# Online Resource Management for Improving Reliability of Real-Time Systems on "Big–Little" Type MPSoCs

Yue Ma, *Student Member, IEEE*, Junlong Zhou, *Member, IEEE*, Thidapat Chantem, *Senior Member, IEEE*, Robert P. Dick, *Member, IEEE*, Shige Wang, *Senior Member, IEEE*, and Xiaobo Sharon Hu, *Fellow, IEEE*

***Abstract*—Heterogeneous multiprocessor systems on a chips (MPSoCs) consisting of cores with different performance/power characteristics are widely used in many real-time embedded systems, where both soft-error reliability and lifetime reliability are key concerns. Although existing efforts have investigated related problems, they either focus on one of the two reliability concerns or propose time-consuming scheduling algorithms that cannot adequately address runtime workload and environmental variations. This paper introduces an online framework which is adaptive to runtime variations and maximizes soft-error reliability while satisfying the lifetime reliability constraint for soft real-time systems executing on MPSoCs that are composed of high-performance cores and low-power (LP) cores. Based on each core's executing frequency and utilization, the framework performs workload migration between high-performance cores and LP cores to reduce power consumption and improve soft-error reliability. Experimental results based on different hardware platforms show that the proposed approach reduces the probability of failures due to soft errors by at least 17% and 50% on average compared to a number of representative existing approaches that satisfy the same lifetime reliability constraints.**

***Index Terms*—Heterogeneous multiprocessor systems on a chip (MPSoC), lifetime reliability, real-time embedded system, soft-error reliability.**

Manuscript received May 28, 2018; revised September 4, 2018; accepted October 18, 2018. This work was supported in part by NSF under awards CNS-1319904, CNS-1319718, CNS-1319784, National Natural Science Foundation of China under Grant 61802185, and Natural Science Foundation of Jiangsu Province under Grant BK20180470. This paper was recommended by Associate Editor C. Yang. *(Corresponding author: Junlong Zhou.)*

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).

J. Zhou is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: jlzhou@njust.edu.cn).

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).

S. Wang is with General Motors, Warren, MI 48093 USA (e-mail: shige.wang@gm.com).

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

Digital Object Identifier 10.1109/TCAD.2018.2883990

## I. Introduction

TO ADDRESS power/energy concerns, various heterogeneous multiprocessor systems on a chip (MPSoCs) have been introduced [1]. A popular MPSoC architecture that is often used in power/energy-conscious real-time embedded applications is composed of pairs of high-performance (HP) cores and low-power (LP) cores. Following the terminology introduced by ARM [2], we refer to this architecture as the "big–little" architecture. Nvidia's variable symmetric multiprocessing [3] is such an example. Such HP and LP cores present unique performance, power/energy, and reliability tradeoffs, which are investigated in this paper.

Resource management in heterogeneous MPSoCs has been widely studied [4]–[8], but few work targets the big–little architecture [9]–[12]. In this architecture, HP (LP) cores are homogeneous and both HP and LP cores have the same instruction set architecture. However, big–little type MPSoCs may support different execution models. In one model, represented by Nvidia's TK1 [13] and Samsung's Exynos 5410 [14], one HP core is paired with one LP core, and the HP and LP cores in the one pair cannot work simultaneously. In another model, represented by Nvidia's TX2 [15] and NXP's i.MX8 [16], although HP and LP cores can work simultaneously, all HP (all LP) cores must execute at the same frequency. We aim to design a resource management framework that is adaptive to different execution models.

Since many real-time embedded systems are deployed in critical applications and are expensive as well as inconvenient to replace, lifetime reliability due to permanent faults[1] as well as soft-error reliability due to transient faults are important design considerations. Although there exist several efforts that either target soft-error reliability [17]–[19] or lifetime reliability [8], [20]–[23], only a few papers have examined both soft-error reliability and lifetime reliability together [24]–[27]. In addition, runtime workload variations further complicate the problem of improving the system overall reliability.

[1]Intermittent faults are unlikely to be strongly dependent on power consumption and therefore are out of the scope of this paper.

Hence, designing an online approach considering both lifetime reliability and soft-error reliability becomes necessary.

This paper systematically addresses reliability concerns for real-time systems running on big–little type MPSoCs. Since transient faults occur much more frequently than permanent faults [28], we focus on increasing soft-error reliability without sacrificing lifetime reliability. Specifically, we solve the problem of maximizing soft-error reliability while satisfying temperature, real-time, and lifetime reliability requirements. Our problem is motivated by many real world applications, such as mobile devices and in-vehicle infotainment systems [29]. We are particularly interested in developing an online framework to address unavoidable workload and environment variations.

Our online framework, referred to as dynamic reliability improvement framework (`DRIF`), solves the problem outlined above by dynamically and judiciously scaling core frequencies to increase soft-error reliability. By leveraging the power and performance features of the big–little type MPSoCs, we dynamically migrate workload and activate the most power-efficient cores to execute tasks. Meanwhile, in order to reduce the computational overhead to check whether the lifetime reliability caused by a thermal profile is larger than a lifetime reliability constraint, we design a tool, referred to as `LTR-Checker`, which is computational efficient to use at run time.

This paper makes three main contributions.

1) We propose a computationally efficient method to determine whether a given temporal thermal profile would respect the corresponding lifetime reliability threshold.
2) By performing extensive experiments on a hardware platform, we experimentally establish a suitable task migration guideline allowing tasks executed on most power efficient cores.
3) We develop an online framework to maximize soft-error reliability under temperature, real-time, and lifetime reliability constraints by scaling cores' frequencies and selecting the most power efficient cores to execute tasks.

We have implemented and validated `DRIF` on two hardware boards containing Nvidia's TK1 [13] chip and TX2 [15] chip, respectively. Based on the results obtained from running the MiBench benchmark suite [30], we show that `DRIF` increases the no soft error occurring time at least 2 more days than existing approaches.

The rest of this paper is organized as follows. We review related work in Section II. Section III introduces the various system models. We experimentally explore the power features of HP and LP cores, and establish a task migration guideline in Section IV. Section V formulates the problem and provides an overview of our framework. Section VI describes the `LTR-Checker`. Section VII describes `DRIF` in detail. Sections VIII and IX describe our experimental setup and results, respectively. Section X concludes this paper.

## II. Related Work

As a special type of heterogeneous MPSoCs, the big–little type MPSoCs use two types of cores: the LP cores offer high power efficiency while the HP cores provide maximum computing performance [2]. This type of MPSoCs provides flexibility to balance the performance and power, and facilitates ease of use [31]. Since different execution models introduce unique constraints, e.g., HP core and LP core in the same pair cannot work simultaneously, or all HP (all LP) cores must execute at the same frequency, most resource management approaches for heterogeneous MPSoCs are not applicable for the big–little architecture [5], [21], [23], [32], [33]. Focusing on the big–little architecture, Liu *et al.* [9] proposed an iterative approach for mapping multithreaded applications on MPSoCs composing of multiple core types to achieve high performance and power efficiency. Annamalai *et al.* [10] designed a novel technique to dynamically swap threads between HP cores and LP cores and change the core frequency to achieve a high throughput/Watt. Considering the constraints for HP cores and LP cores, Carroll and Heiser [11] investigated the mechanisms for frequency scaling, and proposed a technique to reduce energy consumption. Singla *et al.* [12] designed an online method to predict and reduce power and runtime temperature for big–little type MPSoCs. While the above work considers the specific features of the big–little architecture, none of them focuses on lifetime reliability or soft-error reliability.

There exist several efforts that directly aim to increase soft-error reliability [17], [18], [34], [35] or lifetime reliability [7], [21], [36], [37]. In order to improve soft-error reliability, Zhao *et al.* proposed a method to allocate recoveries for tasks [17], [18] while Nahar and Meyer [38] assigned redundancies to tasks statically. Fan *et al.* proposed a dynamic voltage and frequency scaling (DVFS)-based method to reduce power consumption under soft-error reliability constraint. Although these methods are effective at improving and ensuring soft-error reliability, they usually reduce lifetime reliability with a high operating temperature. For periodic tasks running on an MPSoC, Huang *et al.* [21] proposed an analytical model to estimate lifetime reliability of MPSoCs and a task mapping and scheduling algorithm to guard against aging effects. Bolchini *et al.* [36] dynamically determined the most effective mapping of tasks to minimize network-on-chip energy consumption and maximize lifetime reliability. Das *et al.* [7] proposed a machine learning-based algorithm to handle inter- and intra-application variations and reduce peak temperature and thermal cycling. These methods are designed to increase lifetime reliability but weaken soft-error reliability.

Our proposed framework considers soft-error reliability and lifetime reliability, both of which have not typically been examined together. The work by Das *et al.* [24] aims to jointly improve soft-error reliability and lifetime reliability by mapping tasks to all cores and scaling core frequencies. However, their solution is too computationally intensive to use at run time. Kapadia and Pasricha [25] proposed a framework to optimize performance and energy. Although transient and permanent faults are considered, their work does not increase reliability but only focuses on reducing power under lifetime reliability and soft-error reliability constraints. Zhou *et al.* [26] proposed an offline technique to maximize system availability by allocating replications of tasks and determining the core

frequency statically. Although these works consider both lifetime reliability and soft-error reliability, they are offline approaches and ignore the specific features of big–little type MPSoCs. In this paper, we focus on big–little type MPSoCs and propose to maximize soft-error reliability under lifetime reliability constraint.

## III. System Models

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

### A. Hardware Model

We consider on big–little type MPSoCs with $n$ HP and $m$ LP cores. We assume that both HP cores and LP cores support DVFS and have multiple frequency levels [13], [15]. A core dissipates static power when it is idle and consumes additional active power when it performs operations [33]. Both active and static power are related to the core's frequency. Let the utilization of a core in a given time interval $|t|$ be $u = (|t_a|/|t|)$, where $|t_a|$ is the amount of time that the core performs operations [33]. A core's utilization is commonly used to estimate real-time performance and soft-error reliability.

We consider two execution models of big–little MPSoCs in this paper. In the first execution model, referred to as Hetero-Paired model and represented by Nvidia' TK1 [13] and Samsung's Exynos 5410 [14], HP cores and LP cores are paired, and the paired HP core and LP core cannot be active simultaneously. In the second execution model, referred to as Homo-Grouped model and represented by Nvidia's TX2 [15] and NXP's i.MX8 [16], all cores can work simultaneously, but HP (LP) cores must execute at the same frequency. There exist other execution models, where HP and LP cores can run simultaneously with their own core frequencies, but such models are not widely supported by MPSoCs.

### B. Task Model

We assume that MPSoCs execute independent periodic tasks with soft deadlines, such as those found in multimedia and communication applications. A task is associated with a tuple $\tau_i = \{d_i, e_i^H, e_i^L\}$, where $d_i$ is the deadline, and $e_i^H$ and $e_i^L$ represent the worst-case execution time when running on an HP core and LP core, respectively. Generally $e_i^H \leq e_i^L$. Since all the jobs of the $i$th task have the same properties, $\tau_i$ also denotes the jobs of the $i$th task. Tasks on each core are scheduled according to a real-time scheduling policy, such as earliest deadline first or rate monotonic scheduling [39]. In this paper, we adopt a mapping approach, where tasks are assigned to cores at design time to balance the workload of cores [40]. We guarantee the real-time constraint by ensuring that the utilization of each core is lower than utilization bound for schedulability [8], [41].

### C. Soft-Error Reliability

In this paper, we aim to maximize reliability in the presence of soft errors caused by transient faults. The soft-error reliability of a single core in a time interval is the probability that soft errors occur during the time interval [26]

$$r(f, t) = e^{-\lambda(f)\times u\times|t|}. \tag{1}$$

The $f$ is the core frequency, $|t|$ is the length of time interval, and $u$ is the core's utilization in this time interval. $\lambda(f)$ is the average fault rate depending on $f$ [26]

$$\lambda(f) = \lambda_0 \times 10^{\frac{d(f_{\max}-f)}{f_{\max}-f_{\min}}}. \tag{2}$$

$\lambda_0$ is the average faults rate at highest core frequency. $f_{\min}$ and $f_{\max}$ are the minimum and maximum core frequency and $d$ ($d > 0$) is a hardware specific constant that indicates the sensitivity of fault rates to frequency scaling. This model indicates that improving core frequency is effective in improving soft-error reliability.

For a big–little type MPSoC with $n$ active HP cores and $m$ active LP cores, the soft-error reliability in the $i$th time interval, $t_i$, is

$$R(t_i) = \prod_{j=1}^{n} r_j^{\mathrm{HP}}(f_j, t_i) \times \prod_{j=1}^{m} r_j^{\mathrm{LP}}(f_j, t_i) \tag{3}$$

where $r_j^{\mathrm{HP}}(f_j, t_i)$ and $r_j^{\mathrm{LP}}(f_j, t_i)$ are the soft-error reliability of the $j$th HP (LP) core in the time interval $t_i$. The aim of this paper is to maximize soft-error reliability of the MPSoC in each time interval.

### D. Lifetime Reliability

Lifetime reliability, which is typically measured by the mean-time-to-failure (MTTF), is dependent on multiple wear-out effects [23]. For the sake of simplicity, we consider electromigration as the primary source of permanent faults in this paper. Other device fault mechanisms can be incorporated using the sum-of-fault rate model [22], [24]. Since the tasks are executed periodically, the temperature variance with respect to time will be also periodical after the system stabilization, so we assume the thermal profiles are same in each task set's hyperperiod, hp. Based on the thermal profile in a hyperperiod, the MTTF can be calculated by

$$\mathrm{MTTF} = |\mathrm{hp}| \times \sum_{i=0}^{\infty} e^{-(i\times A)^{\beta}} \tag{4}$$

where $|\mathrm{hp}|$ is the length of the hyperperiod and $\beta$ is the slope parameter in the Weibull distribution [21]. $A$ is a temperature-related parameter. If one hyperperiod can be divided by $p$ time intervals of the same length, and the operating temperature is constant in each time interval, we calculate $A$ as

$$A = \sum_{i=1}^{p} \frac{|t|}{\alpha(T_i)} \tag{5}$$

where $|t|$ and $T_i$ are the length of the time interval and the temperature at the $i$th time interval, respectively. $\alpha(T_i)$ relates to the arrival rate of permanent faults and depends on the hardware and temperature $T_i$ [21].

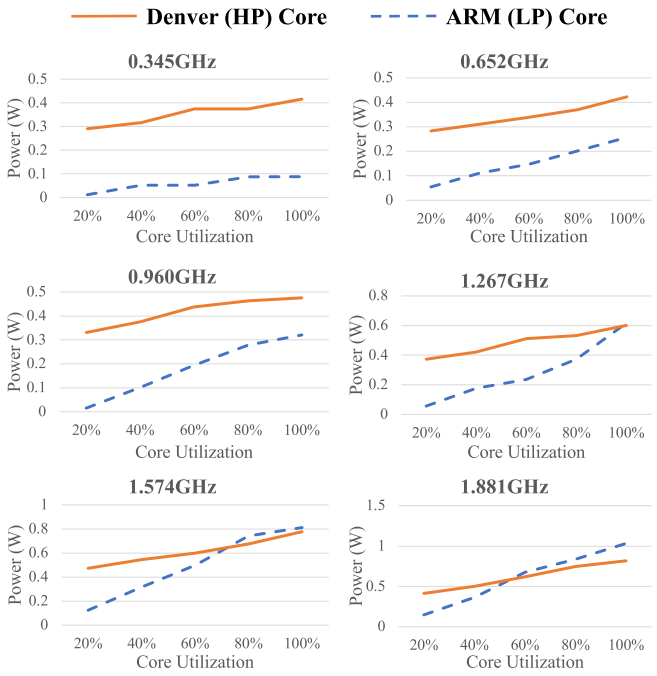


Fig. 1. Power consumption of an HP (Denver) core and an LP (ARM) core under different utilization and frequency levels.

## IV. Empirical Study: Power Consumption of Cores

In this section, we describe the big–little type MPSoCs consisting of HP and LP cores, especially explore their unique power features. We first observe that executing tasks on an LP core may consume more power and energy than executing on an HP core. We provide a measurement-based method to quantitatively compare the power and energy consumption of HP and LP cores. Based on this method and the measurement results, we establish a suitable task mapping and migration guideline to migrate tasks between cores and reduce a chip's power consumption.

Whereas the primary goal of big–little MPSoCs is to reduce power consumption by executing a light workload on the LP cores, an LP core may consume more power than an HP core. To totally capture the power consumption behavior of big–little MPSoCs, we have conducted a series of measurement-based experiments. We measure the power consumption of the HP core and LP core[2] in Nvidia's TX2 [15]. We use FLUKE AC/DC current clamp meters [43] and National Instruments USB-6216 data acquisition system [44] to acquire power consumption when cores execute at different core frequencies and at different utilizations.

To generally evaluate the power features of HP and LP cores, we propose a measurement-based method to quantitatively compare power consumption of HP and LP cores. This method measures and compares the power consumption of cores with different frequencies and utilizations, and the comparison results can guide the mapping of tasks. A low utilization means that the workload is light, a core consumes less active power, and the leakage power may be dominated. In order to maintain the core's utilization at a specific level, we develop a feedback-based tool which can maintain the core's utilization at a specific value.

[2]Note that TX2 is composed of ARM Cortex A57 cores geared for multithreading, and Nvidia' Denver cores for high single-thread performance with dynamic code optimization [42]. In this measurement, we only consider single-thread applications for TX2, therefore the Denver core is an HP core and the ARM core is an LP core.

TABLE I
Task Mapping and Migration Guideline

| Utilization | Core Frequency (in GHz) | | | | | |
|---|---|---|---|---|---|---|
| | 1.881 | 1.574 | 1.267 | 0.960 | 0.652 | 0.345 |
| 100% | HP | - | - | LP | LP | LP |
| 80% | - | - | LP | LP | LP | LP |
| 60% | - | LP | LP | LP | LP | LP |
| 40% | - | LP | LP | LP | LP | LP |
| 20% | LP | LP | LP | LP | LP | LP |

The measured power consumptions are illustrated in Fig. 1. The results show that for any core frequency, both HP and LP cores have a higher power consumption with a heavier workload. However, LP cores are not always power efficient. The LP core consumes less power than the HP core only when the core frequency is low and the workload is light. For other platforms, such as Nvidia's TK1 [13], we have similar observations that the LP core has a lower power than the HP core only when the utilization and core frequency are low [27]. One possible reason to explain this phenomenon is that the HP and LP core have different microarchitectures, such as on TX2. Meanwhile, although HP and LP cores on TK1 have the same microarchitecture, the transistors in the HP core and LP core have different threshold voltages. The LP core consumes low leakage power but requires high voltage to operate at high frequencies. On the contrary, the HP core can work at high frequency with a low voltage. The measurement results reveal that in order to reduce power consumption of MPSoCs, we should keep the workload light in the LP cores, and it is necessary to migrate tasks between HP and LP cores if cores' utilizations vary at run time.

Based on the data collected from our extensive experiments, we can establish a suitable task mapping and migration guideline guiding the selection of cores for executing workload to balance the power consumption and performance. This guideline indicates that whether the LP core or the HP core consumes less power for each given core frequency and core utilization. With this guideline, we should map and migrate tasks to the core consuming less power. As an example, Table I presents the guideline for Nvidia TX2. In this table, "HP" ("LP") indicates the HP (LP) core is more power efficient with the corresponding core frequency and utilization, so the workload should be executing on an HP (LP) core. Note that due to small variations in ambient temperature, as well as chip operating voltage and current, the power consumption may vary slightly even for exactly the same workload. Therefore, it is insufficient to conclude that a core always consumes less power when its measured power is lower than that of another core by a small amount. We treat two measured power values as the same if their difference is smaller than 0.1 W, which is the resolution of our sensors. In Table I, "–" indicates that the difference in power consumption of an HP core and an LP core is smaller than this threshold. In this case, workload can run either on an HP core or an LP core.

In this paper to dynamically improve reliability, we will use this guideline to migrate tasks between HP and LP cores at run time to guarantee tasks are always executed on the most power efficient cores. This task migration reducing power consumption and temperature allows the cores to execute at a high core frequency and achieves a high soft-error reliability.

## V. Problem Formulation and Framework Overview

In this section, we first formulate the problem addressed in this paper and then describe our solution DRIF at high level.

### A. Problem Formulation

The problem that we aim to solve is motivated by applications, such as in-vehicle infotainment systems. For such systems, tasks are expected to complete before their deadlines, and both lifetime and soft-error reliability are critical to guarantee the safety of human drivers and passengers [29]. At the same time, the infotainment and other in-vehicle computational subsystems should be power efficient especially for electric vehicles [45]. Furthermore, the workload in these systems can vary significantly at run time due to variations in input data and the environment.

Before formulating the problem, we first introduce two definitions.

*Definition 1:* A sampling window (SW) is defined as a time interval during which the temperature is constant.

*Definition 2:* A profiling window (PW) is composed of multiple equal-length SWs.

We determine the core frequencies and cores' workloads for each SW, and the PW is used to estimate lifetime reliability. The soft-error reliability, frequency, utilization, and operating temperature of the $j$th HP (LP) core at the $i$th SW are denoted by $r(\mathrm{SW}_i, \mathrm{HP}_j)$ ($r(\mathrm{SW}_i, \mathrm{LP}_j)$), $f(\mathrm{SW}_i, \mathrm{HP}_j)$ ($f(\mathrm{SW}_i, \mathrm{LP}_j)$), $u(\mathrm{SW}_i, \mathrm{HP}_j)$ ($u(\mathrm{SW}_i, \mathrm{LP}_j)$), and $T(\mathrm{SW}_i, \mathrm{HP}_j)$ ($T(\mathrm{SW}_i, \mathrm{LP}_j)$).

Assume that a PW is composed of $p$ SWs and the MPSoC has $n$ HP cores and $m$ LP cores.[3] Our objective is to maximize the system-level soft-error reliability in each PW

$$R = \prod_{i=1}^{p} \left( \prod_{j=1}^{n} r\big(\mathrm{SW}_i, \mathrm{HP}_j\big) \times \prod_{j=1}^{m} r\big(\mathrm{SW}_i, \mathrm{LP}_j\big) \right) \tag{6}$$

$$\text{s.t.} \begin{cases} T(\mathrm{SW}_i, \mathrm{HP}_j) \le T_{\mathrm{th}}, \forall \mathrm{SW}_i, \forall \mathrm{HP}_j & (7) \\ T(\mathrm{SW}_i, \mathrm{LP}_j) \le T_{\mathrm{th}}, \forall \mathrm{SW}_i, \forall \mathrm{LP}_j & (8) \\ u(\mathrm{SW}_i, \mathrm{HP}_j) \le u_{\mathrm{th}}, \forall \mathrm{SW}_i, \forall \mathrm{HP}_j & (9) \\ u(\mathrm{SW}_i, \mathrm{LP}_j) \le u_{\mathrm{th}}, \forall \mathrm{SW}_i, \forall \mathrm{LP}_j & (10) \\ \mathrm{MTTF}(\mathrm{TP}(\mathrm{PW})) \ge \mathrm{MTTF}_{\mathrm{th}}. & (11) \end{cases}$$

The first two constraints require the temperature of both HP and LP cores are less than the thresholds $T_{\mathrm{th}}$ in any SW. Note that this temperature constraint also limits the power consumption of the system. The third and forth constraints capture the real-time requirement, where $u_{\mathrm{th}}$ is the upper bound on utilization to satisfy schedulability. The last constraint requires the MTTF resulting from the thermal profile, TP(PW), to be not less than a threshold $\mathrm{MTTF}_{\mathrm{th}}$. For soft real-time systems, temporarily violating the real-time and lifetime reliability constraints is acceptable, but the temperature constraint must be satisfied to avoid thermal throttling.

Different execution models of big–little type MPSoCs introduce different execution related constraints. For the Hetero-Paired execution model, the paired HP core and LP core cannot work simultaneously. If the $j$th HP core is paired with the $j$th LP core, one of them must be idle, i.e.,

$$f\big(\mathrm{SW}_i, \mathrm{HP}_j\big) \times f\big(\mathrm{SW}_i, \mathrm{LP}_j\big) = 0. \tag{12}$$

We assume that a core whose frequency is 0 is powered-off. For the Homo-Grouped execution model, all HP (LP) cores should have the same core frequency, i.e.,

$$\begin{cases} f(\mathrm{SW}_i, \mathrm{HP}_j) = f(\mathrm{SW}_i, \mathrm{HP}_{j+1}), \forall j & (13) \\ f(\mathrm{SW}_i, \mathrm{LP}_j) = f(\mathrm{SW}_i, \mathrm{LP}_{j+1}), \forall j. & (14) \end{cases}$$

Our framework is applicable to both execution models and dynamically improves the soft-error reliability under the temperature, real-time, and lifetime reliability constraints in each PW.

In order to solve the formulated problem, there are two main challenges that we need to overcome: 1) since the history (i.e., tasks' execution times) does not always reflect the future, it is possible for the constraints to be violated when using history-based predictions and 2) a highly efficient algorithm is needed to avoid excessive overhead. We address these challenges by proposing an online framework to: 1) obtain system runtime status and 2) dynamically migrate tasks between cores, power off idle cores, and determine core frequencies based on the system status in history.

### B. Overview of Reliability Improvement Framework

As stated earlier in this paper, to better respond to workload and environmental changes that are unavoidable in real-time embedded systems, we aim to develop an online approach to solve the problem defined in (6)–(11) by taking into consideration of execution models given in (12) or (13)–(14). The basic idea of our framework, DRIF, is to incrementally solve the optimization problem by using the history of system states in the previous PW. The system state includes which cores are active and each active core's frequency, operating temperature, and utilization. Note that our method can be easily applied to any arbitrary history window size. DRIF consists of three main components: a schedule generator (SG), which is triggered at the beginning of each PW, a schedule executor (SE), which is triggered at the beginning of each SW, and a state collector (SC), which collects the system state in each SW (see Fig. 2).

DRIF works as follows. In each SW, SC collects and saves the system state. At the end of each PW, the system state during this PW is sent to SG. Based on the state information, SG then generates a solution, called schedule, which specifies cores' workloads and frequencies for each SW in the next PW (see Section VII-A). The migration guideline given in Table I is used by SG to migrate tasks between cores to achieve a lower power consumption as well as operating temperature. In order to reduce the computational cost, SG relies

[3] $m$ is equal to $n$ for MPSoCs with Homo-Grouped execution model.

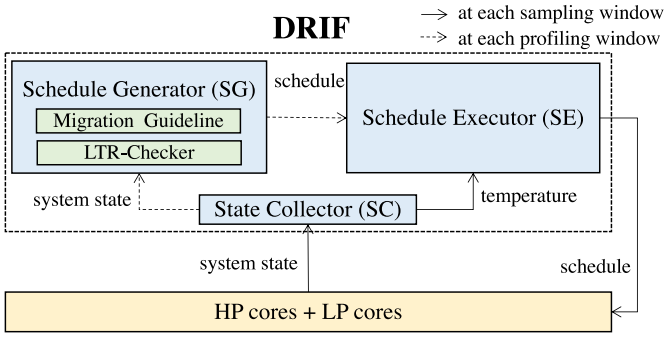


Fig. 2. High-level overview of DRIF.

on LTR-Checker to efficiently check whether the lifetime reliability constraint is satisfied. In each SW, SE either adopts the schedule generated by SG or modifies the schedule to adapt to runtime variations (see Section VII-B).

We highlight the effectiveness of DRIF. First of all, DRIF is adaptive to different types of big–little MPSoCs and different number of cores and/or pairs of cores. Meanwhile, considering that the workload in systems may vary at runtime, DRIF periodically obtain the status of each core. Based on the obtained runtime status, DRIF determines the most appropriate cores to execute tasks satisfying the real-time, lifetime reliability and operating temperature constraints. In order to reduce the computational overhead, we propose heuristics to periodically migrate tasks and tune core frequencies in linear time. Note that the execution order of tasks in each core can be determined by some existing scheduling policies. DRIF is adaptive to and can work on any scheduling policy, such as rate monotonic and earliest deadline first [39]. The details of our DRIF are elaborated in the next section.

## VI. LTR-Checker: Tool to Check Lifetime Reliability Constraint

In this section, we design a tool LTR-Checker, which computational efficiently checks whether the lifetime reliability caused by a given thermal profile in a task set's hyperperiod is larger than a prespecified constraint, $\text{MTTF}_{\text{th}}$. Calculating MTTF by using (4) is extremely time consuming and may not be practical to use at runtime. Hence, the target of LTR-Checker is reducing the runtime computational overhead by allowing some calculations are operated offline.

We first introduce a concept called super hyperperiod, sp, which is a set of multiple adjacent hyperperiods. Let the length of a super hyperperiod be |sp|, and $|\text{sp}| = |\text{hp}| \times k$, where $k$ is a positive integer. Since one super hyperperiod is composed of multiple adjacent hyperperiods and thermal profiles are same in each super hyperperiod, the lifetime reliability can also be expressed as

$$\text{MTTF} = |\text{sp}| \times \sum_{i=0}^{\infty} e^{-(i \times A^{\star})^{\beta}} \tag{15}$$

where

$$A^{\star} = \sum_{i=1}^{k \times p} \frac{|t|}{\alpha(T_i)}. \tag{16}$$

For a given thermal profile in the hyperperiod, LTR-Checker checks whether the corresponding MTTF is larger than $\text{MTTF}_{\text{th}}$. LTR-Checker reduces the online computational overhead by operating the accumulation offline and only calculating $A^{\star}$ online.

The aim of the offline part in LTR-Checker is to find a threshold for $A^{\star}$, referred to as $A^{\star}_{\text{th}}$, such that if $A^{\star} \leq A^{\star}_{\text{th}}$, the corresponding MTTF is larger than $\text{MTTF}_{\text{th}}$. We first arbitrarily determine the length of super hyperperiod |sp|. Since |hp| is usually in seconds and $\text{MTTF}_{\text{th}}$ is in years, setting |sp| to months can satisfy that |sp| can be evenly divided by any possible |hp|. After determining the value of |sp|, we can find the threshold $A^{\star}_{\text{th}}$ such that

$$|\text{sp}| \times \sum_{i=0}^{\infty} e^{-(i \times A^{\star}_{\text{th}})^{\beta}} = \text{MTTF}_{\text{th}}. \tag{17}$$

If the $A^{\star}$ caused by a thermal profile is smaller than $A^{\star}_{\text{th}}$, the corresponding system's MTTF is larger than $\text{MTTF}_{\text{th}}$.

The online part of LTR-Checker calculates $A^{\star}$ based on the thermal profile in a hyperperiod. With the determined |sp|, we first find the relationship between $A$ [in (5)] and $A^{\star}$, which is described in Lemma 1.

*Lemma 1:* If one super hyperperiod is composed of $k$ hyperperiods, i.e., $|\text{sp}| = k \times |\text{hp}|$, then $A^{\star} = A \times k$.

*Proof:* Since thermal profiles are same in each hyperperiod, each hyperperiod's $i$th time interval has the same temperature, i.e., $T_i = T_{i+p} = \cdots = T_{i+kp}$. Furthermore, $\alpha(T_i) = \alpha(T_{i+p}) = \cdots = \alpha(T_{i+kp})$. Hence

$$A^{\star} = \sum_{i=1}^{kp} \frac{|t|}{\alpha(T_i)} = k \times \sum_{i=1}^{p} \frac{|t|}{\alpha(T_i)} = A \times k. \tag{18}$$

■

Since |sp| is arbitrarily determined offline and |hp| is constant for a given task set, we only need to calculate $A$ in order to obtain $A^{\star}$. $A$ can be obtained by using (5), and its computational overhead only depends on the value of |hp|, which is much smaller than |sp| and $\text{MTTF}_{\text{th}}$. Comparing to obtain MTTF directly by using (4) and (5), the online operation of LTR-Checker is only obtaining $A$ by using (5). Hence, LTR-Checker dramatically reduces the online computational overhead and it can be easily used even when the computational resources are limited. In DRIF, we require the length of the PW is multiple of the length of the task set's hyperperiod, and the SG utilizes the LTR-Checker to determine whether a given operating temperature can guarantee the lifetime reliability constraint.

## VII. Design of Reliability Improvement Framework

We provide the details of our framework DRIF to improve the soft-error reliability under the temperature, real-time, and lifetime reliability constraints.

**Algorithm 1** SG for Homo-Grouped MPSoCs

1: $hf$ ($lf$): the cores with high (low) core frequencies
2: $l(lf, SW_i)$: frequency level of $lf$ cores at sampling window $SW_i$
3: $TP_j$: thermal profile in the $q^{th}$ profiling window
4: **procedure** GENERATORHOG($Sc(PW_j)$, $St(PW_j)$)
5: **if** MTTF($TP_j$) $< MTTF_{th}$ **then**
6: **for** each sampling window $SW_i$ **do**
7: **if** $u(l(hf, SW_i) - 1) < u_{th}$ **then**
8: $l(hf, SW_i) = l(hf, SW_i) - 1$
9: **else if** $u(l(hf, SW_i) - 1) < u_{th}$ **then**
10: $l(lf, SW_i) = l(lf, SW_i) - 1$
11: **end if**
12: **end for**
13: **else**
14: **for** each sampling window $SW_i$ **do**
15: **if** $T(l(lf, SW_i) + 1) < T_{th}$ **then**
16: $l(H, SW_i) = l(H, SW_i) + 1$
17: **end if**
18: **end for**
19: **end if**
20: **for** each sampling window $SW_i$ **do**
21: $Sc^{\star}(SW_i) \leftarrow$migrate workload based on TABLE I
22: **end for**
23: $Sc(PW_{j+1}) \leftarrow \{Sc^{\star}(SW_1), \ldots, Sc^{\star}(SW_p)\}$
24: **end procedure**

**Algorithm 2** SG for Hetero-Paired MPSoCs

1: $\rho_k$: the $k^{th}$ active core
2: $l(\rho_k, SW_i)$: HP's frequency level at sampling window $SW_i$
3: $TP_j$: thermal profile in the $q^{th}$ profiling window
4: **procedure** GENERATORHEP($Sc(PW_j)$, $St(PW_j)$)
5: **if** MTTF($TP_j$) $< MTTF_{th}$ **then**
6: **for** each sampling window $SW_i$ **do**
7: Sort core with their core frequencies
8: **for** $\rho_k$ (starting form the core with high frequency) **do**
9: **if** $u(l(\rho_k, SW_i) - 1) < u_{th}$ **then**
10: $l(\rho_k, SW_i) = l(\rho_k, SW_i) - 1$
11: break
12: **end if**
13: **end for**
14: **end for**
15: **else**
16: **for** each sampling window $SW_i$ **do**
17: Sort core with their core frequencies
18: **for** $\rho_k$ (starting form the core with low frequency) **do**
19: **if** $T(l(\rho_k, SW_i) + 1) < T_{th}$ **then**
20: $l(\rho_k, SW_i) = l(\rho_k, SW_i) + 1$
21: break
22: **end if**
23: **end for**
24: **end for**
25: **end if**
26: **for** each sampling window $SW_i$ **do**
27: $Sc^{\star}(SW_i) \leftarrow$migrate workload based on TABLE I
28: **end for**
29: $Sc(PW_{j+1}) \leftarrow \{Sc^{\star}(SW_1), \ldots, Sc^{\star}(SW_p)\}$
30: **end procedure**

### A. Schedule Generator

The goal of SG is to generate a schedule, i.e., each core's workload and frequency, for the next PW based on the system status in the current PW. Although it is possible to use an optimization solver to generate an optimal schedule for the problem defined in (6)–(11), such a solver would be too time consuming for online use. Instead, we design a computational effective heuristic migrating tasks and dynamically scaling core frequencies.

As pointed out earlier, we assume that the workload has already been mapped and the workload is balanced between cores. Considering the runtime variations of workload, SG determines the frequencies of all cores to maximize soft-error reliability and meet all constraints in (7)–(11) by considering the execution models of big–little MPSoCs given in (12) or (13), (14).

Before we present the algorithm in SG, we first introduce some concepts. System state, St($\text{PW}_j$), denotes the state in the PW $\text{PW}_j$, which includes the utilization, frequency, and operating temperature of each core in the SWs of $\text{PW}_j$. St($\text{SW}_i$), a subset of St($\text{PW}_j$), represents the state in the SW $\text{SW}_i$. System schedule, Sc($\text{PW}_j$), specifies each core's workload and frequency in all SWs in $\text{PW}_j$. Similarly, Sc($\text{SW}_i$) represents schedule in the SW $\text{SW}_i$.

SG is invoked at the end of each PW and takes St($\text{PW}_j$) and Sc($\text{PW}_j$) as inputs. SG generates a schedule for Homo-Grouped MPSoCs (in Algorithm 1) or for Hetero-Paired MPSoCs (in Algorithm 2), respectively. We provide the details to generate a schedule for Homo-Grouped MPSoCs first. The idea is that we check whether the lifetime reliability constraint is satisfied, and try to increase core frequencies if the lifetime reliability is larger than its constraint, otherwise, reduce core frequencies (in lines 5–19). Since all HP (LP) cores run at the same core frequency, we use $hl$ ($lf$) to represent cores running at high (low) core frequencies. For each SW, if the system status in the previous PW, St($\text{PW}_j$), violates the lifetime reliability constraint, SG reduces the core frequencies of cores running at high frequency if doing so does not violate the real-time constraint (in lines 7 and 8). Otherwise, reduce the core frequencies of cores with low core frequency if not violate the real-time constraint (in lines 9 and 10). Meanwhile, if St($\text{PW}_j$) meets the lifetime reliability constraint, SG increases frequencies for cores with low core frequency to improve soft-error reliability under the temperature constraint (in lines 14–18). After determining core frequencies, SG migrates tasks between cores to reduce the power consumption and temperature (in lines 20–22). We provide the details of task migration in Algorithm 3. After determining core frequencies and migrating tasks between cores, the schedule for the next PW, Sc($\text{PW}_{j+1}$), is generated (in line 23). The computational complexity to determine the core frequencies for Homo-Grouped MPSoCs is $O(p)$, where $p$ is the number of SWs in a PW.

SG generates a schedule for Hetero-Paired MPSoCs in Algorithm 2. If the system status in the previous PW, St($\text{PW}_j$), violates the lifetime reliability constraint, SG tries to reduce the core frequency for the core which executes at the highest frequency if doing so does not violate the real-time constraint (in lines 6–14). On the contrary, if St($\text{PW}_j$) satisfies the lifetime reliability constraint, SG increases the core frequency of cores with low core frequency under the temperature constraint (in lines 16–24). Similar to Homo-Grouped MPSoCs,

**Algorithm 3** Migrate Workload

1: $Ty(\rho_j)$: the type of $\rho_j$, its HP or LP
2: $u(\rho_j, SW_i)$: $\rho_j$'s utilization at $SW_i$
3: $u(\rho_j, W)$: $\rho_j$'s utilization if executing workload $W$
4: $e_k^{\rho_p}$: the execution time of task $\tau_k$ on core $\rho_p$
5: **procedure** MIGRATE($Sc(SW_i)$, $St(SW_i)$, TABLE I)
6: **if** Homo-Grouped MPSoCs **then**
7: **for** each core ($\rho_j$) **do**
8: $\tau_k$: the task on $\rho_j$ with shortest execution time
9: $\rho_p$: the lowest utilization core at different type of $\rho_j$
10: Search TABLE I with $u(\rho_j, SW_i)$ and $f(\rho_j, SW_i)$
11: $\mathcal{T} \leftarrow$ the type of the most power efficient core
12: **while** $Ty(\rho_j) \neq \mathcal{T}$ **do**
13: **if** $u(\rho_p) + \frac{e_k^{\rho_p}}{d_k} < u_{th}$ **then**
14: Migrate $\tau_k$ to core $\rho_p$
15: **end if**
16: $\mathcal{T} \leftarrow$ search TABLE I
17: **end while**
18: **end for**
19: **end if**
20: **if** Hetero-Paired MPSoCs **then**
21: **for** each active core $\rho_j$ **do**
22: $\mathcal{T} \leftarrow$ search TABLE I with $u(\rho_j)$ and $f(\rho_j)$
23: $W$: the workload on $\rho_j$
24: $\rho_p$: $\rho_j$'s paired core
25: **if** $Ty(\rho_j) \neq \mathcal{T}$ and $u(W, \rho_p) < u_{th}$ **then**
26: Migrate all workload to $\rho_p$ paired core
27: **end if**
28: **end for**
29: **end if**
30: **for** each core $\rho_j$ **do**
31: **if** $\rho_j$'s workload is empty **then**
32: Power off $\rho_j$
33: **end if**
34: **end for**
35: **end procedure**

SG migrates tasks (in lines 26–28) and finally generates a new schedule $\text{Sc}(\text{PW}_{j+1})$ (in line 29). The computational complexity of Algorithm 2 is $O(p \times (n+m) \times \log(n+m))$, where $p$ is the number of SWs in a PW, and $n$ and $m$ are the number of HP cores and LP cores, respectively.

We provide the details on how to migrate tasks and select power efficient cores to execute tasks are in Algorithm 3. This task migration algorithm is called by Algorithms 1 and 2 at each SW, and its inputs are the migration guideline given in Table I, the system status, and schedule at each SW. The key idea is that we search the migration guideline with each core's utilization and frequency, and migrate tasks based on the search results. For the Homo-Grouped MPSoCs, for a core, $\rho_j$, if the migration guideline indicates we should tune $\rho_j$'s utilization to save power, we migrate the task with shortest execution to an LP or HP core (in lines 6–19). We iteratively migrate tasks between cores until the results of search migration guideline match the types of all cores. For the Hetero-Paired MPSoCs, the paired HP and LP cores work exclusively. Hence, if tasks are ready optimally mapped to each pair initially, we only need to select the HP or LP core to use for each pair. If the searching results from the task migration guideline do not match the type of the active core $\rho_j$, migrate all tasks on $\rho_j$ to its paired core if doing so does not violate the real-time constraint (in lines 20–29). For both Homo-Grouped and Hetero-Paired MPSoCs, if a core's workload is empty, power off this core to save energy (in lines 30–34). For the Homo-Grouped MPSoCs, the computational complexity of Algorithm 3 is $O(\wp \times (m+n))$, where $\wp$, $m$, $n$ are the number of tasks, HP cores, and LP cores, respectively. For Hetero-Paired MPSoCs, the complexity is $O(m+n)$.

### B. Schedule Executor

The SE, determines the active cores' frequencies at the beginning of each SW. A straightforward approach is to simply follow the schedule generated by SG. However, since the schedule $\text{Sc}(\text{PW}_{j+1})$ is generated based on the system status $\text{St}(\text{PW}_j)$, but the utilization in the PW $\text{PW}_j$ can be different from that in the $\text{PW}_{j+1}$, $\text{Sc}(\text{PW}_{j+1})$ may actually violate some or all of the constraints during run time. For soft real-time systems, it is acceptable to temporarily violate the real-time and lifetime reliability constraints in (9)–(11) as they can be compensated in the next PW. However, violating the temperature constraint may either cause timing faults or unexpected throttling. Therefore, SE should be designed to avoid the occurrence of such a case.

SE adjusts core frequency for each core. At the beginning of each SW, SE receives the initial temperatures from SC, which is the temperature of the previous SW, and gets the cores' frequencies from $\text{Sc}(\text{PW}_{j+1})$. We can statically design a table that for all possible initial temperatures and core frequencies. This table indicates the worst-case temperature in an SW by assuming the core utilization is 100%. SE checks whether the worst-case temperature can remain below the thermal threshold. If not, we reduce the core frequency one level lower than that specified in the schedule $\text{Sc}(\text{PW}_{j+1})$. Since we establish such a table statically, the computational complexity of SE is $O(1)$.

## VIII. Experimental Setup

To evaluate the proposed DRIF, we conducted experiments to compare with two representative approaches. In this section, we present the platforms, workloads, and the frameworks used for comparison in our experiments.

### A. Comparison Targets

We compared the performance of DRIF to two representative frameworks. The multiobjective optimization of system reliability (MOO) finds the Pareto-optimization of soft-error reliability and lifetime reliability by using a genetic algorithm [24]. Since the genetic algorithm-based solver is too costly to be used at runtime, core frequencies are determined offline and cannot be changed online. In order to evaluate the benefits of migrating tasks between cores, we compare DRIF with a framework, called simplified DRIF (S-DRIF), which scales core frequencies as in DRIF, but does not migrate tasks between cores.

Three metrics are considered in the comparison. The probability of failures (PoF) due to soft errors quantifies the soft-error reliability. The PoF is defined as $1 - R$, where $R$

TABLE II
TASKS' EXECUTION TIMES ON TK1

| Tasks | Execution time | |
|---|---|---|
| | HP ARM Core | LP ARM Core |
| qsort | 145 ms | 145 ms |
| blowfish | 150 ms | 152 ms |
| crc32 | 195 ms | 196 ms |

is the system-level soft-error reliability. An approach achieving a lower PoF is the same as achieving a higher soft-error reliability. We used the percentage of feasible solutions for real-time constraint (FS-RT) to describe the capability of satisfying real-time constraint. In experiments, the jobs of each task are periodically released. We checked which job meeting its deadline and the percentage of FS-RT is quantified as the ratio of the number of jobs meeting its deadline over the total number of all jobs. Similarly, the percentage of feasible solution for lifetime reliability (FS-LTR) constraint describes the capability of satisfying lifetime reliability. In experiments, we utilized `LTR-Checker` to check whether the lifetime reliability is satisfied at each PW. The percentage of FS-LTR is quantified as the ratio of the number of PWs achieving a higher lifetime reliability than the lifetime reliability constraint over the total number of PWs.

### B. Experimental Platforms

The experiments are conducted on two boards containing Nvidia's TK1 [13] and TX2 [15] chip, respectively. The TK1 chip provides four HP cores and one LP core, but the HP cores and the LP core cannot work simultaneously. Hence, the TK1 chip is a Hetero-Paired type MPSoC, and it only provides one HP–LP core pair. In our experiments, the workload for TK1 is designed to be light enough to fit on one HP or LP core. The TX2 chip includes two HP cores (with Nvidia's Denver microarchitecture [42]) and four LP cores (with ARM Cortex A57 microarchitecture). Hence, TX2 chip is a Homo-Grouped type MPSoC. Note that we only consider single-thread tasks, so the Denver core has a better performance than the ARM core [42].

We obtained the chip's operating temperature by reading their integrated thermal sensors. Note that although TK1 and TX2 only report one CPU temperature, it is enough to show that `DRIF` can achieve a lower temperature and guarantee the temperature constraint. For both HP and LP cores in TK1, we use the core frequencies 1.092 GHz, 0.96 GHz, 0.828 GHz, 0.696 GHz, and 0.564 GHz. For TX2, we select the core frequencies 1.881 GHz, 1.574 GHz, 1.267 GHz, 0.960 GHz, 0.652 GHz, and 0.345 GHz.

### C. Workloads

We now discuss the tasks set for experiments on TK1 and TX2. Considering the low performance of cores in TK1, we chose three tasks from Mibench benchmark suite [30] and measured their execution times when the core's frequency is 1.092 GHz (see Table II). TK1 only provides one 1 HP–LP core pair, so tasks execute either on the HP core or the LP core. For experiments on TX2, we used two ARM cores and one Denver core to execute eight tasks from Mibench [30]. We first measured the execution times of the tasks on an ARM and Denver core with the highest core frequency (see Table III). Based on the measurements, we mapped these tasks to ARM and Denver cores and balanced the workloads of cores (see Table IV). Note that although TX2 provides four ARM cores and two Denver cores, we only used one Denver core and two ARM cores because the workload is light. If allocating the selected tasks to three ARM cores and/or two Denver cores, the workload of each core is such light that a core can always execute at the highest frequency. Meanwhile, we aim at independent tasks and the soft-error reliability achieved by `DRIF` is related to a cores utilization but independent to the number of cores. Hence, executing tasks on two ARM cores and one Denver core is sufficient to validate the capability of `DRIF` in improving soft-error reliability.

TABLE III
TASKS' EXECUTION TIMES ON TX2

| Tasks | Execution time | |
|---|---|---|
| | Denver Core | ARM Core |
| cjpeg | 24 ms | 33 ms |
| qsort | 49 ms | 69 ms |
| dijkstra | 47 ms | 64 ms |
| blowfish | 26 ms | 52 ms |
| susan | 52 ms | 78 ms |
| stringsearch | 2 ms | 3 ms |
| crc32 | 30 ms | 75 ms |
| patricia | 12 ms | 16 ms |

TABLE IV
TASK ALLOCATION FOR TX2

| Tasks | Mapping to |
|---|---|
| cjpeg | ARM Core 0 |
| qsort | ARM Core 0 |
| dijkstra | ARM Core 1 |
| blowfish | ARM Core 1 |
| susan | Denver Core 0 |
| stringsearch | Denver Core 0 |
| crc32 | Denver Core 0 |
| patricia | Denver Core 0 |

We designed two task groups. In the first group, tasks are frame-based and share the same period and deadline. For experiments on TX2, tasks' periods and deadlines are 150 , 200 , 250 , and 300 ms, and for experiments on TK1, they are 700 , 800 , 900 , and 1000 ms. In the second group, a task's deadline and period are set to be the same but random in the ranges between 150 –200 ms, 200 –250 ms, 250 –300 ms for TX2, and for TK1, the ranges are 700 –800 ms, 800 –900 ms, and 900 –1000 ms. We used the deadline-monotonic scheduling policy to schedule tasks, where a task with shorter deadline is assigned a higher priority and executed earlier [39]. Also, change from tasks to jobs to be consistent. Such setups ensure that tasks are schedulable, and represent multiple workloads ranging from heavy to light.

## IX. EXPERIMENTAL RESULTS

In this section, we examine the performance of the proposed `DRIF` compared to the `S-DRIF` and `MOO`.

### A. Experiments on TK1 Chip

We first validated our approach on a TK1 chip with Hetero-Paired execution model. We compared the proposed `DRIF`

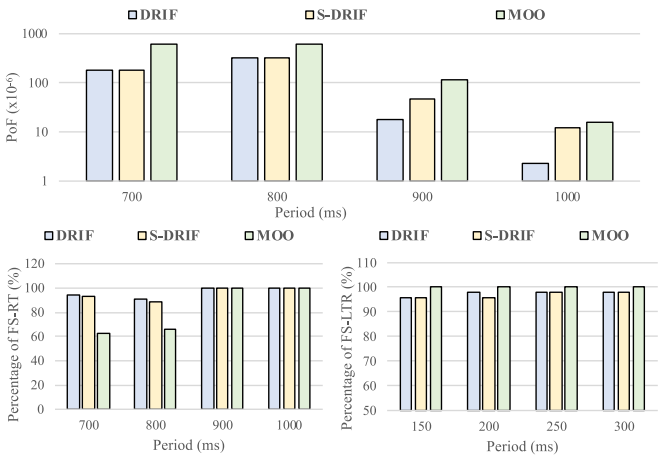


Fig. 3. PoFs due to soft errors and percentage of feasible solutions for a frame-based task set running on TK1.

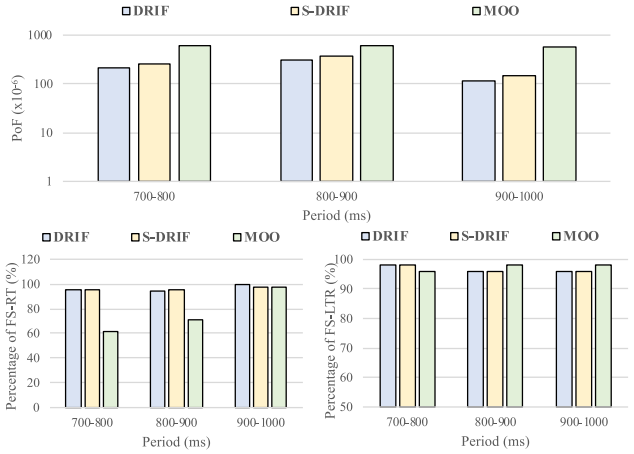


Fig. 4. PoFs due to soft errors and percentage of feasible solutions for a general periodic task set running on TK1.

with MOO and S-DRIF to determine whether DRIF can improve soft-error reliability without violating temperature, real-time, and lifetime reliability constraints.

Fig. 3 shows the experimental results when tasks are frame-based. DRIF and S-DRIF have similar performance when the workload is heavy, but DRIF achieves a lower PoF than MOO and S-DRIF in all the cases. The PoF of DRIF is 97.89%, 95.64%, 37.9%, and 18.89% of S-DRIF when the period is 700 , 800 , 900 , and 1000 ms, respectively. This reduced PoF guarantees the system can work without soft errors at least 2 min more than S-DRIF, and up to 10 h. Meanwhile, since our task migration considers the real-time and lifetime reliability constraints, the percentages of FS-RT and FS-LTR of DRIF, S-DRIF and MOO are close, especially when the workload is light. For the soft-error reliability, the PoF of DRIF is only 29.18%, 51.21%, 16.29%, and 15.04% of MOO. It means that the system can work successfully without soft errors 1.1 , 0.4 , 12.7 , and 100.8 h more than MOO, respectively.

We extended the experiment to validate DRIF for a general periodic task set, where tasks' periods and deadlines are equal but randomly generated in different ranges (see Fig. 4). The average PoF of DRIF is 81% of S-DRIF and 51% of MOO, which translates to DRIF allowing the system to successfully

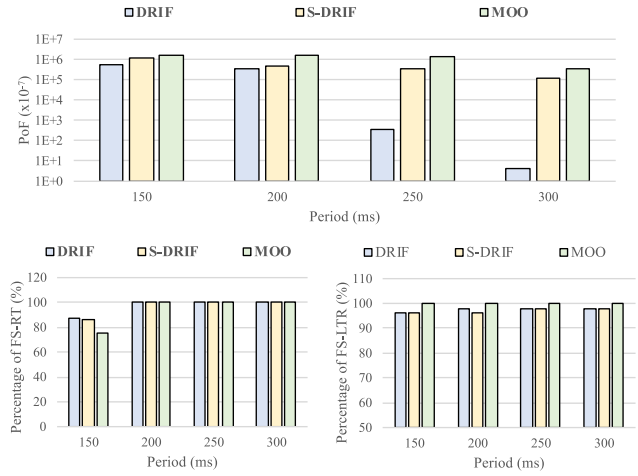


Fig. 5. PoFs due to soft errors and percentage of feasible solutions for a frame-based task set running on TX2.

work for 17 min more than S-DRIF on average, and 63 min more than MOO on average. Comparing to the results in Fig. 3, DRIF provides less benefits when tasks have different periods. The reason is that the workload in each SW varies dramatically, and DRIF guarantees the lifetime reliability constraint with a low core frequencies, which limits the performance in improving soft-error reliability. However, DRIF is still a better approach than S-DRIF and MOO, and achieves a lower PoF.

We measured the time and power consumption of DRIF on an ARM core. DRIF consumes less than 1 ms to complete and we cannot observe power changes when operating DRIF because the resolution of our power measurement tool is about 0.1 W. Based on these measurements, we claim that the time and power consumption of DRIF on TK1 can be ignored.

We also compared DRIF with a brute force search-based approach which finds the optimal solution at each SW. This approach, although can guarantee the highest soft-error reliability at each SW, is computation intensive and cannot be used at the runtime. The execution time of this approach is about 30 s if running on the TK1's HP core. Compared to this approach, the computation time of DRIF is less than 1 ms even if one PW has 100 SWs. Since both approaches determine core frequencies for each PW, which is typically in minutes, the brute force search may not be a good choice to use at runtime.

Although the brute force search-based approach can find optimal solutions at each SW, it is too computational complicated to apply at runtime. On the contrary, DRIF determines the core frequencies at each PW by tuning the operating core frequencies one level at one time. Our experiments show that DRIF can find the best solution starting from the fifth PW. Since the length of a PW is in minutes, not finding the best solution in the first five PWs (less than 10 min) has negligible effect on lifetime reliability and soft-error reliability.

### B. Experiments on TX2 Chip

We conducted experiments on TX2 chip to evaluate the performance of DRIF in the platform with Homo-Grouped execution model. On this platform, DRIF scales core frequencies and migrates tasks to increase soft-error

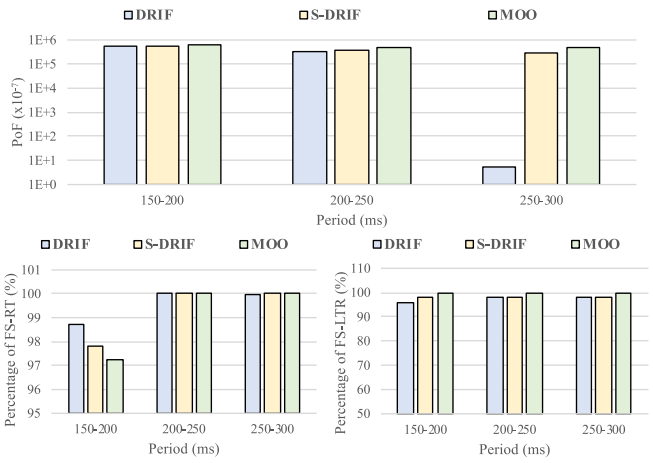


Fig. 6. PoFs due to soft errors and percentage of feasible solutions for a general periodic task set running on TX2.

reliability under temperature, real-time and lifetime reliability constraints.

Similar to the experiments on TK1, we validated DRIF for: 1) a frame-based task set (see Fig. 5) and 2) a general periodic task set (see Fig. 6). For the frame-based task set, the PoF of DRIF is 47.25%, 81.95%, 0.1%, and 0.003% of S-DRIF when the period is 150, 200, 250, and 300 ms, respectively. This low PoF guarantees the system can successfully work 158 h more than S-DRIF on average and up to 24 days. Thanks to the dynamic task migration, DRIF dynamically selects the most appropriate cores to execute tasks. DRIF can also dynamically power off any idle cores to reduce power consumption and allow active cores running at high core frequency. Hence, the benefits of DRIF are clearer than the experiments on TK1 in Fig. 3. Comparing to MOO, DRIF achieves a lower PoF in all cases, and leads to a system that can successfully work about 6.6 days more than MOO on average and up to 26 days. In terms of satisfying real-time and lifetime reliability constraints, both DRIF and S-DRIF achieve a similar percentage of FS-RT and FS-LTR to MOO especially when the workload is light.

Fig. 6 shows the performance of DRIF when the workload is a general periodic task set. The PoF of DRIF is about 98%, 86%, and 0.001% of S-DRIF when periods of tasks in ranges 150–200 ms, 200–250 ms, and 250–300 ms, respectively. It means that DRIF guarantees the system successfully work without soft errors 7.6 days more than S-DRIF on average, and up to 22.8 days. Meanwhile, the soft-error reliability improvement of DRIF over MOO is similar to that over S-DRIF. Comparing to MOO, DRIF increases the system's successful execution time about 7.6 days on average, and up to 22.8 days. Finally, the execution time of DRIF is less than 1 ms either on the ARM core or the Denver core. The power consumption of DRIF on TX2, similar as on TK1, is also too small to be observed. In summary, the above experiments confirm that our approach DRIF has a better performance in improving soft-error reliability in all cases, especially when the workload is light.

## X. Conclusion

Focusing on two execution models of big–little type MPSoCs, we proposed a DRIF to maximize soft-error reliability under temperature, real-time, and lifetime reliability constraints. We designed a computational efficient tool to check whether the lifetime reliability caused by a thermal profile is larger than a prespecified constraint. In order to reduce power consumption, we empirically studied the power features of the HP and LP cores and established a task migration guideline to indicate the most appropriate and power efficient core to execute tasks. Based on these contributions, our framework dynamically migrates tasks between cores and adjusts the core frequencies to satisfy all constraints. The results on chips supporting different execution models show that our approach is effective in increasing soft-error reliability under constraints compared to other representative approaches. As future work, we plan to extend our approach to more general task models and consider MPSoCs with GPU.

## References

[1] W. Wolf, A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 550–561, Oct. 2008.

[2] ARM. *Big.LITTLE Technology: The Future of Mobile*. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[3] Nvidia. *Variable SMP (4-Plus-1) a Multi-Core CPU Architecture for Low Power and High Performance*. [Online]. Available: https://www.nvidia.com/content/PDF/tegra_white_papers

[4] A. S. Hartman, D. E. Thomas, and B. H. Meyer, "A case for lifetime-aware task mapping in embedded chip multiprocessors," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Oct. 2010, pp. 145–154.

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

[6] L. Huang, F. Yuan, and Q. Xu, "On task allocation and scheduling for lifetime extension of platform-based MPSoC designs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 789–800, Dec. 2011.

[7] A. Das *et al.*, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proc. Design Autom. Conf.*, Jun. 2014, pp. 1–6.

[8] Y. Ma, T. Chantem, R. P. Dick, and X. S. Hu, "Improving system-level lifetime reliability of multicore soft real-time systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 6, pp. 1895–1905, Jun. 2017.

[9] G. Liu, J. Park, and D. Marculescu, "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems," in *Proc. Int. Conf. Comput. Design*, Oct. 2013, pp. 54–61.

[10] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu, "An opportunistic prediction-based thread scheduling to maximize throughput/watt in AMPs," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2013, pp. 63–72.

[11] A. Carroll and G. Heiser, "Unifying DVFS and offlining in mobile multicores," in *Proc. Int. Conf. Real Time Embedded Technol. Appl. Symp.*, Apr. 2014, pp. 287–296.

[12] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Proc. Design Autom. Test Europe*, Mar. 2015, pp. 1–6.

[13] Nvidia. (2018). *Technical Brief of NVIDIA Jetson TK1 Development Kit*. Accessed: Oct. 2018. [Online]. Available: http://developer.download.nvidia.com

[14] Samsung. (2018). *Samsung Exynos 5 Octa (5410) Mobile Processor*. Accessed: Oct. 2018. [Online]. Available: http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_5_Octa_5410.html

[15] Nvidia. (2018). *Jetson Tegra X2*. Accessed: Oct. 2018. [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2

[16] NXP. (2018). *I.MX 8 Family ARM Cortex-A53, Cortex-A72, Virtualization, Vision, 3D Graphics, 4K Video*. Accessed: Oct. 2018. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/i.mx-applications-processors/i.mx-8-processors:IMX8-SERIES
[17] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technology," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2009, pp. 63–70.
[18] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *Proc. Int. Conf. Real Time Embedded Technol. Appl. Symp.*, Apr. 2011, pp. 285–294.
[19] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *Proc. Design Autom. Conf.*, Jun. 2011, pp. 381–386.
[20] 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. Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2009, pp. 169–180.
[21] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling on MPSoC platform," in *Proc. Design Autom. Test Europe*, Mar. 2009, pp. 51–56.
[22] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," in *Proc. Design Autom. Test Europe*, Mar. 2013, pp. 689–694.
[23] 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 Autom. Test Europe*, Mar. 2013, pp. 1373–1378.
[24] 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 Autom. Test Europe*, Mar. 2014, pp. 1–6.
[25] N. Kapadia and S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *Proc. Design Autom. Test Europe*, Mar. 2015, pp. 1060–1065.
[26] J. Zhou, X. S. Hu, Y. Ma, and T. Wei, "Balancing lifetime and soft-error reliability to improve system availability," in *Proc. Asia South Pac. Design Autom. Conf.*, Jan. 2016, pp. 685–690.
[27] 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. Design Autom. Test Europe*, Mar. 2017, pp. 1–6.
[28] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 316–328, May 2010.
[29] G. Macario, M. Torchiano, and M. Violante, "An in-vehicle infotainment software architecture based on Google Android," in *Proc. Int. Symp. Ind. Embedded Syst.*, Jul. 2009, pp. 257–260.
[30] University of Michigan. (2018). *MiBench*. Accessed: Oct. 2018. [Online]. Available: http://vhosts.eecs.umich.edu/mibench
[31] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu, "High-performance and energy-efficient mobile Web browsing on big/little systems," in *Proc. Int. Conf. High Perform. Comput. Archit.*, Feb. 2013, pp. 13–24.
[32] P. Pop, K. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Sep. 2007, pp. 233–238.
[33] Y. Fu, N. Kottenstette, C. Lu, and X. Koutsoukos, "Feedback thermal control of real-time systems on multicore processors," in *Proc. Int. Conf. Embedded Softw.*, Oct. 2012, pp. 113–122.
[34] M. Fan, Q. Han, S. Liu, and G. Quan, "On-line reliability-aware dynamic power management for real-time systems," in *Proc. Int. Symp. Qual. Electron. Design*, Mar. 2015, pp. 361–365.
[35] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, Oct. 2011, pp. 247–256.
[36] C. Bolchini *et al.*, "Run-time mapping for reliable many-cores based on energy/performance trade-offs," in *Proc. Design Autom. Conf.*, Jun. 2013, pp. 58–64.
[37] A. Das, A. Kumar, and B. Veeravalli, "Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs," in *Proc. Design Autom. Test Europe*, Mar. 2014, pp. 1–6.
[38] B. Nahar and B. H. Meyer, "RotR: Rotational redundant task mapping for fail-operational MPSoCs," in *Proc. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, Oct. 2015, pp. 21–28.
[39] C. L. Liu and J. W. Layland, "Scheduling algorithm for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[40] 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 Symp. VLSI*, May 2015, pp. 79–82.
[41] Y. Fu *et al.*, "Feedback thermal control for real-time systems," in *Proc. Int. Conf. Real Time Embedded Technol. Appl. Symp.*, Apr. 2010, pp. 111–120.
[42] Nvidia Development Blog. (2018). *Nvidia Jetson TX2 Delivers Twice the Intelligence to the Edge*. Accessed: Oct. 2018. [Online]. Available: https://devblogs.nvidia.com/parallelforall/jetson-tx2-delivers-twice-intelligence-edge/
[43] FLUKE. (2018). *80i-110s AC/DC Current Clamp*. Accessed: Oct. 2018. [Online]. Available: http://www.fluke.com/fluke/iden/accessories/current-clamps/80i-110s.htm?pid=55352
[44] National Instruments. (2018). *NI USB-6216 BNC*. Accessed: Oct. 2018. [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/207100
[45] J. Moreno, M. E. Ortuzar, and J. W. Dixon, "Energy-management system for a hybrid electric vehicle, using ultracapacitors and neural networks," *IEEE Trans. Ind. Electron.*, vol. 53, no. 2, pp. 614–623, Apr. 2006.

**Yue Ma** (S'16) received the B.S. degree from the Chengdu University of Technology, Chengdu, China, and the M.S. degree from the University of Electronic Science and Technology of China, Chengdu. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA.

His current research interests include real-time embedded systems, reliable system design, power efficiency, and temperature-aware resource management.

**Junlong Zhou** (S'15–M'17) received the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2017.

He was a Visiting Scholar with the University of Notre Dame, Notre Dame, IN, USA, from 2014 to 2015. He is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His current research interests include real-time embedded systems, cloud computing and IoT, and cyber physical systems.

Dr. Zhou has been an Associate Editor for the *Journal of Circuits, Systems, and Computers* since 2017.

**Thidapat Chantem** (S'05–M'11–SM'18) received the bachelor's degree from Iowa State University, Ames, IA, USA, in 2005 and the master's and Ph.D. degrees from the University of Notre Dame, Notre Dame, IN, USA, in 2011.

She is an Assistant Professor of electrical and computer engineering with Virginia Tech, Blacksburg, VA, USA. Her current research interests include real-time embedded systems, energy-aware and thermal-aware system-level design, cyber-physical system design, and intelligent transportation systems.

**Robert P. Dick** (S'95–M'02) received the B.S. degree from Clarkson University, Potsdam, NY, USA, in 1996 and the Ph.D. degree from Princeton University, Princeton, NJ, USA, in 2002.

He is an Associate Professor of electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA. He was a Visiting Professor with the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2002, as a Visiting Researcher with NEC Labs America, Princeton, NJ, USA in 1999, and was on the faculty of Northwestern University, Evanston, IL, USA, from 2003 to 2008. He is also CEO of the Stryd, Inc., Boulder, CO, USA, which produces wearable electronics for athletes.

Dr. Dick was a recipient of the NSF CAREER Award and the Department's Best Teacher of the Year Award in 2004. In 2007, his technology won a Computerworld Horizon Award, and the Best Paper Award at DATE for his research in 2010. His paper was selected as one of the 30 in a special collection of DATE papers appearing during the past 10 years. 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 the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and as a Guest Editor for *ACM Transactions on Embedded Computing Systems*.

**Shige Wang** (S'02–M'05–SM'11) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 2004.

He is a Staff Research Scientist with General Motors Research and Development, Warren, MI, USA. His current research interests include system modeling and analysis, software architecture for parallel processing in automated driving systems, and embedded real-time control systems.

**Xiaobo Sharon Hu** (S'85–M'89–SM'02–F'16) received the B.S. degree from Tianjin University, Tianjin, China, the M.S. degree from the Polytechnic Institute of New York, New York, NY, USA, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA.

She is Professor with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA. Her current research interests include real-time embedded systems, LP system design, and computing with emerging technologies. She has published over 250 papers in the above areas.

Dr. Hu was a recipient of the NSF CAREER Award in 1997, and the Best Paper Award from Design Automation Conference in 2001, and the IEEE Symposium on Nanoscale Architectures, in 2009. She served as an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, *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 DAC, in 2014 and 2015, respectively.