An On-Line Framework for Improving Reliability of Real-Time Systems on "Big-Little" Type MPSoCs

Yue Ma[1], Thidapat Chantem[2], Robert P. Dick[3], Shige Wang[4], and X. Sharon Hu[1]

[1]Department of CSE, University of Notre Dame, Notre Dame, IN 46656, USA, {yma1,shu}@nd.edu
[2]Department of ECE, Virginia Polytechnic Institute and State University, Arlington, VA, 22203, tchantem@vt.edu
[3]Department of EECS, University of Michigan, Ann Arbor, MI 48109, dickrp@umich.edu
[4]General Motors R&D, Warren, MI, 48090, shige.wang@gm.com

***Abstract*—Heterogeneous MPSoCs consisting of cores with different performance/power behaviors are widely used in many power-constrained real-time systems. Both soft-error reliability and lifetime reliability are key concerns in such systems. Although existing work have investigated related problems, they either focus on one of the two reliability concerns or propose complicated scheduling algorithms that cannot adequately address run-time workload and environment variations. This paper introduces an on-line heuristic to maximize soft-error reliability while satisfying a lifetime reliability constraint for soft real-time systems executed on MPSoCs composed of high-performance cores and low-power cores. Based on the run-time cores' frequencies and utilizations, the heuristic performs workload migration between the high-performance cores and low-power cores to achieve improved soft-error reliability. Experimental results from both a hardware platform and a simulator show that the proposed algorithm reduces the probability of faults by at least 30% compared to a number of representative existing approaches while satisfying the same lifetime reliability constraints.**

***Keywords*—Soft-error reliability; Lifetime reliability; Heterogeneous MPSoC; Real-time embedded system.**

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 as "big-little" architecture. Nvidia's variable symmetric multiprocessing (vSMP) also falls into this category [3]. Such HP–LP core pairs present unique performance, power/energy, and reliability tradeoffs, which are investigated in this paper.

Since many real-time embedded systems that use MPSoCs are deployed in critical applications and are expensive as well as inconvenient to replace, life-time reliability (LTR) due to permanent faults[1] as well as soft-error reliability (SER) due to transient faults are important design considerations. Decreasing device dimensions and increasing transistor counts generally decrease SER and LTR. Techniques for reducing power, energy and/or temperature also impact the two reliability metrics in different ways. Run-time workload variations further complicate the problem of improving the overall system reliability.

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 [4], we focus on increasing SER without sacrificing LTR. Specifically, we solve the following problem: maximize SER while satisfying real-time requirements, an upper bound on LTR, and other constraints including energy and temperature. Such problems can be found in many real world applications such as mobile devices and in-vehicle infotainment systems [5]. We are particularly interested in developing an on-line framework to address unavoidable workload and environment variations.

Most existing work either targets SER [6]–[8] or LTR [9]–[13]. A few recent papers have examined both SER and LTR together [14]–[17]. Both Chou et al. [14] and Huang et al. [15] proposed a fault-aware technique to recover from faults. Although transient and permanent faults are considered, these efforts do not focus on increasing SER and LTR. The work by Das et al. aimed to jointly improve SER and LTR by mapping tasks to all cores and scaling core frequencies [16]. However, their solution is too computationally intensive to use online. Zhou et al. proposed a technique to maximize system availability by allocating replicated tasks and determining the core frequency statically [17]. This off-line approach cannot capture run-time variations.

We introduce an on-line framework, referred to as `DRIF` (for Dynamic Reliability Improvement Framework) to solve the problem identified above. Core frequencies are dynamically scaled to increase SER. By exploiting the power and performance features of the big-little type MPSoCs, we dynamically activate the most power efficient core in each HP–LP core pair to reduce power and temperature.

Our paper makes three main contributions. (i) By performing experiments on a real hardware platform, we experimentally establish suitable migration points for moving tasks between HP and LP cores based on power and performance features. (ii) We propose a computationally efficient method to determine whether a given temporal thermal profile would result in the corresponding LTR to be larger than a threshold. (iii) We develop an on-line framework to maximize SER under real-time, thermal, power, and LTR constraints by scaling cores' frequencies and selecting the most power efficient cores to

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

978-3-9815370-8-6/17/$31.00

execute tasks.

We have implemented and validated our algorithm both on a simulator and on a hardware board (Nvidia's Jetson TK1 [18]). Based on the results obtained from running the MiBench benchmark suite [19], our algorithm decreases SER as measured by the probability of transient faults by at least 30% when compared with existing work.

II. Big-Little MPSoc And Core Power Features

In this section, we describe the big-little type MPSoCs, exploit the power features, and establish a guideline for suitable migration points for migrating tasks to cores.

The big-little type MPSoCs are composed of multiple HP cores and LP cores. In a typical big-little MPSoC, a HP core and LP core are configured in a pair, but are activated exclusively[2]. Given the fact that all pairs are identical and work independently, we assume all pairs have the same power features and analyze the power features of one pair[3].

Whereas the primary goal of big-little type of MPSoCs is to reduce power consumption by executing a light workload on the LP cores, a LP core may sometimes consume more power than a HP core when the cores are running the same workload at the same high frequency settings [18]. Considering the difficulties in direct measurement of a core's power consumption, one may use the steady-state temperatures without external heatsink as an indirect method to measure power consumption. When using steady-state temperature to measure power consumption, measurements must be performed with the same ambient temperature and heat transfer conditions. A higher steady-state temperature indicates higher power consumption. Although our measurements were taken on the Nvidia TK1 board, the methodology is applicable to HP and LP cores in other MPSoCs as well.

Fig. 1 shows the measured temperatures representing the power of Nvidia's TK1 board [18]. As shown, the LP core reaches a higher steady-state temperature under a heavy workload and a high frequency due to the high dynamic power. Due to the small variations in ambient temperature, chip operating voltage and current, the power consumption may vary, leading to the variations in the measured temperature. As such, it is not sufficient to conclude that a HP core consumes less power when its temperature is lower than that of a LP core by a small amount. We treat two temperatures are same if their difference is smaller than 0.5 °C, which is the resolution of our thermal sensors. We only register power savings when the temperature of the LP core exceeds that of the HP core by this threshold.

Based on our measurements, we can determine a set of workload migration points when the workload should be moved to a core different to balance the power consumption and the performance. As an example, Table I presents the migration points for Nvidia TK1 board derived from our measurements. It shows the cores that the workload should be running on given the core frequency and utilization. In this table, "HP" and "LP" indicate the workload should be migrating to a HP or LP core, and "-" indicates the workload should remain on its current core.

[2]HP and LP cores can run simultaneously with the ARM global task scheduling, but this is not widely supported by all MPSoCs.

[3]For the Nvidia vSMP with four HP cores and only one LP core, we consider a light workload that can run on a single core.

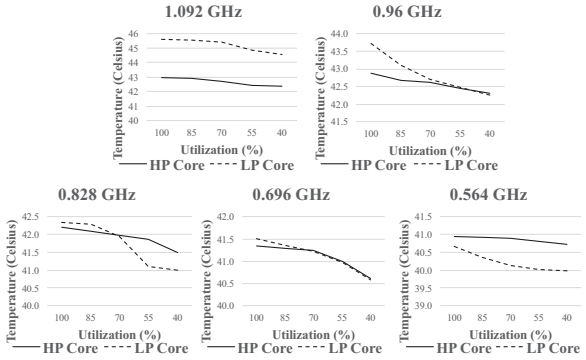

Fig. 1. The steady-state temperature of a HP core and a LP core under different utilization and frequency levels, while other cores are powered off.

While migrating workload between a HP core and a LP core can reduce MPSoC power consumption, the migration introduces additional cost in time and power. In this paper, the power cost of a migration is negligible compared with the power consumed to execute the workload. The time cost for a migration varies from application to application. Our measurements of running the applications from Mibench [19] on Nvidia TK1 board, as shown in Table II, indicate the time cost can be significant and differ from the 2 ms upper bound for powering on the core and stabilizing the voltage rails [3]. This suggests that a system developer should obtain the migration time cost of a target application during design and allow the migration only when the cost does not lead to violation of the real-time constraints.

III. System Model and Problem Formulation

In this section, we first describe the LTR model and propose a computationally efficient method to check whether adopting a thermal profile may cause the system to violate its LTR constraint. Then, we describe other models and formally describe the problem to be solved in this paper.

A. Lifetime reliability

Lifetime reliability, which is typically measured by the mean-time-to-failure (MTTF), is dependent on multiple wear-out effects [12]. For the sake of simplicity, we consider electromigration (EM) as the primary source of permanent faults in

TABLE I. Guidelines for Allocating Workloads

Util.	1.092 GHz	0.960 GHz	0.828 GHz	0.696 GHz	0.564 GHz
100%	HP	HP	-	-	-
85%	HP	-	-	-	LP
70%	HP	-	-	-	LP
55%	HP	-	-	-	LP
40%	HP	-	-	-	LP

TABLE II. Measured Timing Overhead of Migration

bitcount	dijkstra	jpeg	sha	stringsearch
8.13 ms	6.24 ms	3.40 ms	1.13 ms	0.14 ms

this paper. Other device fault mechanisms can be incorporated using the sum-of-fault rate model [11], [16]. Based on the thermal profile of each hyper-period, the established MTTF is

$$MTTF = \Delta hp \times \sum_{i=0}^{\infty} e^{-(i \times A)^{\beta}}, \quad (1)$$

where A depends on the hardware and the thermal profile in the hyper-period, and its computation is provided [10]. Δhp is the length of the hyper-period.

In order to check whether a given thermal profile can cause a system's LTR constraint, $MTTF_{th}$, to be violated, one method is to calculate the MTTF by using Eq. (1) [17]. However, two variables, A and Δhp, depend on the thermal profile, and are the main sources of complex computation. We tackle this challenge by deriving a new formula that is dependent on only one variable.

We introduce a concept called *super hyper-period*, shp, which is a set of multiple adjacent hyper-periods. Since $MTTF_{th} >> \Delta hp$, it is safe to assume that $MTTF_{th}$ is evenly divided by Δhp [10]. Also, we can find a super hyper-period whose length is constant, smaller than $MTTF_{th}$, and evenly divided by any possible lengths of the hyper-period. If the length of this super hyper-period is Δshp, for any lengths of the hyper-period, there must be a corresponding k that $\Delta shp = \Delta hp \times k$.

By using super hyper-period, the LTR can be expressed as

$$MTTF = \Delta shp \times \sum_{i=0}^{\infty} e^{-(i \times A^{\star})^{\beta}}, \quad (2)$$

where $A^{\star}$ depends on the hardware and the thermal profile in the super hyper-period. We prove that $A^{\star} = A \times k$, but omit the details due to page limit. Hence, if $\Delta shp \times m = MTTF_{th}$, the formula to check the MTTF is

$$\sum_{i=0}^{\infty} e^{-(i \times A^{\star})^{\beta}} > m. \quad (3)$$

In Eq. (3), only the variable $A^{\star}$ depends on the thermal profile.

We propose using Eq. (3) as part of a two-phase process to check whether a thermal profile may allow the system to meet its LTR constraint. During the design phase, we arbitrary determine the Δshp. Since Δhp is usually in seconds, setting Δshp to years can satisfy that Δshp is evenly divided by any possible Δhp. Then, we calculate $m = \frac{MTTF_{th}}{\Delta shp}$ and find a threshold $A^{\star}_{th}$. If the $A^{\star}$ of a thermal profile is smaller than $A^{\star}_{th}$, the corresponding MTTF is larger than $MTTF_{th}$. During the run-time phase, for a given thermal profile in the hyper-period, we calculate A to yield $A^{\star}$ as $A^{\star} = k \times A$, $k = \frac{\Delta shp}{\Delta hp}$. Although the calculation performed during the design phase is complicated, it only needs to be carried out once and allows for a more-efficient online computation. In fact, the overhead of our method in the run-time phase only depends on the length of the hyper-period, which is relatively small, making our approach suitable for online use.

B. Soft-error reliability

In this paper, we aim to maximize reliability in the presence of transient faults caused by soft errors. The soft-error reliability of a single core in a time interval is the probability that no faults result from soft errors during that time interval,

$$r = e^{-\lambda(f) \times u \times \Delta t}. \quad (4)$$

$\lambda(f)$ is the average fault rate which highly depends on the core frequency (f). Δt is the time interval and u is the core's utilization. If a hyper-period has p time intervals and a system has n HP-LP core pairs, the SER at a hyper-period is

$$R = \prod_{i=0}^{p} \prod_{j=0}^{n} r(i, C_j), \quad (5)$$

where $r(i, C_j)$ is the SER of the j^{th} core pair in the i^{th} time interval. The aim of this paper is to maximize the average SER among hyper-periods under the LTR constraint.

C. Hardware and workload model

Based on the discussion in Section II, we focus on big-little type MPSoCs. We assume the system has n HP cores and n LP cores. We pair one HP core with a LP core, and inside each pair, the HP core and the LP core work exclusively. We also assume that for each pair, the HP core and the LP core can work at the same frequency [18], but they have different dynamic and leakage power consumptions.

In this paper, we consider independent periodic tasks with soft deadlines. A task that misses deadline is immediately terminated. We adopt a hybrid approach for mapping tasks to cores. Specifically, at design time, tasks are allocated to each HP–LP core pair (similar to partitioned scheduling [20]), and during run time tasks can be migrated within a HP–LP pair. Migration between different pairs is not allowed. This hybrid approach can adapt to dynamic run-time variations and has been used in different areas [6], [13].

D. Problem formulation

We formulate our problem motivated by applications like mobile and in-vehicle infotainment systems. Infotainment systems have soft real-time, power, LTR, and SER constraints, and are typically run on operating systems [5]. The overhead of the operating system cannot be ignored and it may change at run time.

Before formulating the problem, we first introduce a concept called sampling window (W). A sampling window is defined as a time interval in which the temperature of each sampling window is constant, and migration is not allowed inside a sampling window. At the i^{th} sampling window, for the j^{th} HP–LP core pair, the HP core ($c(W_i, C_j) =$ "HP") or the LP core ($c(W_i, C_j) =$ "LP") executing tasks and the frequency is $f(W_i, C_j)$. If a hyper-period is composed by p sampling windows and the system has n HP–LP core pairs, the problem is to maximize the SER,

$$\max R = \prod_{i=0}^{p} \prod_{j=0}^{n} r(W_i, C_j), \quad (6)$$

where $r(W_i, C_j)$ is the SER of the j^{th} pair in the i^{th} sampling window. For each HP–LP core pair (represented by the j^{th}

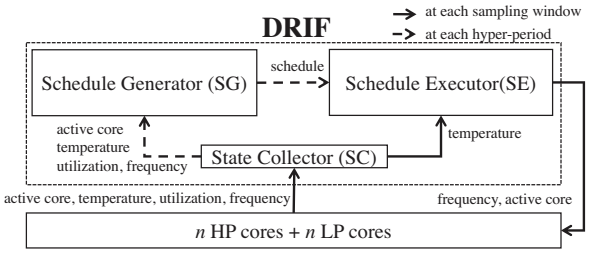

Fig. 2. High-level overview of DRIF.

pair), the solution to Eq. (6) must satisfy the following

$$\begin{cases} u(W_i, C_j) = u_{os}(W_i, C_j) + u_\tau(W_i, C_j) \leq u_{max}, & (7) \\ p(W_i, C_j) \leq p_{budget}, & (8) \\ MTTF(TP_j) > MTTF_{th}, & (9) \\ \max\{T(W_i, C_j)\} \leq T_{th}. & (10) \end{cases}$$

The first constraint captures the real-time requirement, where $u_\tau(W_i, C_j)$ and $u_{os}(W_i, C_j)$ are the utilization of applications and the operating system, respectively. u_{max} is the upper bound on utilization to satisfy schedulability. The second constraint is introduced to ensure the average power consumption at each sampling window, $p(W_i, C_j)$, is lower than the power budget, p_{budget}. The third constraint requires the MTTF resulting from the thermal profile, TP_j, to be larger than a threshold. The last constraint requires the temperature at each sampling window, $T(W_i, C_j)$, to be less than a threshold, T_{th}. For soft real-time systems like infotainment systems, temporarily violating the first three constraints is acceptable, but the last constraint must be satisfied to avoid timing faults resulting from overheating.

R, $u(W_i, C_j)$, $p(W_i, C_j)$, $T(W_i, C_j)$, and TP_j are functions of the core frequencies and activities. The solution to this problem, called schedule, indicates the active core $c(W_i, C_j)$ and the frequency $f(W_i, C_j)$ for each HP–LP core pair in each sampling window. We design an on-line framework to find the solution.

IV. Reliability Improvement Framework

In this section, we propose a dynamic reliability improvement framework (DRIF) to improve the SER under the real-time, thermal, power, and LTR constraints.

A. Overview

As stated earlier in the paper, to better respond to workload and environment changes that are unavoidable in real-time embedded systems, we aim to develop an on-line approach to solve the problem defined in Eq. (6)-(10). The basic idea of our framework, DRIF, is to incrementally solve the optimization problem by using the history of task execution times and OS utilizations in the previous hyper-period. Note 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 beginning of each hyper-period, a schedule executor (SE), which is triggered at the beginning of each sampling window, and a state collector (SC), which collects the system states in each sampling window. (See Fig. 2.) The system states include the temperature, utilization, frequency, and active core of each HP–LP core pair.

DRIF works as follows. In each sampling window, the SC collects and saves the system states. At the end of each hyper-period, the system state of this hyper-period is sent to the SG. The SG then generates a new schedule (referred to as *starting schedule*), which specifies the active core(s) and frequency of each active core for each sampling window in the next hyper-period, based on the available state information. The starting schedule is the solution of the problem in Eq. (6)-(10). In each sampling window, the SE either adopts the starting schedule or generates a modified version of the starting schedule to adapt to run-time variations.

Though the general idea of DRIF is relatively intuitive, there are a couple of challenges that we need to overcome: (i) since the history (i.e., tasks' execution times) does not always reflect the future, it is possible for the constraints to be violated, and (ii) a highly efficiently algorithm is needed to avoid excessive overhead. We elaborate on our methods of solving these problems in the rest of this section.

B. Schedule generator

We present the details of the schedule generator, SG, in this subsection. The goal of the SG is to generate a schedule for the next hyper-period based on the state information in the current hyper-period. Though it is possible to use an optimization solver to generate an optimal schedule based on Eq. (6)–(10), it would be quite time consuming. Instead, we design a simple heuristic which effectively uses the migration policy presented in Sec. II as well as the efficient LTR estimation method introduced in Sec. III-A.

The main procedure of SG is given in Alg. 1. The inputs are the system state $State_k$ and system schedule $Sched_k$ in the k^{th} hyper-period. Both the $State_k$ and the $Sched_k$ are a set of the state and schedule of each HP–LP core pair. Let $State_{k,j}$ denote the state of the j^{th} pair, which includes the utilization ($U_{k,j}$), core frequency ($F_{k,j}$), power ($P_{k,j}$), active core ($C_{k,j}$), and thermal profile ($TP_{k,j}$). $Sched_{k,j}$ denotes the active core and core frequency in each sampling window of the j^{th} pair.

The idea behind the SG is that for each HP–LP core pair (represented by the j^{th} HP-LP core pair), if the state $State_{k,j}$ can satisfy all of the constraints, try to increase core frequency in $Sched_{k,j}$ to generate a new schedule for the $(k+1)^{th}$ hyper-period (Lines 4–9), otherwise, adjust $Sched_{k,j}$ to satisfy the constraints (Lines 11–13). Given a $State_{k,j}$, the function *check* indicates which constraints are not satisfied, e.g., {(7)} represents the first constraint (in Eq. (7)) being violated. The overhead of checking the LTR constraint by using the proposed method in Sec. III-A is low. Since each HP–LP core pair works independently and task migration among pairs is not allowed, the system schedule $Sched_k$ for all pairs is a combination of the schedules of each pair.

If the j^{th} pair's schedule, $Sched_{k,j}$, can satisfy constraints, we try to increase the core frequency iteratively (Lines 4–9). In one iteration, the minimum frequency is increased to the next level (Line 5). Then, the active core is determined based on this new frequency, utilization, timing overhead of migration and Table I (Line 6). After that, the $State_{k,j}$ is updated based

Algorithm 1 Schedule Generation

```
1: procedure SG(State_k, Sched_k)
2:     for each HP–LP core (represented by the j^th) pair do
3:         if check(State_{k,j}) = Empty then
4:             while check(State_{k,j}) = Empty do
5:                 f(W_h, C_j) = min_{∀W_i}{f(W_i, C_j)}
6:                 increase f(W_h, C_j) to the next level
7:                 c(W_h, C_j) ← core(f(W_h, C_j), u(W_h, C_j))
8:                 State_{k,j} ← update(f(W_h, C_j), c(W_h, C_j))
9:             end while
10:            reduce f(W_h, C_j) to the previous level
11:        else
12:            while check(State_{k,j}) ≠ Empty do
13:                find_action()
14:            end while
15:        end if
16:        for each (represented by the i^th) sampling window do
17:            c(W_i, C_j) ← core(f(W_i, C_j), u(W_i, C_j))
18:            Sched_{k+1,j} ← f(W_i, C_j) ∪ u(W_i, C_j) ∪ Sched_{k,j}
19:        end for
20:    end for
21:    Sched_{k+1} = {Sched_{k+1,j}}, j = 1, ..., n
22: end procedure
```

on the new active core and core frequency (Line 7). The loop exits if the increased $f(W_h, C_j)$ leads $State_{k,j}$ to violate one of the constraints. Hence, we reduce $f(W_h, C_j)$ to the previous level to satisfy the constraints (Line 9).

If $Sched_{k,j}$ cannot satisfy the constraints, we adjust the active core and its frequency to generate a new schedule satisfying the constraints (Lines 11–13). Depending on which constraints are not satisfied (the output of *check*), the actions to adjust the core frequency are summarized in Table III. For the first, second, and fourth constraints (in the first and second row in Table III), we can identify the sampling windows where the constraint is violated. For this situation, the action is to adjust the core frequency in that sampling window. If the third constraint (in the last row in Table III) is not satisfied, we find the sampling window that has the highest power, then reduce the core frequency in that sampling window. In practice, multiple constraints may be violated in different sampling windows. In this case, the action to adjust the core frequency is the combination of the actions in Table III. For example, if both the second and fourth constraints are violated, we reduce the core frequencies in the sampling windows where the temperatures and power exceed their budgets. The function *find_action* (Line 12) determines the corresponding action according to Table III and updates $State_{k,j}$ as in Line 7.

After adjusting the core frequency, the active core based on the new frequency is determined (Line 16). Both the active core and its frequency are added to $Sched_{k+1,j}$ (Line 17). Finally, $Sched_k$ is constructed and returned (Line 20).

C. Schedule executor

The schedule executor, SE, determines the active cores and their frequencies at the beginning of each sampling window. A straightforward approach is to simply follow the schedule generated by the SG. However, since the schedule, $Sched_{(k+1)}$, is generated based on $State_k$, and the utilization in the $(k+1)^{th}$ hyper-period may be different from that in the k^{th} hyper-period, $Sched_{(k+1)}$ may actually violate some or all of the constraints during run time. For soft real-time systems, it is acceptable to temporarily violate constraints Eq. (7)–(9) as they can be compensated for in the next hyper-period. However, violating the temperature constraint in Eq. (10) may either cause timing faults or unexpected throttling. Therefore, the SE should be designed to properly handle such a case.

For each HP–LP core pair, at the beginning of each sampling window, the SE receives the initial temperature from the SC, which is the operating temperature of the previous sampling window, and gets the cores frequencies from $Sched_{(k+1)}$. The SE checks whether the operating temperature in this sampling window can bear the thermal threshold assuming the utilization is 100%. If not, it reduces the core frequency to the previous level. Since the SE needs to be activated in each sampling window, reducing the overhead of the SE is critical. We observe that the thermal threshold and the length of sampling window are constant. Thus, we can statically establish a safe initial temperature for every core frequency. Then, at run time, the SE only needs to check if the initial temperature is larger than the safe one. If yes, the temperature of this sampling window may become larger than the thermal threshold. This method significantly reduces the overhead of executing the SE.

V. Experimental results

To evaluate the proposed DRIF, we performed experiments to compare it with two existing approaches. The experiments are conducted both on a Nvidia's TK1 board [18] and in a simulator. TK1 board is built around Tegra K1 SoC which implements the vSMP architecture and includes 4 HP cores and 1 LP core. In our experiments, the workload is required to be light enough to fit on one core. In order to evaluate DRIF on platforms with multiple HP–LP core pairs, we constructed a simulator that uses the power and thermal parameters extracted from the TK1 board. In the simulator, the running temperature is obtained using Hotspot, a widely adopted thermal modeling tool [21]. The task set is composed by 10 different tasks from the MiBench benchmark suite [19], which represent automotive, network, consumer, and security applications. Tasks are statically mapped to core pairs. Details on parameter extraction and simulator construction are omitted due to page limit.

We compared the performance of DRIF to two representative frameworks, reliability-aware power management (RA−PM) [6] and multi-objective optimization of system reliability (MOO) [16]. RA−PM represents a series of well-cited work [6]–[8] to minimize the energy consumption under the SER constraint. MOO finds the Pareto-optimization of SER

TABLE III. Actions to Satisfy Constraints

Unsatisfied Constraint	Actions
{(7)}	increase $f(W_i, C_j)$ to the next level if $u(W_i, C_j) > u_{max}$
{(8)}, {(10)}	reduce $f(W_i, C_j)$ to the previous level if $p(W_i, C_j) > p_{budget}$ or if $T(W_i, C_j) > T_{max}$
{(9)}	reduce $f(W_i, C_j)$, $p(W_i, C_j)$ is $\max\{P_{k,j}\}$

and LTR by using a genetic algorithm (GA) [16]. We use two metrics for the comparison: (i) the average probability of soft-error fault (PoF) over those hyper-periods where the hard temperature constraint is satisfied, referred to as H-PoF and (ii) the average PoF over those hyper-periods where all the constraints are satisfied, referred to as A-PoF.

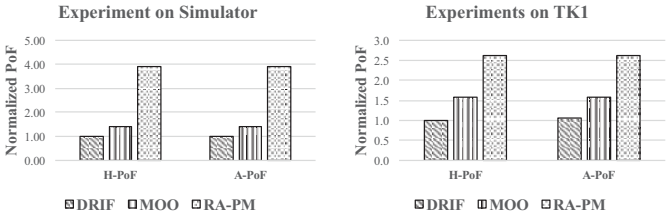

Fig. 3. PoFs for the benchmarks running on the simulator and TK1.

We summarize the experimental results in Fig. 3. As can be readily seen, DRIF achieves a lower PoF than RA−PM and MOO. DRIF adapts to variations in task utilization and increase the core frequency to increase SER when the workload is light. When compared to MOO, DRIF reduces the H-PoF by about 33% and A-PoF by about 31%. RA−PM keeps the SER larger than an acceptable threshold, and its H-PoF and A-PoF are much larger than the DRIF and MOO. On average, both the H-PoF and A-PoF of DRIF is only about 29% for RA−PM.

In order to evaluate the feasibility of DRIF, we counted how many hyper-periods satisfy the constraints. On average, DRIF guarantees 100% of hyper-periods satisfy the hard constraint and 82% of hyper-periods satisfy all constrains, which is only 2% smaller than the MOO and the RA−PM. This small loss does not limit its applications in many soft real-time systems. We also evaluated the overhead of DRIF by measuring its execution time on TK1. The execution time is less than 1 ms, which is much smaller than the length of hyper-period and acceptable for many applications.

VI. Conclusions

We proposed a dynamic reliability improvement framework to maximize soft-error reliability under lifetime reliability, power, and real-time constraints. By exploiting the power features of the HP and LP cores in MPSoC, the framework dynamically activates the most power efficient core of each HP–LP core pair and adjust the core frequencies to satisfy these constraints. The results show that our approach is effective in increasing soft-error reliability compared to existing multi-objective optimization and reliability-aware power management approaches and does not degrade real-time performance.

Acknowledgement

This work was supported in part by NSF under awards CNS-1319904, CNS-1319718 and CNS-131978.

References

[1] W. Wolf, A. A. Jerraya, and G. Martin, “Multiprocessor system-on-chip (MPSoC) technology,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 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-1tm) a multi-core CPU architecture for low power and high performance.” [Online]. Available: https://www.nvidia.com/content/PDF/tegra_white_papers

[4] B. Zhao, H. Aydin, and D. Zhu, “On maximizing reliability of real-time embedded applications under hard energy constraint,” *IEEE Trans. Industrial Informatics*, vol. 6, no. 3, pp. 316–328, May 2010.

[5] G. Macario, M. Torchiano, and M. Violante, “An in-vehicle infotainment software architecture based on Google Android,” in *Proc. Int. Symp. Industrial Embedded Systems*, Jul. 2009, pp. 257–260.

[6] B. Zhao, H. Aydin, and D. Zhu, “Enhanced reliability-aware power management through shared recovery technology,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2009, pp. 63–70.

[7] ——, “Energy management under general task-level reliability constraints,” in *Proc. Int. Conf. the Real-Time and Embedded Technology and Application Symp.*, Apr. 2011, pp. 285–294.

[8] ——, “Generalized reliability-oriented energy management for real-time embedded applications,” in *Proc. Design, Automation Conf.*, June. 2011, pp. 381–386.

[9] A. 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. Measurement and Modeling of Computer System*, Jun. 2009, pp. 169–180.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] C. Chou and R. Marculescu, “FARM: fault-aware resource management in NoC-based multiprocessor platform,” in *Proc. Design, Automation and Test in Europe*, Mar. 2011, pp. 1–6.

[15] J. Huang, et al., “Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems,” in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2011, pp. 247–256.

[16] 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.

[17] 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.

[18] Nvidia, “Jetson Tegra K1.” [Online]. Available: https://developer.nvidia.com/embedded/develop/hardware

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

[20] 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.

[21] 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.