level reliability of real-time multicore systems.

III. Models

In this section, we present the hardware platform as well as the task and reliability models used in the RUC framework.

A. Hardware Platform

We focus on homogeneous multicore systems in this paper. Let $M = \{\rho_1, \rho_2, ..., \rho_m\}$ denote the set of m cores in a given system. Power consumption is the fundamental cause of rising temperature and hence system failure. A core dissipates idle power when it is idle and consumes additional active power when it performs operations [6]. Both active and idle power are related to the core's frequency. Let the utilization of a core in a given time interval Δt be $U = \frac{\Delta t_a}{\Delta t}$, where Δt_a is the amount of time that the core executes operations [6]. Run-time temperature can be estimated using an RC thermal modeling tool or measured by thermal sensors. We use both thermal sensors and Hotspot [29], an integrated circuit thermal modeling tool, in experimental evaluations.

B. Task Model

We consider independent real-time tasks with soft deadlines. A task that misses its deadline is immediately terminated. Tasks are periodic and follow partitioned scheduling, i.e., they

are mapped to cores statically and no migration is allowed. Furthermore, tasks can be arbitrarily preempted. Such an execution model is prevalent in real-time systems since it incurs low development cost and runtime overhead. Tasks on each core are scheduled according to a real-time scheduling policy such as earliest deadline first (EDF) or rate monotonic (RM) scheduling [30].

We use τ_i to denote the i^{th} task. Since all the jobs of the i^{th} task have the same properties, τ_i also denotes the jobs of the i^{th} task. τ_i is associated with a tuple $\{e_i, d_i\}$, where e_i is the execution time and d_i is the relative deadline, as well as the period. For a given duration, hereafter referred to as sampling window k (SW_k), if core ρ_i's frequency is fixed at $f_i(k)$, its utilization can be calculated as

$$U_i(k) = \sum_{\tau_j \in \Gamma(k)} \frac{e_j}{d_j} = \sum_{\tau_j \in \Gamma(k)} \frac{e_j^* + e_j'/f(k)}{d_j}, \qquad (1)$$

where e_j' reflects the portion of job execution that is dependent on the core's frequency and e_j^* is the independent portion. $\Gamma(k)$ is the set of jobs executed in SW_k.

C. System-Level Reliability

We consider four main IC failure mechanisms in this paper: electromigration (EM), time dependent dielectric breakdown (TDDB), stress migration (SM), and thermal cycling (TC) [24]. EM is the dislocation of metal atoms and TDDB is the deterioration of the gate oxide layer. SM is caused by directionally biased motion of atoms in metal wires. Wear due to EM, SM, and TDDB is strongly dependent on temperature. TC refers to IC fatigue failures caused by thermal mismatch deformation. We focus on short-term thermal cycling in this work, as it dramatically decreases reliability [11] and accelerates system failure [31]. In this paper, we propose a framework to improve system-level reliability by reducing the effects of both temperature and thermal cycling. System-level reliability is calculated by a system-level Monte Carlo reliability modeling tool [1].

Wear due to EM, SM, and TDDB is exponentially dependent on temperature. However, the wear due to TC depends on the amplitude (e.g., the difference between the peak and valley temperature), period, and maximum temperature of thermal cycles. Fig. 1 summarize some system MTTF data obtained from the system-level reliability tool with default settings [1]. Figs. 1(a), (b), and (c) depict the MTTF of an example system as a function of the amplitude, period, and peak temperature of thermal cycles, respectively. As a comparison, Fig. 1(d) shows the system MTTF due to temperature alone without thermal cycles. As can be seen from Fig. 1, system MTTF is generally increases at lower temperatures and with smaller thermal cycles, but the precise relationship is complicated.

We used a Monte Carlo simulation based modeling tool [1] to calculate system-level reliability [11], [32]. In Monte Carlo simulation, numerous trials allow high accuracy but increase computation overhead (i.e., the execution time to complete MTTF calculation). Hence, analyzing the accuracy-overhead tradeoff is necessary. We use the tool [1] to calculate the system MTTF with default settings and show how the number

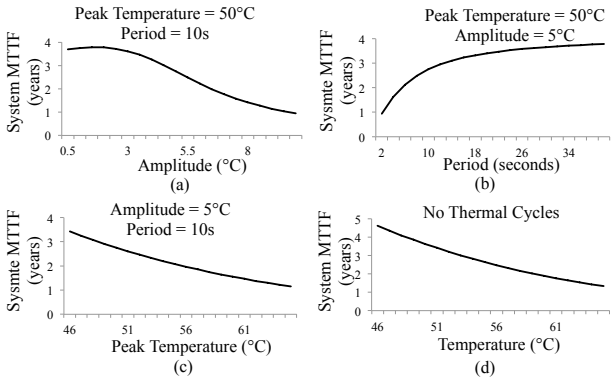

Fig. 1. System MTTF dependence on thermal cycle (a) amplitude, (b) period, and (c) peak temperature, and on (d) temperature without thermal cycles.

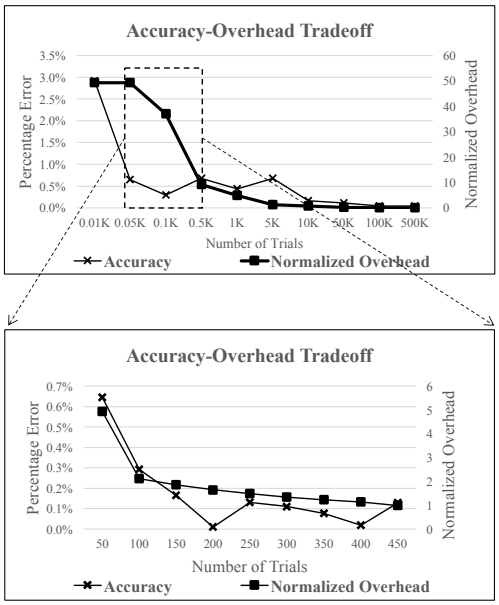

Fig. 2. The percentage error and overhead due to Monte Carlo trials.

of Monte Carlo trials affects overhead and accuracy (see Fig. 2). We set the total number of cycles to failure to 10^6, the default of this tool, as the baseline for comparison. The metric, percentage error, is used to quantify the accuracy. Increasing trial count decreases speed and increases accuracy. However, the accuracy saturates. The values in Fig. 2 are chip dependent, but the accuracy-overhead tradeoff is more general. Fig. 2 helps to find the most appropriate number of trials. For example, 100 is a good choice if 0.5% error is acceptable. Note that other techniques, e.g., importance sampling, may be used to improve the computational efficiency of reliability modeling if supported by the modeling tool in use.

IV. RUC Framework

In this section, we first formulate the design problem, and then briefly introduce how RUC solves this problem.

A. Problem Formulation

A system's lifetime reliability is determined by its operating temperature and thermal cycles. Given that lower frequencies

and voltages lead to higher utilizations (see Eq. (1)) but lower temperatures, utilization control in a DVFS-enabled system can be used to manage temperature. Based on this observation, we propose to indirectly improve the system MTTF by directly controlling system utilization.

Our goal is to minimize the deviation of the cores' utilization from a set point in each time interval, referred to as sampling window (SW). This concept has previously been used in controller based resource management [33] and temperature-aware control strategies [34]. The length of SW is chosen such that a core's temperature can be considered constant within a sampling window. Before formulating the utilization control problem, we introduce the following notation:

- $U_i(k)$ is the utilization of core ρ_i during the k^{th} sampling window, SW_k;
- U^s is the utilization set point;
- $U^{\max}$ is the upper bound on the utilization to ensure schedulability;
- $f_i(k)$ is the frequency of core ρ_i during SW_k;
- $f_{\min}$ and $f_{\max}$ are the lowest and highest core's frequencies allowed, respectively; and
- $\bar{f}(k)$ is the average frequency over all cores during SW_k.

We aim to solve the following problem:

$$\min \sum_{\rho_i \in M} (U^s - U_i(k))^2. \tag{2}$$

The solution to Eq. (2) must satisfy these constraints:

$$U_i(k) \leq U^{\max} for \rho_i \in M, \tag{3}$$

$$-f_{th} \leq f_i(k) - \bar{f}(k) \leq f_{th} for \rho_i \in M, and \tag{4}$$

$$f_{\min} \leq f_i(k) \leq f_{\max} for \rho_i \in M. \tag{5}$$

The first constraint ensures that no cores exceed the schedulability bound. Based on the discussion in Section III-C, the second constraint is introduced to bound the differences in the core frequencies, which in turn bounds the core temperature differences. This constraint requires the differences between each core's frequency and the average frequency of all cores, $\bar{f}(k)$, to be smaller than a threshold f_{th}. The third constraint limits core frequencies.

Solving the above problem by controlling a core's frequency reduces operating temperature under the real-time constraints. A shorter sampling window results in a lower temperature, but frequently altering a core's frequency causes more fluctuation in operating temperature and increases the effects of thermal cycling. We tune the length of the sampling window to balance peak temperature and thermal cycling dependent wear.

B. Overview of Reliability-Aware Utilization Control (RUC)

Real computing systems (and especially embedded systems) frequently operate in environments with ambient temperature and task execution time variations that cannot be predicted at design time, motivating us to develop an on-line reliability optimization framework. This framework improves system MTTF by solving the optimization problem defined in (2)–(5) and dynamically tuning the length of the sampling window.

The framework, RUC, consists of two main components: a global utilization controller (GUC) and a sampling window controller (SWC) (see Fig. 3). The GUC reduces the peak temperature by dynamically adjusting core frequencies to adhere to the utilization set point. The SWC minimizes thermal cycling wear by dynamically adjusting the length of the sampling window. We assume that each core has a utilization monitor (UM) and a temperature sensor (TS).

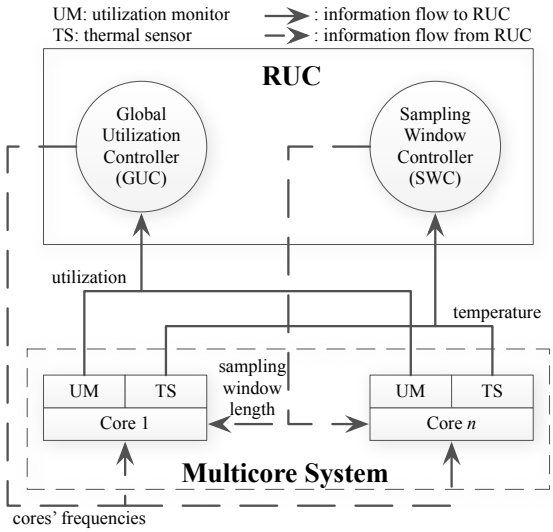

Fig. 3. High-level overview of RUC.

At the end of the k^{th} sampling window SW_k, the utilization and temperature of each core are measured and sent to the GUC and SWC (see the solid lines in Fig. 3), respectively. Based on the utilization, the GUC solves the optimization problem defined in (2)–(5). The solution sets core frequencies for the next sampling window, which are sent (see the dashed lines in Fig. 3) to each core before the start of SW_{k+1}. Unlike the GUC, the SWC stores temperature history. Specifically, the temperatures measured in the most recent s sampling windows are saved. We refer to the s sampling windows as one profiling window. At the end of each profiling window, a temperature profile is generated. SWC then analyzes this temperature profile and determines whether to increase or reduce the sampling window length for the next profiling window.

V. Resource Utilization Controller (RUC) Details

This section presents the detailed design of the GUC and SWC. We introduce a new utilization model describing how a core's utilization changes with frequency. Based on this model, we discuss how the GUC solves the problem defined in (2)–(5) using a controller. Finally, we elaborate on how the heuristic used in the SWC adjusts the length of the sampling window dynamically at runtime to improve system MTTF.

A. Dynamic Utilization Model

In order to control utilization, we must first model it. We use a dynamic model to determine utilization as a function

of core frequency. We first rewrite the non-linear utilization model in Eq. (1) in linear form

$$u_j(k) = \sum_{\tau_i \in \Gamma(k)} \frac{e_i^*}{d_i} + \sum_{\tau_i \in \Gamma(k)} \frac{e_i'}{d_i} \times F_j(k), \tag{6}$$

where $F_j(k) = 1/f_j(k)$ is the clock period, which we refer to as the *manipulated variable*. The dependence of utilization on frequency can be modeled as

$$u_j(k+1) = u_j(k) + g_j(k) \times \Delta F_j(k), \tag{7}$$

where $g_j(k) = \sum_{\tau_i \in \Gamma(k)} \frac{e_i'}{d_i}$ and $\Delta F_j(k) = F_j(k+1) - F_j(k)$. Since we have m cores, we use a matrix to describe the dynamic utilization model as

$$U(k+1) = U(k) + G(k) \times \Delta F(k). \tag{8}$$

$G(k)$ is a diagonal matrix and $G(k)_{i,i} = g_i(k)$. $\Delta F(k) = [\Delta F_1(k), \ldots, \Delta F_m(k)]^T$ and $U(k) = [u_1(k), \ldots, u_m(k)]^T$.

Although Eq. (8) indicates the behavior of utilization as a function of the manipulated variable, it cannot be used directly because $G(k)$ is only known at runtime. According to feedback control theory, if stable control can be achieved, the value of $G(k)$ does not affect the final result [33]. Therefore, assuming the system is stable, we can set $G(k)$ to constant G and rewrite Eq. (8) as

$$U(k+1) = U(k) + G \times \Delta F(k). \tag{9}$$

B. Global Utilization Controller (GUC) Design

We solve the constrained optimization problem defined in (2)–(5) using a model predictive controller (MPC). The basic idea behind any MPC is to optimize a cost function. Hence, we first find similarities between the constrained optimization problem defined in (2)–(5) and an MPC cost function optimization problem. After this, we transform the MPC cost function optimization problem to a standard quadratic programming problem and solve it using existing solvers.

Inside the k^{th} sampling window, MPC minimizes the cost function,

$$J(k) = \sum_{i=1}^{N_1} \delta_i [U(k+i) - \xi(k+i)]^2 + \sum_{j=1}^{N_2} \lambda_j [\frac{1}{\Delta f(k+j-1)}]^2, \tag{10}$$

where N_1 is the prediction horizon and N_2 is the control horizon. δ_i is the tracking error weight and λ_j is the control penalty error [6]. The user-specified reference trajectory $\xi(k+i)$ defines the ideal trajectory along which the utilization should converge to the set point.

Although this standard cost function in Eq. (10) is not our proposed optimization function in Eq. (2), the solution of the cost function is also the solution of Eq. (2). The first term in the cost function Eq. (10) is a variation of the function in Eq. (2). The second term in Eq. (10) minimizes the changes in the manipulated variable and does not affect the final result of the optimization problem. It only helps to make the control more stable. Hence, minimizing the cost function Eq. (10) under the constraints in (3)–(5) would also lead to an optimal solution to the problem defined in (2)–(5).

We propose using a quadratic programming solver to solve the optimization problem. A standard quadratic programming problem can be written as

$$\min_{\varepsilon} \{\frac{1}{2}\varepsilon^T \Omega \varepsilon + \zeta^T \varepsilon\}, s.t. \begin{cases} A_0 \times \varepsilon \leq b_0 \\ A_1 \times \varepsilon = b_1 \\ b_l \leq \varepsilon \leq b_u \end{cases}, \tag{11}$$

where Ω, A_0, and A_1 are matrices. ζ, b_0, b_1, b_l, and b_u are vectors and ε denotes the change in a core's frequency.

Since this standard quadratic programming problem can be directly solved by existing tools, the key point in MPC design is to transform the cost function defined in Eq. (10) and the constraints defined in (3)–(5) to Eq. (11). Based on ε and the core's frequency inside SW_k, the core's frequency at SW_{k+1} can be directly calculated. Because the solution to Eq. (10) is also the solution to Eq. (2), the optimal solution to the quadratic programming problem is also the optimal solution to the proposed optimization problem defined in (2)–(5).

We first transform the MPC cost function in Eq. (10) to the target function in standard quadratic problem. Suppose we have $h = \max\{N_1, N_2\}$, then the cost function can be rewritten as

$$J(k) = [Y'(k) - \Phi(k)]^T \Pi [Y'(k) - \Phi(k)] + \varepsilon^T \Theta \varepsilon, \tag{12}$$

where $Y'(k) = [U(k+1), \ldots, U(k+h)]^T$ and $\Phi = [\xi(k+1), \ldots, \xi(k+h)]^T$. Π and Θ are two diagonal matrices where $\Pi_{i,i} = \delta_i$ and $\Theta_{i,i} = \lambda_i$. Based on Eq. (9),

$$Y'(k) = Y(k) + \Lambda \times \varepsilon, \tag{13}$$

where $Y(k) = [U(k), \ldots, U(k)]^T$. Λ is a low triangular matrix and $\Lambda_{i,j} = G$ if $i \geq j$ and 0 otherwise. Therefore, Eq. (10) can be simplified as

$$J(k) = \varepsilon^T (\Lambda^T \Pi \Lambda + \Theta)\varepsilon + 2(Y(k)^T \Pi \Lambda - \Phi(k)^T \Pi \Lambda)\varepsilon + H. \tag{14}$$

H is independent of the manipulated variable ε. The cost function is successfully transformed to the objective function in a standard quadratic programming problem in Eq. (11), where $\Omega = 2(\Lambda^T \Pi \Lambda + \Theta)$ and $\zeta = 2[Y(k)^T \Pi \Lambda - \Phi(k)^T \Pi \Lambda]^T$.

The next step is to transform the constraints to the standard form. The first constraint in (3) can be described as

$$\Lambda \times \varepsilon \leq U^* - Y(k), \tag{15}$$

where $U^* = [U^{\max}, \ldots, U^{\max}]^T$. This is a standard form where Λ and $U^* - Y(k)$ correspond to A_0 and b_0 in Eq. (11), respectively.

The second constraint in (4) enforces the even distribution of core frequencies. The constraint on core frequencies constrains core clock periods,

$$-F_{th} \leq F_i(k) - \bar{F}(k) \leq F_{th}, \tag{16}$$

where the F_{th} is the threshold related to f_{th}. The clock period of core ρ_i at the $(k+h)^{th}$ sampling window, $F_i(k+h)$, is

$$\Psi_0 = \Psi_2 + \Psi_1 \times \varepsilon, \tag{17}$$

where Ψ_1 is a block matrix where the element $\Psi_1(i,j)$ is a unit matrix if $j \leq i$, and 0 otherwise. $\Psi_0 = [F_1(k+1), \ldots, F_m(k+1), \ldots, F_1(k+h), \ldots, F_m(k+h)]^T$ and $\Psi_2 =$

$[F_1(k), \ldots, F_m(k), \ldots, F_1(k), \ldots, F_m(k)]^T$. Based on (16) and Eq. (17), the second constraint can be expressed as

$$(\Psi_3\Psi_1 - \Psi_3\Psi_5\Psi_1)\varepsilon \leq \Psi_4 - \Psi_3\Psi_2 + \Psi_3\Psi_5\Psi_2, \quad (18)$$

where $\Psi_4 = [F_{th}, \ldots, F_{th}, -F_{th}, \ldots, -F_{th}]^T$ and $\Psi_3 = [E, -E]^T$. Ψ_5 is diagonal matrix and $\Psi_{5_{i,i}} = \Psi_6$, where Ψ_6 is a matrix where all elements are equal to $1/m$.

The last constraint in (5) sets the upper and lower bounds on core frequencies. Based on Eq. (17), it can be expressed as

$$\Psi_3\Psi_1\varepsilon \leq \Psi_7 - \Psi_3\Psi_2, \quad (19)$$

where $\Psi_7 = [F_{max}, \ldots, F_{max}, -F_{min}, \ldots, -F_{min}]^T$, $F_{max} = \frac{1}{f_{max}}$, and $F_{min} = \frac{1}{f_{min}}$.

After the above transformations, the problem in (2)–(5) is a standard quadratic programming problem and can be directly solved using standard solvers, e.g., COPL_QP [35] or Python's package CVXOPT [36].

GUC aims to reduce the peak and average temperature for the entire multicore chip and to balance the temperatures accross cores. Although a lower temperature tends to increase the system MTTF, the latter also depends on the amplitudes, periods, peak and valley temperature of thermal cycles. Hence, we propose a sampling window controller to reduce the aging effect of thermal cycling.

C. Sampling Window Controller (SWC) Design

We design SWC to minimize the aging effect of thermal cycling by dynamically changing the length of the sampling window, L_{sw}. Since a core's frequency can change from one sampling window to the next, the length of sampling window directly impacts thermal cycling. In general, a shorter sampling window means scaling core's frequency more frequently, and helps to reduce the peak temperature and amplitude of thermal cycles, but may increase the frequency of thermal cycles.

It is difficult to precisely model how the sampling window affects the system reliability, and the complicated reliability model also increases the overhead of searching for the best sampling period. Hence, to balance the impact of peak temperature against that of thermal cycles on the system MTTF, we design an efficient online heuristic based on binary search to adjust L_{sw} at runtime.

Our heuristic algorithm in the SWC uses the concept of a profiling window to adjust the L_{sw}. A profiling window is composed of s equal-length sampling windows. In each profiling window, the s temperature points make up a temperature profile. At the end of the profiling window, we calculate the system MTTF from the temporal temperature profile using a reliability modelling tool [1]. Since this tool assumes the input temperature profile is repeated until the chip fails, the established temperature profile must have enough temperature points. Although a larger s makes the calculation of system MTTF more precise, it increases the overhead of the reliability modeling tool. In our experimental evaluations, we find that setting $s = 50$ achieves an accurate system MTTF with an acceptable overhead. We store the history of the system MTTF for the various length of the sampling window and rely on binary search to find the most appropriate L_{sw} for the sampling windows in the next profiling window.

To perform binary search, we need to determine the lower and upper bound of L_{sw}. We set the lower bound of L_{sw} (L_{sw}^{lb}) to 1 second. Since the quadratic programming solver executes at the end of each sampling window, and the execution time of a typical solver, e.g., COPL_QP [35], is less than 10 ms on our platform (with ARMv7 1.23 GHz), setting L_{sw}^{lb} to $1s$ keeps the overhead due to the GUC under 1% of the sampling window length. To determine the upper bound of the L_{sw}^{ub}, we consider two factors. Since the GUC expects a constant temperature in a given sampling window, the sampling window length must be smaller than some constant, say C_{HW}. C_{HW} is dependent on the hardware platform. A low power density chip leads to a large C_{HW} and vice versa. As for the other factor, since thermal cycles can essentially be avoided if the L_{sw} equals the hyperperiod of the periodic task set, any length larger than the hyperperiod increases temperature and thermal cycling. Hence, we only need to consider lengths of the sampling window less than or equal to the hyperperiod. As a result, we set $L_{sw}^{ub} = \min\{HP, C_{HW}\}$.

The pseudo-code for the SWC is given in Alg. 1. Lines 1–2 initialize the needed variables, where L_{sw}^c is the length of the current sampling window. The infinite while-loop (Lines 3–25) allows the algorithm to run throughout system lifetime. At the end of each sampling window, the temperature of each core is read from temperature sensors (Lines 5–7). At the end of the j^{th} profiling window ($i = s$), a temperature profile, $TP(j)$, is constructed based on the collected temperature of the m cores in the current s sampling windows (Line 10). The corresponding MTTF, $MTTF_j$, is obtained using a reliability modeling tool [1] (Line 11). The L_{sw}^c is next determined for the sampling windows in the upcoming profiling window (Lines 12–23). L_{sw}^c is set to $(L_{sw}^{lb} + L_{sw}^{ub})/2$ at the second sampling window (Lines 12–13), and is then dynamically changed based on the comparison of MTTFs (Lines 15–20). L_{sw}^{lb} and L_{sw}^{ub} are updated and L_{sw}^c converges to the most appropriate value. Finally, i and j are updated to indicate the beginning of the next profiling window (Line 22).

VI. Experimental Setup

To evaluate the proposed reliability-aware utilization control framework, we conducted experiments to compare the proposed framework with two existing approaches. In this section, we present the platforms, benchmarks, and the existing frameworks used for comparison in our experiments.

A. Benchmarks

A total of 10 benchmarks were chosen from Mibench [23] which include different tasks from automotive, network, office, security, telecommunication, and consumer applications. We measured the execution times of the benchmarks on a quad-core ARM Cortex-A15 chip, Nvidia Tegra K1 [22], when a core's frequency is 1.24 GHz and then assigned them to appropriate cores (see Table I). Note that Core 0 is reserved for the operating system. Based on the execution time data, we designed two groups of task setups. In the first group, tasks

Algorithm 1: SWC(L_{sw}^{lb},L_{sw}^{ub})

```
i ← 1, j ← 1
L_sw^c ← L_sw^lb
while True then
    if at the end of SW_i then
        for each core ρ_k then
            measure temperature T_k(i)
        end for
        i ← i + 1
        if i = s then
            TP(j) ← {T_i(1), ··· , T_m(s)}
            calculate MTTF_j
            if j = 1 then
                L_sw^c ← (L_sw^lb + L_sw^ub)/2
            else
                if MTTF_j ≥ MTTF_{j−1} then
                    L_sw^lb ← L_sw^c
                else
                    L_sw^ub ← L_sw^c
                end if
                L_sw^c ← (L_sw^lb + L_sw^ub)/2
            end if
            i ← 1, j ← j + 1
        end if
    end if
end while
```

TABLE I
BENCHMARKS

Benchmark	Application	Mapping to	Execution time
bitcount	Automotive	Core 2	960 ms–969 ms
susan	Automotive	Core 1	129 ms–151 ms
qsort	Automotive	Core 1	176 ms–195 ms
dijkstra	Network	Core 3	130 ms–141 ms
patricia	Network	Core 3	445 5ms–459 5ms
stringsearch	Office	Core 2	3 ms–10 ms
blowfish	Security	Core 1	1352 ms–1369 ms
sha	Security	Core 2	54 ms–60 ms
crc32	Telecommunication	Core 3	843 ms–874 ms
cjpeg	Consumer	Core 3	16 ms–20 ms

are frame-based and share the same period and deadline. We will measure the system-level MTTFs when the deadline is 1.6, 1.8, 2.0, 2.2, and 2.4 seconds. In the second group, a task's deadline and period are random in the ranges 1.2–1.6, 1.6–2.0, 2.0–2.4, and 2.4–2.8 seconds.

B. Comparison Targets

We compared our proposed RUC with two existing frameworks: pure utilization control (UC) [33] and temperature-aware control (TA) [6], [9]. In contrast with RUC, the length of sampling window in UC is fixed. Generally, it is set to 1 s [33]. In addition, since UC is not designed for lifetime reliability, it does not bound the differences in core frequencies. TA has a similar design to UC, but the goal is to control the temperature to a specific set-point, which is lower than the maximum temperature allowed by the chip. Hence, TA is an effective way to guarantee that the operating temperature is safe.

Two metrics are considered in the comparison. Reliability is quantified by the system MTTF, which is obtained using the system-level reliability modeling tool [1]. Real-time performance is quantified as the percentage of feasible solutions (FS), which is the ratio of the number of tasks satisfying their real-time requirements to the total number of tasks.

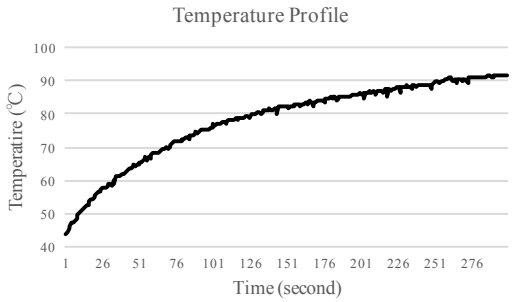

Fig. 4. Temperature profile with four fully loaded cores running at the highest frequency.

C. Experimental Platforms

We conducted experiments on two platforms: an ARMv7 chip, and a simulator. The first experimental platform is a quad-core Cortex-A15 ARM chip: Nvidia's Tegra K1 (TK1) [22]. This Nvidia TK1 SoC is designed for mobile and automotive applications. The quad-core ARM Cortex-A15 CPU and 192 Kepler GPU cores provide high performance with low power requirement. As a low-power chip, TK1 supports 12 different frequencies from 1.24 GHz to 2.32 GHz, but all cores are required to have the same frequency. Except for the primary core (core 0), all cores can be powered on and off dynamically. There are 11 thermal sensors to sample the CPU, GPU, and other component temperatures every 50 ms. However, the default interface only provides one CPU temperature for all CPU cores. TK1 is shipped with an operating system based on Ubuntu 14.04 LTS. Hence, most of desktop-level benchmarks can be directly executed on TK1.

In order to evaluate RUC on different platforms, we constructed a hardware platform simulator. We extracted the parameters from TK1 to build power model. In the simulator, the temperature is obtained using Hotspot, a thermal modeling tool [29]. Our simulator is tested via a comparison to the TK1.

We first obtained the parameter values for Hotspot. Since TK1 only reports one temperature for all cores, we made an assumption that all cores have the same thermal capacitance, C, and thermal resistance R. We used no fan when measuring the temperature. The temperature profile for four fully loaded core running at the highest frequency is shown in Fig. 4. Note that due to power variation and other uncontrollable parameters, e.g., small changes in ambient temperature, the obtained temperature profile fluctuated slightly. In this configuration, the chip's power is 6.98 W [37], so R can be estimated as 10.24 °C/W. For the same reasons, we estimated C to be in the range of 2.34–7.34 J/°C. The capacitance used in Hotspot is randomly generated in this range.

Based on R and C, we determined the power model for the simulator. Let $P_{act}(f)$ be the core's active power at frequency f. $P_{oth}(f)$ is the power independent of core utilization but related to core frequency. $P_{oth}(f)$ includes the core's idle power and other components' power, e.g., GPU and memory. The average power ($\bar{P}$) can be calculated as

$$\bar{P} = U \times P_{act}(f) + P_{oth}(f). \tag{20}$$

TABLE II
PARAMETERS IN POWER MODEL

Frequency (GHz)	$P_{act}(f)$	$P_{oth}(f)$
1.24	0.14648	2.24609
1.33	0.19124	2.26847
1.43	0.22108	2.31595
1.53	0.23600	2.38851
1.63	0.26991	2.48684
1.73	0.37164	2.51329
1.84	0.46251	2.54245
1.94	0.53033	2.56822
2.01	0.59001	2.58993
2.12	0.69851	2.61163
2.22	0.86806	2.61502
2.32	1.11626	2.64147

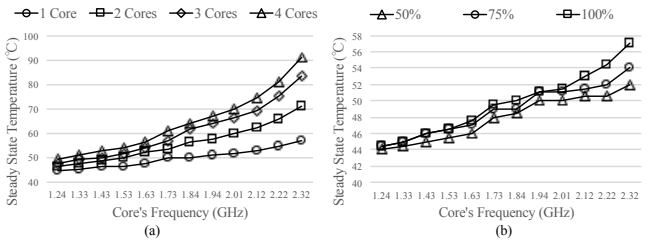

Fig. 5. Steady-state temperature with (a) different numbers of fully loaded cores running at different frequencies and (b) one core running at different frequencies and utilizations.

The value of $P_{act}(f)$ and $P_{oth}(f)$ are extracted from the measurement of TK1. It is difficult to measure TK1 power consumption directly. Instead, we used steady state temperatures to determine values of $P_{oth}(f)$ and $P_{act}(f)$. Fig. 5(a) shows the steady state temperature with different numbers of fully loaded cores running at different frequencies. Fig. 5(b) depicts the steady state temperatures for one core running at different frequencies and utilizations. Note that since the resolution of the thermal sensors is 0.5°C, manipulating a core's frequency may not change the measured steady-state temperature, especially when the workload is light. Based on these measurements, the power values are derived and summarized in Table II.

In order to validate this power model, we compared the temperature readings from the thermal sensors with those estimated by Hotspot (see Fig. 6). As can be seen from Fig. 6, the parameter values that we have obtained resulted in accurate temperature estimation. The average temperature difference between TK1 and the simulator is less than 2°C, and Fig. 7 shows that this small difference does not impact the system-level MTTF. The data in Fig. 7 indicate that RUC has a similar performance on both TK1 and the simulator. The maximum difference is about 2%, and the average is less than 1%. These experiments demonstrated that the parameter values that we have obtained indeed lead to reliable accuracy of the simulator.

VII. RESOURCE UTILIZATION CONTROL (RUC) EVALUATION

In this section, we examine the performance of the proposed RUC compared to the utilization control (UC) and temperature-aware control (TA) approaches.

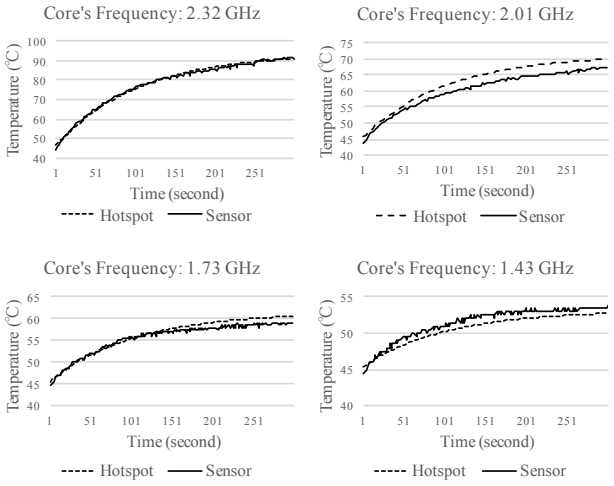

Fig. 6. Temperature readings from thermal sensors and estimated by Hotspot for various core frequencies.

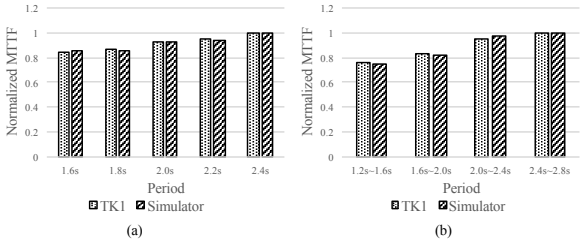

Fig. 7. Normalized MTTFs with TK1 and constructed simulator.

A. Experiments on the Tegra chip

We compared the proposed RUC with UC and TA to determine whether RUC can improve lifetime reliability without sacrificing real-time performance. For RUC and UC, the utilization set-point is close to the schedulability boundary, which was set to 70%. We used a temperature set point of 70°C in TA, which is lower than the peak operating temperature.

We first studied the overhead of the solver to the optimization problem defined in (2)–(5). If four cores are required to have the same frequency, such as TK1 [22], a brute-force search solver consumes less than 1 ms to find the optimal solution. This overhead is tolerable in our RUC. However, if cores can run at different frequencies, the overhead of brute-force search increases to 74 ms. In this case, brute-force search is impractical. The overhead can be reduced if we use standard solvers, e.g., COPL_QP [35] or CVXOPT [36]. The overhead of COPL_QP is about 4 ms and CVXOPT is about 25 ms. Both their overheads are small, and we can choose one of them offline.

Fig. 8 shows the experimental results when tasks are frame-based. RUC achieves a higher MTTF than UC and TA in all the cases. Thanks to the proposed sampling window control, when compared to UC, RUC improves the MTTF by 9.4% on average, and up to 32%. TA keeps the run-time temperature at 70°C, and the corresponding MTTF is much smaller than RUC and UC when the workload is light. RUC doubles the MTTF when compared to TA. In terms of real-time performance, all

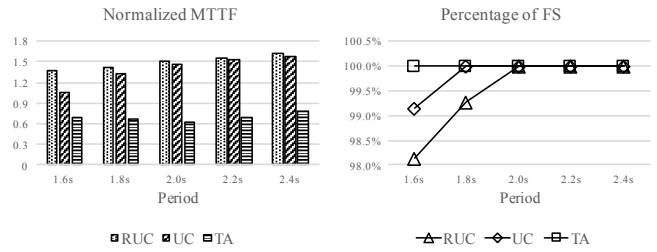

Fig. 8. Normalized MTTF and percentage of FS when tasks are frame-based and running on TK1.

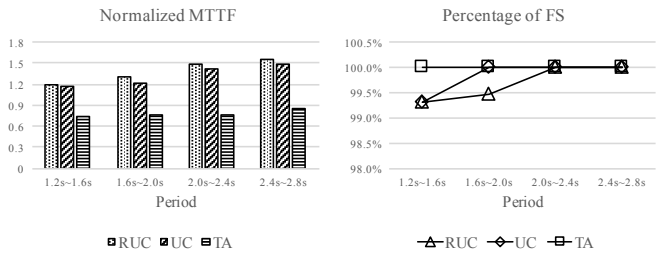

Fig. 9. Normalized MTTF and percentage of FS when tasks' periods are random and running on TK1.

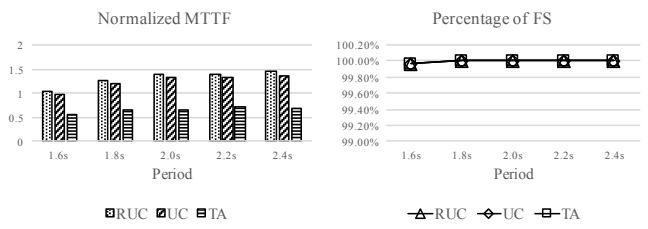

Fig. 10. Normalized MTTF and percentage of FS when tasks are frame-based and running on a TK1 based simulator.

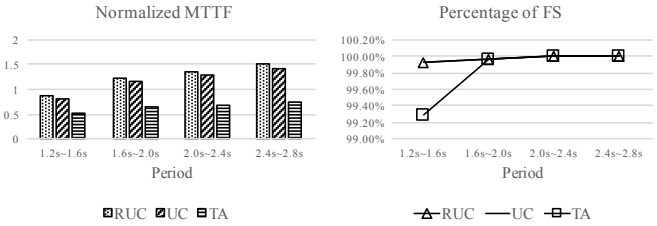

Fig. 11. Normalized MTTF and percentage of FS when tasks' periods are random and running on a TK1 based simulator.

the tested frameworks have a large and similar percentage of feasible solutions (FS). Since RUC may increase the length of the sampling window and reduce the DVFS frequency, it may lead to more tasks missing their deadlines. However, its percentage of FS is only 2% smaller than TA, and also close to 100%. In most soft real-time systems, this is tolerable [33].

In Fig. 9, each task has the same deadline and period, which is randomly generated in different ranges for different tasks. The average MTTF improvement of RUC over UC is 6.4%, and up to 9.5%. Compared to TA, this improvement increases to over 100%. For real-time performance, RUC does not have the highest percentage of FS, but the small loss in number of FS does not limit its application in many soft real-time systems.

B. Simulation Results

Leveraging the simulator, we conducted two sets of experiments to further assess the pros and cons of RUC in a more general setting. Tasks have been assigned to appropriate cores, and each task's execution time is the average of measured execution time on TK1 (see Table I). We extended the simulator so that different cores can operate at different frequencies on this simulator. In the first set of experiments, we assumed the chip has the same low power density as TK1. We increased the power density in the second set of experiments. We used the CVXOPT [36] Python package, to solve the quadratic programming problem.

Fig. 10 and 11 depict the results of the first set of experiments. Fig. 10 shows the MTTF and percentage of FS when tasks are frame-based and Fig. 11 shows the results when tasks' period is randomly generated. Compared to UC, the average improvement of RUC is 5.9% when tasks are frame-based and 6.5% when task periods are randomly generated. As in the experiment on TK1, RUC also doubles the MTTF when compared to TA. All three frameworks allow almost the same number of deadlines to be met. Note that the absolute value of the MTTF and percentages of FS could be different in the experiments on TK1 and on the simulator. This is because the estimated temperature and tasks' execution times may differ from those on the TK1. Both of these experiments show that RUC improves the system MTTF and also guarantees that almost all task deadlines are met.

All the above experiments demonstrate that RUC has a better system reliability performance. However, for a low-power chip, different core frequencies do not lead to significant power changes, which weakens the effect of DVFS and also the sampling window control. For this kind of chip, the MTTF improvement of RUC over UC is less than 32%.

In order to study the effectiveness of RUC on a high power density chip, we used the same simulator as in the previous experiments but increased the chip's power density by $3\times$[1] the power density. We assumed that the Connectland 1504002 heatsink [38] was used. With this heatsink, the thermal resistance is estimated to be 7.02 J/°C, and the thermal capacitance is in the range of 1.92–25.87 watt/°C. For RUC and UC, we used the same CVXOPT [36] to solve the quadratic programming problem.

The experimental results are shown in Fig. 12. In all cases, the average improvement in MTTF by RUC is 39.5% for frame-based tasks and 54.9% when tasks' periods are random. At higher power density, reducing core frequency dramatically reduces run-time temperature. At the same time, frequent changes in DVFS settings introduce more temperature fluctuations, which increases aging from thermal cycling. Hence, sampling window control in RUC benefits lifetime reliability. RUC can achieve a more stable temperature profile than UC, which is one of the most important factors that positively

[1]Increasing the power density by $3\times$ guarantees that the maximum operating temperature still remains below the temperature bound.

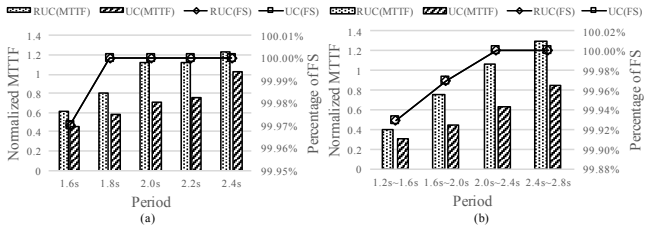

Fig. 12. Normalized MTTF and percentage of FS when running on a simulator with high power density chips with (a) frame-based tasks; (b) tasks with random periods.

impacts the system-level MTTF. The behaviors of RUC and UC have similar real-time performance to that in the previous experiments. The percentage of FS is close to 100%. RUC allows most tasks complete before their deadlines even when the workload is heavy.

VIII. Conclusion

We proposed a reliability-aware utilization control framework to maximize the lifetime of multicore systems under soft real-time constraints. Based on analysis of the influence of operating temperature on thermal cycling and system-level lifetime reliability, we designed an online framework that lowers peak temperature, balances the temperature differences among cores, and reduces thermal cycling. Our framework uses a model predictive controller to reduce peak temperature by adjusting core frequencies and voltages within each sampling window. A heuristic was developed to dynamically adapt the sampling window length to reduce thermal cycling and improve reliability. We conducted experiments with different types of tasks on multiple platforms, including a quad-core ARM chip and a simulator, and considered chips with different power densities. The results revealed that, on all the platforms and with a variety of task setups, our approach was effective in increasing the lifetime of soft real-time systems compared to existing temperature-aware and utilization control approaches.

References

[1] Y. Xiang, et al., "System-level reliability modeling for MPSoCs," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2010, pp. 297–306.

[2] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Proc. Int. Conf. Dependable Systems and Networks*, Jun. 2004, pp. 177–186.

[3] N. Fisher, J. J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proc. Int. Conf. the Real-Time and Embedded Technology and Application Symp.*, Apr. 2009, pp. 131–140.

[4] H. Huang and G. Quan, "Leakage aware energy minimization for real-time systems under the maximum temperature constraint," in *Proc. Design, Automation and Test in Europe*, Mar. 2011, pp. 1–6.

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

[6] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, "Feedback thermal control of real-time systems on multicore processors," in *Proc. Int. Conf. Embedded Software*, Oct. 2012, pp. 113–122.

[7] T. Chantem, X. S. Hu, and R. P. Dick, "Online work maximization under a peak temperature constraint," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 2009, pp. 105–110.

[8] G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *IEEE Trans. Industrial Informatics*, vol. 6, no. 3, pp. 329–339, Aug. 2010.

[9] H. Huang, G. Quan, J. Fan, and M. Qiu, "Throughput maximization for periodic real-time systems under the maximal temperature constraint," in *Proc. Design, Automation Conf.*, June. 2011, pp. 363–368.

[10] O. Sahin, P. T. Varghese, and A. K. Coskun, "Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2015, pp. 839–846.

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

[12] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling on MPSoC platform," in *Proc. Design, Automation and Test in Europe*, Mar. 2009, pp. 51–56.

[13] ——, "On task allocation and scheduling for lifetime extension of platform-based MPSoC designs," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 12, pp. 789–800, Dec. 2011.

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

[15] ——, "Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs," in *Proc. Design, Automation and Test in Europe*, Mar. 2014, pp. 1–6.

[16] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "A Linux-governor based dynamic reliability manager for Android mobile devices," in *Proc. Design, Automation and Test in Europe*, Mar. 2014, pp. 1–4.

[17] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," in *Proc. Design, Automation and Test in Europe*, Mar. 2014, pp. 1–6.

[18] A. Hartman, D. Thomas, and B. Meyer, "A case for lifetime-aware task mapping in embedded chip multiprocessors," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2010, pp. 145–154.

[19] A. Hartman and D. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2012, pp. 13–22.

[20] J. Zhou, X. S. Hu, Y. Ma, and T. Wei, "Balancing lifetime and soft-error reliability to improve system availability," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 2016, pp. 685–690.

[21] A. Das, R. A. Shafik, and G. V. Merrett, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proc. Design, Automation Conf.*, June. 2015, pp. 1–6.

[22] Nvidia, "Jetson Tegra K1." [Online]. Available: https://developer.nvidia.com/embedded/develop/hardware

[23] Electrical Engineering and Computer Science Department, University of Michigan, "Mibench." [Online]. Available: http://vhosts.eecs.umich.edu/mibench//

[24] J. Srinivasan, et al., "The case for microarchitecture awareness of lifetime reliability," in *Proc. Int. Symp. Comp. Arch.*, Jun. 2004, pp. 276–187.

[25] A. Coskun, et al., "A simulation methodology for reliability analysis in multi-core SoCs," in *Proc. Great Lakes Symposium on VLSI*, Apr. 2006, pp. 169–180.

[26] 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. Design, Automation Conf.*, June. 2013, pp. 58–64.

[27] M. Mandelli, G. Castilhos, G. Sassatelli, L. Ost, and F. G. Moraes, "A distributed energy-aware task mapping to achieve thermal balancing and improve reliability of many-core systems," in *Proc. Symp. on Integrated Circuits and Systems Design*, Aug. 2015, pp. 13–19.

[28] C. Bolchini, L. Cassano, and A. Miele, "Lifetime-aware load distribution policies in multi-core systems: An in-depth analysis," in *Proc. Design, Automation and Test in Europe*, Mar. 2016, pp. 804–809.

[29] K. Skadron, et al., "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[30] C. Liu and J. Layland, "Scheduling algorithm for multiprogramming in a hard-real-time environment," *J. of ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[31] V. H. Nguyen, "Multilevel interconnect reliability on the effects of electro-thermomechanical stresses," Ph.D. dissertation, Queensland University of Technology, 2004.
[32] Y. Ma, T. Chantem, X. S. Hu, and R. P. Dick, "Improving lifetime of multicore soft real-time systems through global utilization control," in *Proc. Great Lakes Symposium on VLSI*, May 2015, pp. 79–82.
[33] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, Jun. 2005.
[34] F. Zanini, et al., "Multicore thermal management with model predictive control," in *Proc. European Conf. Ciruit Theory and Design*, Aug. 2009, pp. 711–714.
[35] Computational Optimization Laboratory, "Interior-point convex quadratic programming solver." [Online]. Available: http://web.stanford.edu/~yyye/col.html
[36] "Python software for convex optimization." [Online]. Available: http://cvxopt.org
[37] Nvidia, "Technical brief of NVIDIA Jetson TK1 development kit." [Online]. Available: http://developer.download.nvidia.com
[38] ConnectLand, "Connectland 1504002 heatsink." [Online]. Available: http://uk.connectland.eu/products/fiche/id/360/name/chipset-heatsink

Yue Ma (S'16) is currently working toward his Ph.D. degree in the department of Computer Science and Engineering at University of Notre Dame, Indiana. He received his B.S. degree from Chengdu University of Technology, China, M.S. from University of Electronic Science and Technology of China, China. He research interests include real-time embedded systems, reliable system design, power efficiency and temperature-aware resources management.

Thidapat Chantem (S'05-M'11) received her Ph.D. and Master's degrees from the University of Notre Dame in 2011 and her Bachelor's degrees from Iowa State University in 2005. She is an Assistant Professor of Electrical and Computer Engineering at Virginia Tech. Her research interests include Real-Time Embedded Systems, Energy-Aware and Thermal-Aware System-Level Design, Cyber-Physical System Design, and Intelligent Transportation Systems.

Robert Dick (S'95-M'02) is an Associate Professor of Electrical Engineering and Computer Science at the University of Michigan. He received his Ph.D. degree from Princeton University in 2002 and his B.S. degree from Clarkson University in 1996. He worked as a Visiting Professor at Tsinghua University's Department of Electronic Engineering in 2002, as a Visiting Researcher at NEC Labs America in 1999, and was on the faculty of Northwestern University from 2003–2008. Robert received an NSF CAREER award and won his department's Best Teacher of the Year award in 2004. In 2007, his technology won a Computerworld Horizon Award and his paper was selected as one of the 30 in a special collection of DATE papers appearing during the past 10 years. His 2010 work won a Best Paper Award at DATE. He served as the Technical Program Committee Co-Chair of the 2011 International Conference on Hardware/Software Codesign and System Synthesis, as an Associate Editor of IEEE Trans. on VLSI Systems, and as a Guest Editor for ACM Trans. on Embedded Computing Systems. He is also CEO of the Stryd, Inc., which produces wearable electronics for athletes.

Xiaobo Sharon Hu (S'85-M'89-SM'02-F'16) received her B.S. degree from Tianjin University, China, M.S. from Polytechnic Institute of New York, and Ph.D. from Purdue University, West Lafayette, Indiana. She is Professor in the department of Computer Science and Engineering at University of Notre Dame. Her research interests include real-time embedded systems, low-power system design, and computing with emerging technologies. She has published more than 250 papers in the related areas. She served as Associate Editor for IEEE Transactions on VLSI, ACM Transactions on Design Automation of Electronic Systems, and ACM Transactions on Embedded Computing. She is the Program Chair of 2016 Design Automation Conference (DAC) and the TPC co-chair of 2014 and 2015 DAC. She received the NSF CAREER Award in 1997, and the Best Paper Award from Design Automation Conference, 2001 and IEEE Symposium on Nanoscale Architectures, 2009.