FEST: Fault-Tolerant Energy-Aware Scheduling on Two-Core Heterogeneous Platform

Piyoosh Purushothaman Nair*, Rajesh Devaraj† and Arnab Sarkar‡
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati, Assam, India
Email: *piyoosh@iitg.ac.in, †d.rajesh@iitg.ac.in, ‡arnabsarkar@iitg.ac.in

***Abstract*—Energy awareness and fault-tolerance have emerged as important design constraints in the development of modern safety-critical real-time systems. Devising scheduling strategies for such systems implemented on heterogeneous platforms is a challenging as well as a computationally demanding problem. As a consequence, today we face a scarcity of low-overhead scheduling techniques which are applicable to heterogeneous platforms. *This work attempts to develop an efficient energy-aware heuristic scheme called, FEST, for the fault-tolerant scheduling of real-time tasks on a two-core heterogeneous platform, where one core is high-performance and the other core is power-efficient.* In order to provide fault-tolerance against transient processor faults, we consider a *standby-sparing* approach where the power-efficient core is used to execute *primary* task versions while the high-performance core is operated as a spare to re-execute fault affected tasks (i.e., *backups*). Since, the execution of backups scheduled on the spare core will be cancelled subsequent to the fault-free execution of their primaries, we employ the *Dynamic Power Management* (DPM) technique on both cores to minimize energy consumption. Further, FEST utilizes *backup-backup overloading* to minimize energy consumption while guaranteeing tolerance against a given number of transient processor faults. Experimental results reveal that our scheme achieves good energy savings as compared to the state-of-the-art.**

***Index Terms*—Fault-tolerant, Energy-efficient, Heterogeneous, Real-time, Scheduling**

I. Introduction

Heterogeneous Platforms [1]–[4]: Over the years, the industry is witnessing a significant shift in the nature of processing platforms in real-time embedded systems. Recently, ARM Holdings has developed a heterogeneous processing architecture, called *ARM big.LITTLE* which has been deployed in the latest mobile devices such as Samsung Galaxy Note 4, S6, etc. The big.LITTLE platform contains two cores, one of which is high-performance, called the *big* core, while the other is the lower performance but power-efficient *LITTLE* core. Since both the types of cores share the same instruction set architecture, tasks can be migrated on the fly from one to the other. Meanwhile, differences in the internal microarchitecture of big and LITTLE cores allow them to *provide different execution rates as well as power consumption for each task*. Processing platforms with such varying types of computing elements are called *heterogeneous* (or unrelated) platforms.

Energy-Aware Execution [5]–[8]: Modern computing platforms involve complex microarchitectural designs to meet the computation and performance demands of real-time applications. Consequently, these platforms consume high energy during run-time. When the computing capacity of such platforms are not fully-utilized, we may employ efficient reconfiguration strategies to control and lower their energy consumptions based on the computation demands of the applications running on them. At the operating system level, two schemes are typically employed for energy management. One of them is *Dynamic Power Management* (DPM), where particular parts of the system are turned off strategically when the processors are in idle state. The other is *Dynamic Voltage and Frequency Scaling* (DVFS), which controls power dissipation by scaling supply voltage / frequency.

Need for Fault-tolerance [9]–[14]: Apart from guaranteeing the timely execution of tasks in a resource-constrained environment, ensuring proper functioning of the system even in the presence of faults (i.e., *fault tolerance*) has currently become a design constraint of paramount importance. Specifically, the processors on which the tasks are executed, are subject to a variety of faults. Such faults are broadly classified to be either *permanent* or *transient* [11]. Permanent processor faults are irrecoverable and do not go away with time. On the other hand, transient faults are short-lived (momentary) and their effect goes away after some time. According to studies [11], the ratio of transient-to-permanent faults can be 100:1 or even higher. Thus, robustness against transient faults is emerging as a very important criterion in the design of safety critical real-time systems.

Techniques for Fault-tolerance [11]: In safety-critical systems, fault-tolerance is usually incorporated through temporal or hardware redundancy. The typical techniques employed in hardware redundancy based fault tolerance include *Triple Modular Redundancy* (TMR), *Primary/Backup* (PB), and *Standby Spare* (SS) [11]. In TMR, copies of the same task run concurrently on three different processors which help to mask errors through majority voting. In PB, each task has two versions (namely *primary* and *backup*) which are always mapped to distinct processors. The backup is activated if and only when the primary fails. In SS, one or more spare processors are maintained as *standby* and are activated only when a currently active processor fails. Among these approaches, TMR being the most expensive in terms of hardware cost is often used only in very critical fault-

978-1-5386-6575-6 /18/$31.00 © 2018 IEEE

tolerant systems. PB and SS are generally used in systems which demand moderate safety-criticality levels.

Energy-efficient Fault-tolerant Scheduling: Given a two-type heterogeneous platform (such as ARM's big.LITTLE) and a set of real-time tasks, successfully satisfying all timing, energy and fault-tolerance related specifications is ultimately a *scheduling* problem [15]–[17]. In recent years, many researchers have explored combined energy-efficient and fault-tolerant scheduling techniques for real-time systems. Ejlali et al. proposed a standby-sparing based energy-aware fault-tolerant scheme for the scheduling of real-time aperiodic tasks on dual-cores [18]. In their work, primary and backup tasks are executed on the main and spare processor units, respectively. In order to reduce the energy consumption of this dual-processor system, they employed *Dynamic Voltage Scaling* (DVS) for the main unit and DPM for the spare. Later, in [19] Haque et al. extended their standby-sparing technique for periodic real-time applications. They applied *Earliest-Deadline-First* (EDF) and *Earliest-Deadline-Late* (EDL) scheduling policies on the primary and spare processors, respectively. Recently, Guo et al. generalized the work presented in [19] for real-time systems deployed on platforms consisting of more than two processing cores [20]. However, all these works [18]–[20] have been targeted towards multi-core platforms consisting of *homogeneous (i.e., identical)* processing units.

Our Work: In the context of emerging heterogeneous multiprocessor / multi-core systems, there exists a severe dearth of energy-aware and fault-tolerant scheduling schemes. Due to the architectural differences, the same task may exhibit different timing as well as power characteristics on heterogeneous systems. Therefore, devising combined energy-aware and fault-tolerant scheduling strategies for real-time heterogeneous platforms is a challenging and computationally demanding problem. Recently, Roy et al. explored energy-awareness on heterogeneous multicore platforms consisting of both high-performance and power-efficient cores, using standby-sparing [21]. However, their strategy is oblivious of the number of faults to be tolerated. Due to this, the off-line schedule constructed by their scheme consumes more energy than that required to tolerate a specified number of faults. To alleviate this issue, we develop a standby-sparing based energy-aware fault-tolerant scheduling strategy for heterogeneous systems. Our *contributions* can be summarized as:

- Development of a low-overhead heuristic scheme called, FEST, for the fault-tolerant scheduling of real-time tasks on a heterogeneous dual-core system consisting of a power-efficient core and a high-performance core.
- FEST utilizes *DPM* and *backup-backup overloading* [22] techniques to minimize the energy consumption while guaranteeing the tolerance against a given number of transient processor faults.
- Analysis conducted using extensive simulation based experiments show that our proposed scheme achieves good energy savings as compared to the state-of-the-art [21].

II. System Model and Assumptions

A. System Model

We consider a real-time system consisting of a set of n independent periodic tasks ($T = \{T_1, T_2, ..., T_n\}$) to be executed on a heterogeneous dual-core processing platform. This system consists of a power-hungry, high-performance (big) core, and a power-efficient, relatively slow (little) core. In this work, we assume that tasks are executed in a *frame-based* manner [21], [23]. That is, all tasks in the system share same period, which is equal to the common deadline D. The worst-case number of cycles required by a task T_i on a given core is denoted by C_i. However, T_i may take up to $e_i = C_i/f$ units of execution time to complete on that core when executed at the frequency level f. Due to the asymmetric nature of the cores, the same task may require different number of cycles and execution times on each of these cores. Therefore, each task T_i ($\in T$) is characterized by a two tuple $\langle e_i^{HP}, e_i^{LP} \rangle$, where e_i^{HP} and e_i^{LP} represent the worst-case execution times of T_i on high-performance (denoted by HP) and low-power (denoted by LP) cores, respectively. It is assumed that e_i^{LP} and e_i^{HP} correspond to execution time under the maximum processing frequencies on LP and HP cores, (denoted by f_{max}^{LP} and f_{max}^{HP}), respectively.

B. Power Model

Due to the asymmetric nature of the cores, the HP and LP cores have different power consumption characteristics. The dynamic power consumption of a task T_i on any processing core is modelled as $P_i(f) = a_i f^3 + \alpha_i$, where a_i indicates the switching capacitance, f denotes the processing frequency of the task, and α_i is the frequency-independent power consumption [21]. Therefore, the same task may exhibit different power characteristics on heterogeneous systems.

Each processing core executes tasks in its high-power state and dissipates power as specified by the processing frequency and the characteristics of the executing task. In this work, we employ the *Dynamic Power Management (DPM)* technique on both cores to minimize the energy consumption. Therefore, when a core becomes idle (that is, not executing any tasks), DPM switch off the core to the low-power state. When a core transits between the high-power state to the low-power state, a specific amount of energy and time are consumed. Therefore, the minimum processor idle time required to compensate the cost of entering a low-power state is defined as the *break-even time* TI_{be} [24] of a processing core. In this work, we assume that the cost of entering a low-power state is negligible and so, TI_{be} is assumed to be zero. Let P_{idle}^{LP} and P_{idle}^{HP} denote the power consumption of LP and HP cores at their low-power states, respectively. The overall energy consumption within a frame is determined by aggregating the energy consumption of all cores in that frame.

C. Fault Model

In this work, we employ a standby sparing technique in which one processing core is designated as *primary* and the other as the *spare*. Fig. 1 depicts a standby-sparing

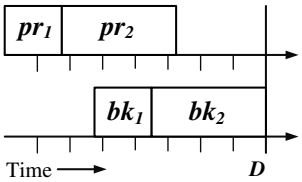

Fig. 1: A standby-sparing system

system. Each task T_i has two versions, namely, *primary copy* (denoted by pr_i) and *backup copy* (denoted by bk_i). bk_i has exactly same timing parameters as that of pr_i. As per the motivation of energy-awareness from literature [21], we assign primary copies of tasks to low-power LP primary core and their backup copies to high-performance HP spare core, respectively. Whenever a primary copy completes, fault-detection mechanisms such as acceptance or sanity tests [25] are conducted to detect a transient fault. If a fault is not detected, (that is, primary completes successfully), the corresponding backup copy on the spare core is deallocated from the schedule dynamically. We assume that each task (primary copy) encounters at most one transient fault and at any point in time, system is able to handle at most k transient faults per frame.

D. Problem Description

Given a set of real-time tasks to be executed on a heterogeneous dual-core system and a number of transient faults to be tolerated, develop an efficient scheduling strategy which (i) meets deadlines of all tasks, (ii) tolerates a specified number of faults, and (iii) minimizes overall energy consumption of the system.

III. Proposed Methodology

In this section, we present our proposed energy-aware fault-tolerant scheduling mechanism, *FEST*, along with an illustrative example to demonstrate its effectiveness.

Before describing the details of *FEST*, we first investigate a design constraint in the standby-sparing *DPM-enabled* system, that is, designating which core to assign as the *primary* and correspondingly, *spare*. There are two configurations:

- Configuration-1: Power-hungry, high-performance HP core as primary and power-efficient, modest performance LP core as spare.
- Configuration-2: High-performance HP core as spare and power-efficient LP core as primary.

In a standby-sparing system, the primary copy of a task is executed in most of the cases, and corresponding backup copy is activated only when its primary fails. In a non-faulty scenario, spare core can always put to its low-power state. It may be inferred that Configuration-2 yields better energy savings as compared to Configuration-1. This is due to the fact that primary copies of tasks are executed on the power-efficient LP core whose power characteristics are much lower than that of the power-hungry HP core.

For example, consider a heterogeneous dual-core system with $f_{max}^{LP} = 0.8$ and $f_{max}^{HP} = 1.0$. Here, f_{max}^{LP} and f_{max}^{HP} denote the normalized maximum frequencies of LP core and HP core, respectively. We assume, $P_{idle}^{LP} = 0.02, P_{idle}^{HP} = 0.05$ and $D = 100ms$. The power consumption parameters for all tasks are given by: $a_i^{HP} = 1.0, \alpha_i^{HP} = 0.1, a_i^{LP} = 0.3$ and $\alpha_i^{LP} = 0.03$. In a non-faulty, fully loaded primary core scenario, Configuration-1 yields an overall energy consumption of $112mJ$, while Configuration-2 consumes only $23.36mJ$ of energy, showing an improvement of 79.14%. Therefore, in our work, we follow the Configuration-2: *high-performance HP core as spare and power-efficient LP core as primary.*

A. Scheduling Strategy

Now, we discuss our proposed energy-aware fault-tolerant strategy *FEST* (refer Algorithm 1), in detail. This strategy can handle at most k (given) transient faults per frame at any point in time and takes the advantage of *backup-backup (BB) overloading* [22], [26]. In BB-overloading, backup copies of multiple tasks are scheduled during the same time interval in order to make efficient utilization of available processing core time. Therefore, in an energy-aware perspective BB-overloading reduces the overall energy consumption of the system.

Algorithm 1 first creates a list Pr_List of tasks in non-increasing order of their execution times (e_i^{LP}) on LP core (step 2). Steps 3-10 determine a schedule for primary copies of tasks in the given task set T. Algorithm 1 extracts each task T_i from Pr_List and schedules its primary copy on LP core by assigning $start\_time$, a time at which T_i will start its execution on the corresponding core. The admission control step of Algorithm 1 (step 5) verifies the schedulability of the given task set T. If T is schedulable, Algorithm 1 creates a list Bk_List of tasks in non-increasing order of their execution times (e_i^{HP}) on HP core (step 12). Then it computes *BB overloading window* ($reserve\_cap$) from the first k tasks in Bk_List based on their execution times (e_i^{HP}) on HP core and reserves $reserve\_cap$ unit of backup slots on HP core as late as possible (steps 13-15). Therefore, instead of assigning time slots for all backup copies, Algorithm 1 reserves only first k backup copies time slots on HP core to minimize overall energy consumption as well as to tolerant k number of faults.

B. Illustrative example

In this section, we demonstrate the working of *FEST* through an example and show its effectiveness over an existing state-of-the-art work.

Consider a system consisting of a set of four real-time tasks T_1, T_2, T_3 and T_4 to be executed on a heterogeneous dual-core platform. This system is characterized by assuming, $f_{max}^{LP} = 0.8, f_{max}^{HP} = 1.0, P_{idle}^{LP} = 0.02, P_{idle}^{HP} = 0.05$ and $D = 100ms$. Table I shows the worst-case execution times (in ms) of these four tasks on HP and LP cores. For all tasks, $a_i^{HP} = 1.0, \alpha_i^{HP} = 0.1, a_i^{LP} = 0.3$ and $\alpha_i^{LP} = 0.03$.

Now, consider a scenario in which these four tasks are executed on a homogeneous dual-core platform with $f_{max} = 1.0$.

Algorithm 1: *FEST*

Input: Heterogeneous dual-core system, T: Set of n real-time tasks, k: Number of transient faults
Output: Fault-Tolerant Task Schedule
// *primary*: power-efficient core (LP); *spare*: high-performance core (HP)
Initialize $start\_time = 0$, $reserve\_cap = 0$;
Create a list Pr_List of tasks in non-increasing order of their execution times (e_i^{LP}) on LP core;
while $Pr\_List \neq \emptyset$ **do**
    Extract the first task T_i from Pr_List;
    if $start\_time + e_i^{LP} \leq D$ **then**
        Schedule primary copy of T_i on LP core at $start\_time$;
        $start\_time = start\_time + e_i^{LP}$;
    else
        Task set T is not schedulable;
        exit;
if T *is schedulable* **then**
    Create a list Bk_List of tasks in non-increasing order of their execution times (e_i^{HP}) on HP core;
    // Compute BB-overloading window
    for *first* k *tasks in* Bk_List **do**
        $reserve\_cap = reserve\_cap + e_i^{HP}$;
    Reserve $reserve\_cap$ unit of backup slots on HP core as late as possible;

TABLE I: A Sample Task Set

	T_1	T_2	T_3	T_4
e_i^{HP}	14	18	10	6
e_i^{LP}	20	24	16	10

In all example scenarios, we focus on the energy consumption of the generated fault-tolerant offline schedule. Fig. 2a depicts tasks schedule on homogeneous system. Here, both primary as well as spare are HP cores. This configuration yields an overall energy consumption of $110.8mJ$. In a heterogeneous dual-core system, we have a power-efficient little (LP) core whose power characteristics are much lower than that of power-hungry big (HP) core. Now, we consider a standby sparing configuration (*SlowerP*) discussed in [21] which assigns primary copies of all tasks to LP core and their backup copies to HP core (Fig. 2b). This yields an overall energy consumption of $68.85mJ$, giving an improvement of 37.85% as compared to homogeneous system. However, this configuration is oblivious to the number of faults to be tolerated and hence, it assigns backup slots for all primaries in the generated schedule. Our proposed strategy, *FEST* reserves only a fixed amount of backup slots based on the number of faults to be tolerated (here, $k = 2$) and allows *backup-backup (BB) overloading* within these backup slots. Therefore, our strategy further reduces the overall energy consumption

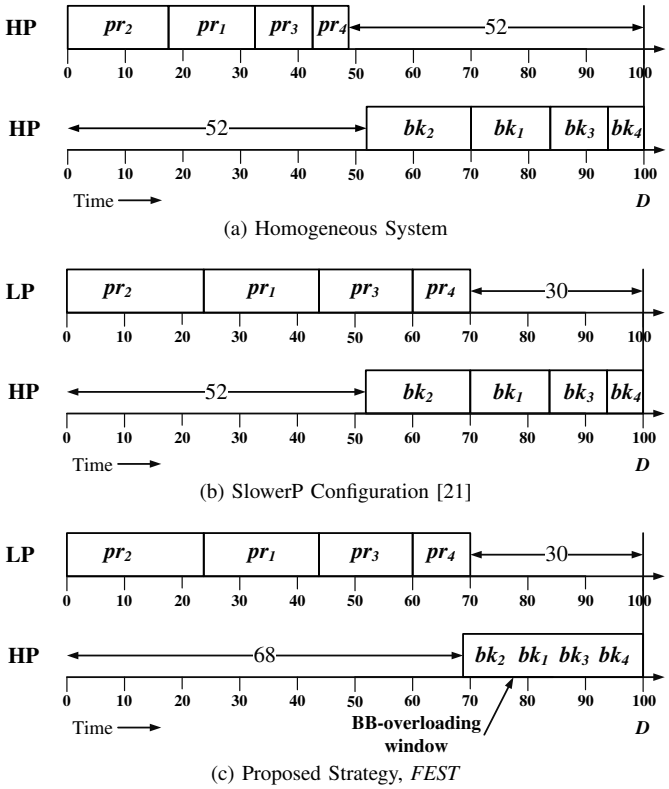

Fig. 2: Different configuration scenarios

to $52.05mJ$ and produces an improvement of 24.4% over $SlowerP$ configuration and 53.02% over homogeneous system. Fig. 2c depicts our proposed fault-tolerant strategy *FEST*.

C. Run-time Behavior

During online execution, when a primary copy completes, fault-detection mechanisms such as acceptance or sanity tests [25] are conducted to detect a transient fault. If a fault is not detected, (that is, primary completes successfully), the corresponding backup copy on the spare core is deallocated from the schedule dynamically. Moreover, the successful completion of a primary copy allows us to dynamically readjust the *BB-overloading window* in order to handle k transient faults on the remaining primary copies. Due to the successful completion of each primary copy, the size of the *BB-overloading window* is readjusted based on the HP core execution times of k remaining tasks and results in shrinking the window size dynamically. If a transient fault is detected, the backup copy of the failed primary executes up to its completion.

IV. Experiments an Results

The performance of our proposed strategy is evaluated through an extensive set of simulation based experiments and compared against an important state-of-the-art scheme *SlowerP* [21].

A. Experimental Setup

We have simulated different heterogeneous dual-core systems which consist of a high-performance HP core with

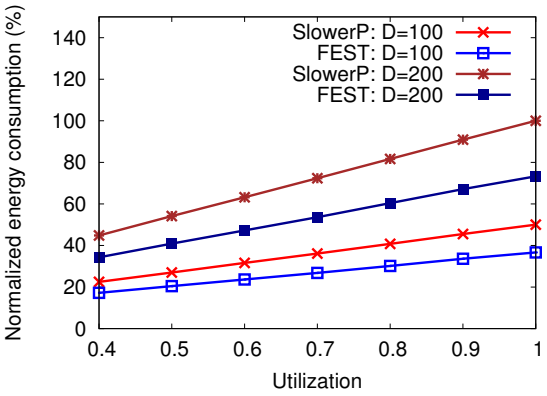

Fig. 3: Impact of utilization

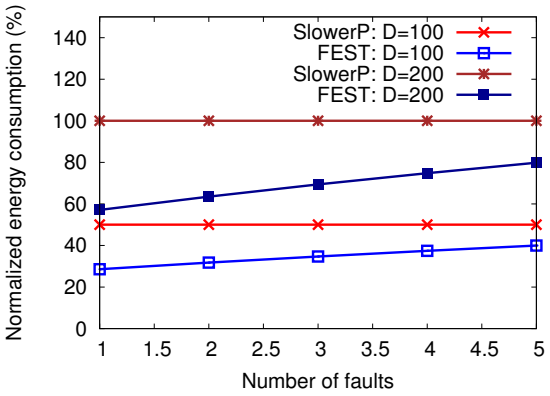

Fig. 4: Impact of number of faults

$f_{max}^{HP} = 1.0$ and a power-efficient LP core with f_{max}^{LP} varying from 0.6 to 0.9. The experimental framework generates different task sets for various experiments. The total utilization U is computed with reference to the power-efficient LP core and normalized with its maximum frequency. Therefore, $U = (\Sigma \frac{C_i^{LP}}{D})/f_{max}^{LP}$. Given the total utilization (U) and the number of tasks to be generated (n), the individual task utilization on the low-power core have been generated from a uniform distribution. The frame deadline D is set to $100ms$ and $200ms$ for all experiments. For all tasks, a_i^{HP} and α_i^{HP} are set to 1.0 and 0.1, respectively. Similarly, $P_{idle}^{LP} = 0.02$ and $P_{idle}^{HP} = 0.05$. Each result data point is the average obtained over 100 different task sets, each containing $n = 10$ tasks. The number of faults (k) to be handled by the system is varied in the range $[1, 5]$.

From literature, it is familiar that same task may require different timing as well as power consumption characteristics on heterogeneous systems [21], [27]. Therefore, we define a time-scaling factor $tscale_i = \frac{C_i^{LP}}{C_i^{HP}}$, and a power-scaling factor $pscale_i = \frac{P_i^{LP}}{P_i^{HP}}$ for each task T_i, as discussed in [21], [28]. The values of $tscale_i$ and $pscale_i$ are randomly generated within the ranges $1.4 \leq tscale_i \leq 2.3$ and $1.4 \leq 1/(tscale_i \times pscale_i) \leq 2.1$, as suggested in [27].

B. Results

We have compared our proposed scheme *FEST* with an existing standby sparing configuration named *SlowerP*, as discussed in [21]. *SlowerP* assigns primary copies of all tasks to LP core and their backup copies to HP core. However, this configuration is oblivious to the number of faults to be tolerated and hence, it assigns backup slots for all primary tasks on LP core to HP core. We employ *DPM* on both LP and HP cores to reduce the energy consumptions.

Impact of utilization: Fig. 3 shows the impact of system load on a heterogeneous system with $f_{max}^{HP} = 1.0$ and $f_{max}^{LP} = 0.8$. The values of n and k are set to 10 and 4, respectively. Here, the utilization shown in the X-axis is considered as regards the power-efficient LP core. With reference to the energy consumption of *SlowerP* at $D = 200$ and $U = 1.0$, we normalize the obtained results. It may be observed from Fig. 3 that as expected, the energy consumption of both *FEST* and *SlowerP* increases when system load increases. This is because for a given number of tasks, the average individual task utilization increases with an increase in the total utilization and hence, task execution time increases. This reduces the idle times of both processing cores, leads to higher energy consumption. It may also be observed that in all system load conditions, our proposed scheme *FEST* outperforms *SlowerP*. This is due to the fact that *FEST* reserves only a fixed amount of backup slots on HP core with respect to the number of faults to be tolerated and allows *backup-backup (BB) overloading* within these backup slots. Further, when D increases from $100ms$ to $200ms$, the energy consumption of both the schemes become almost double. This is because the execution times of tasks become almost double when D increases to $200ms$.

Impact of number of faults: The impact of number of faults (k) on a moderately loaded heterogeneous system is shown in Fig. 4. Here, $f_{max}^{HP} = 1.0, f_{max}^{LP} = 0.8$ and $U = 0.6$ (60% load on power-efficient LP core). The value of n is set to 10. With reference to the energy consumption of *SlowerP* at $D = 200$ and $k = 5$, we normalize the obtained results. It may be observed from Fig. 4 that the energy consumption of *FEST* increases when the number of faults in the system increases whereas *SlowerP* exhibits a constant energy consumption. This is because the amount of backup slots reserved on the HP core by *FEST* increases with an increase in k. This reduces the idle times on HP processing core and results in higher energy consumption. On the other hand, *SlowerP* is oblivious to the number of faults to be tolerated and hence, it assigns backup slots for all primary tasks on LP core to HP core, results in a constant energy consumption. It may also be observed that even though the energy consumption of *FEST* increases with k, it always performs better than *SlowerP*. This is due to the fact that *FEST* reserves only a fixed amount of backup slots on HP core with respect to the number of faults to be tolerated and allows *BB overloading* within these backup slots.

V. Conclusion

This paper presents a standby-sparing based fault-tolerant energy-aware scheduling strategy, named *FEST*, for heterogeneous systems. For a DPM enabled system, we found that designating power-efficient (modest performance) core as primary and power-hungry (high-performance) core as spare yields better energy savings as compared to its counterpart. In order to minimize overall energy consumption and to tolerate a given number of faults, *FEST* reserves only a fixed number of backup slots on the high-performance core and takes the advantage of backup-backup overloading. Our experimental results show that *FEST* performs significantly over the state-of-the-art work.

Acknowledgment

Mr. Piyoosh Purushothaman Nair and Mr. Rajesh Devaraj are partially supported by TATA Consultancy Services (TCS), India, through TCS Research Fellowship Program.

References

[1] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015.

[2] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[3] S. Moulik, R. Devaraj, A. Sarkar, and A. Shaw, "A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks," in *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE, 2017, pp. 204–210.

[4] S. Moulik, R. Devaraj, and A. Sarkar, "Hetero-sched: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *20th IEEE International Conference on High Performance Computing and Communications (HPCC), 2018, in press*.

[5] A. Sarkar, S. Swaroop, S. Ghose, and P. P. Chakrabarti, "Erfair scheduler with processor shutdown," in *High Performance Computing (HiPC), 2009 International Conference on*. IEEE, 2009, pp. 4–12.

[6] M. A. Awan, P. M. Yomsi, G. Nelissen, and S. M. Petters, "Energy-aware task mapping onto heterogeneous platforms using dvfs and sleep states," *Real-Time Systems*, vol. 52, no. 4, pp. 450–485, Jul 2016.

[7] P. P. Nair, A. Sarkar, N. Harsha, M. Gandhi, P. Chakrabarti, and S. Ghose, "Erfair scheduler with processor suspension for real-time multiprocessor embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, p. 19, 2016.

[8] S. Moulik, A. Sarkar, and H. K. Kapoor, "Energy aware frame based fair scheduling," *Sustainable Computing: Informatics and Systems*, vol. 18, pp. 66 – 77, 2018.

[9] P. P. Nair, R. Devaraj, A. Sen, A. Sarkar, and S. Biswas, "Des based modeling and fault diagnosis in safety-critical semi-partitioned real-time systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5029–5034, 2017.

[10] R. Devaraj, A. Sarkar, and S. Biswas, "Fault-tolerant scheduling of non-preemptive periodic tasks using sct of timed des on uniprocessor systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9315–9320, 2017.

[11] C. Krishna, "Fault-tolerant scheduling in homogeneous real-time systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 48, 2014.

[12] R. Devaraj, A. Sarkar, and S. Biswas, "Fault-tolerant preemptive aperiodic rt scheduling by supervisory control of tdes on multiprocessors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, p. 87, 2017.

[13] P. P. Nair, S. Biswas, and A. Sarkar, "Design of light weight exact discrete event system diagnosers using measurement limitation: case study of electronic fuel injection system," *International Journal of Systems Science*, vol. 49, no. 8, pp. 1760–1783, 2018.

[14] R. Devaraj, A. Sarkar, and S. Biswas, "A design fix to supervisory control for fault-tolerant scheduling of real-time multiprocessor systems with aperiodic tasks," *International Journal of Control*, vol. 88, no. 11, pp. 2211–2216, 2015.

[15] K. Chauhan, P. Piyoosh, A. Sarkar, and S. Biswas, "A priori overload handling in erfair scheduled embedded systems: Hybrid automata approach," in *2014 Annual IEEE India Conference (INDICON)*. IEEE, 2014, pp. 1–6.

[16] R. Devaraj, A. Sarkar, and S. Biswas, "Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed des," in *American Control Conference (ACC), 2017*. IEEE, 2017, pp. 3212–3217.

[17] R. Devaraj, A. Sarkar, and B. Santosh, "Supervisory control approach and its symbolic computation for power-aware rt scheduling," *IEEE Transactions on Industrial Informatics*, 2018.

[18] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. ACM, 2009, pp. 193–202.

[19] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware standby-sparing technique for periodic real-time applications," in *Computer Design (ICCD), 2011 IEEE 29th International Conference on*. IEEE, 2011, pp. 190–197.

[20] Y. Guo, D. Zhu, and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*. IEEE, 2013, pp. 62–71.

[21] A. Roy, H. Aydin, and D. Zhu, "Energy-aware standby-sparing on heterogeneous multicore systems," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[22] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerant scheduling on a hard real-time multiprocessor system," in *Parallel Processing Symposium, 1994. Proceedings., Eighth International*. IEEE, 1994, pp. 775–782.

[23] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, p. 23, 2013.

[24] I. Lee, J. Leung, and S. H. Son, Eds., *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC Press, 2007.

[25] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Elsevier, 2010.

[26] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Transactions on Parallel and distributed systems*, vol. 8, no. 3, pp. 272–284, 1997.

[27] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*. IEEE, 2013, pp. 1–10.

[28] A. Roy, H. Aydin, and D. Zhu, "Energy-efficient primary/backup scheduling techniques for heterogeneous multicore systems," in *Green and Sustainable Computing Conference (IGSC), 2017 Eighth International*. IEEE, 2017, pp. 1–8.