Resource Management for Improving Overall Reliability of Multi-Processor Systems-on-Chip

Yue Ma, Junlong Zhou, Thidapat Chantem, Robert P. Dick, and X. Sharon Hu

1 Introduction

This section presents the concepts and models associated with soft-error reliability and lifetime reliability, and reviews the related work on these topics.

1.1 Background

Modern multi-processor systems on a chip (MPSoCs) may contain both multicore processors and integrated GPUs, which are especially suitable for real-time embedded applications requiring massively parallel processing capabilities. Since MPSoCs

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

J. Zhou
School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
e-mail: jlzhou@njust.edu.cn

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

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

J. Henkel, N. Dutt (eds.), *Dependable Embedded Systems*, Embedded Systems,
https://doi.org/10.1007/978-3-030-52017-5_10

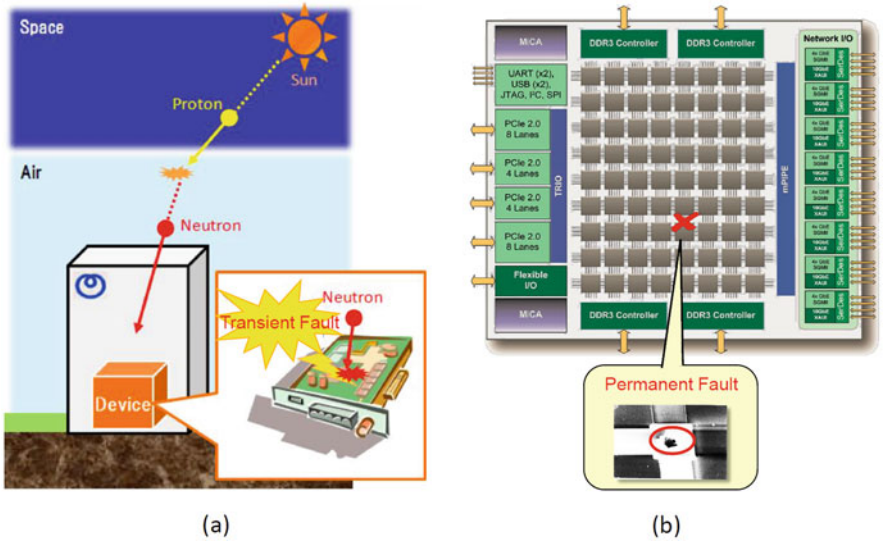

Fig. 1 Illustration of transient and permanent faults

offer good performance and power consumption, they have been widely used in many real-time applications such as consumer electronics, automotive electronics, industrial automation, and avionics [1]. For these applications, the MPSoC needs to satisfy deadline, quality-of-service (e.g., resolution of video playback), and reliability requirements. The reliability requirements include both soft-error reliability (SER), influenced by transient faults, and lifetime reliability (LTR), influenced by permanent faults. This chapter presents approaches to improving SER and/or LTR while satisfying deadline and quality-of-service requirements for real-time embedded systems.

Transient faults are mainly caused by high-energy particle strikes, e.g., resulting from spallation from cosmic rays striking atoms in the upper atmosphere [2] (see Fig. 1a). Transient faults may lead to errors that appear for a short time and then disappear without damaging the device or shortening its lifetime; these are called soft errors. They may prevent tasks from completing successfully. SER is used to quantify the probability that tasks will complete successfully without errors due to transient faults. SER can be increased by using reliability-aware techniques such as replication, rollback recovery, and frequency elevation, which either tolerate transient faults or decrease their rates.

Permanent faults are caused by wear in integrated circuits. An example is illustrated in Fig. 1b. Permanent faults can lead to errors that persist until the faulty hardware is repaired or replaced. Multiple wear-out effects such as electromigration (EM), stress migration (SM), time-dependent dielectric breakdown (TDDB), and thermal cycling (TC) can lead to permanent faults. The rates of these effect depend exponentially on temperature. In addition, thermal cycling depends on the

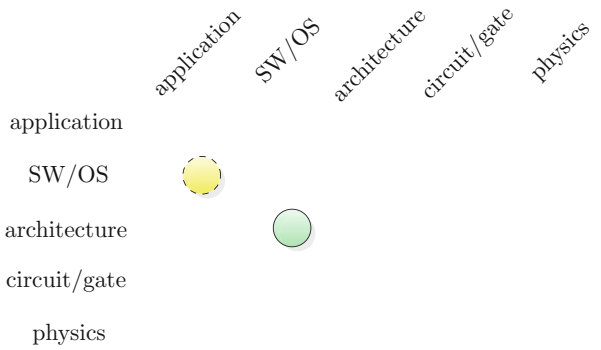

Fig. 2 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

temperature range, maximum temperature, and cycle frequency. To improve LTR, temperature peaks and variation must be limited.

To reduce the cost of repairing/replacing an MPSoC system and maintain some desired level of quality-of-service, improving SER due to transient faults and LTR due to permanent faults become an imperative design concern. In this chapter we present two techniques that optimize SER and LTR separately and show how to make appropriate trade-offs between them for improving overall system reliability. Figure 2 illustrates the abstraction layers representing the main contribution of this chapter.

1.2 Related Work

Considerable research has been done on improving SER. Haque et al. [3] present an energy-efficient task replication method to achieve a high SER target for periodic real-time applications running on a multicore system with minimum energy consumption. Salehi et al. [4] propose a low-overhead checkpointing-based rollback recovery scheme to increase system SER and reduce the number of checkpoints for fault-tolerant real-time systems. Zhou et al. [5] improve system SER by judiciously determining proper replication and speedup of tasks. Zhou and Wei [6] describe a stochastic fault-tolerant task scheduling algorithm that specifically considers uncertainty in task execution caused by transient fault occurrences to increase SER under task deadline constraints. These work increase SER but do not consider permanent faults.

Many studies have focused on increasing LTR. Huang et al. [7] describe an analytical model to derive the LTR of multicore systems and a simulated annealing algorithm to reduce core temperature and temperature variation to improve system LTR. Chantem et al. [8] present a dynamic task assignment and scheduling scheme to maximize system LTR by mitigating core wear due to thermal cycling. Ma et

al. [1] optimize system LTR by establishing an online framework that dynamically controls cores' utilization. Das et al. [9, 10] improve the LTR of network-on-chips (NoCs) and also solve the energy–reliability trade-off problem for multimedia MPSoCs. However, these approaches neglect transient faults.

There is research on handling SER and LTR together. Zhou et al. [5] propose a task frequency and replication selection strategy that balances SER and LTR to maximize system availability. Ma et al. [11] establish an online framework for increasing SER and LTR of real-time systems running on "big–little" type MPSoCs. A genetic algorithm based approach [12] that determines task mappings and frequencies is developed to jointly improve SER and LTR. Aliee et al. [13] adopt mean time to failure (MTTF) as the common metric to evaluate SER and LTR and design a success tree based scheme for reliability analysis for embedded systems. Unlike work [5, 11–13] that ignore the variations in performance, power consumption, and reliability parameters, Gupta et al. [14] explore the possibility of constructing reliable systems to compensate for the variability effects in hardware through software controls. These efforts consider CPU reliability but ignore the reliability effects of GPUs.

1.3 Soft-Error Reliability Model

SER is the probability that no soft errors occur in a particular time interval [5], i.e.,

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

where f is the core frequency, $|\Delta t|$ is the length of the time interval, U is the core's utilization within $|\Delta t|$, and $\lambda(f)$ is the average fault rate depending on f [5]. Specifically, we have

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

where λ_0 is the average fault rate at the maximum core frequency. $f_{\min}$ and $f_{\max}$ are the minimum and maximum core frequency, and d ($d > 0$) is a hardware-specific constant indicating the sensitivity of fault rate to frequency scaling. Reducing frequency leads to an exponential increase in fault rate because frequency is a roughly linear function of supply voltage. As frequency reduces, supply voltage decreases, decreasing the critical charge (i.e., the minimum amount of charge that must be collected by a circuit to change its state) and exponentially increasing fault rate [15].

Since CPU and GPU fabrication processes are similar, the device-level SER model above applies to both. Let r_G and r_i ($i = 1, 2, \ldots, m$) represent the SER of the GPU and the ith CPU core, respectively. As the correct operation of an MPSoC system-level depends on the successful execution of GPU and CPU cores, the system-level SER is calculated as the product of reliabilities of all individual cores, i.e.,

$$R = r_G \times \prod_{i=1}^{m} r_i. \tag{3}$$

1.4 Lifetime Reliability Model

MTTF is commonly used to quantify LTR. We focus on four main failure mechanisms: EM, TDDB, SM, and TC. EM refers to the dislocation of metal atoms caused by momentum imparted by electrical current in wires and vias [16]. TDDB refers to the deterioration of the gate oxide layer [17]. SM is caused by the directionally biased motion of atoms in metal wires due to mechanical stress caused by thermal mismatch between metal and dielectric materials [18]. TC is wear due to thermal stress induced by mismatched coefficients of thermal expansion for adjacent material layers [19].

The system-level MTTF modeling tool introduced by Xiang et al. [20] can be used to estimate LTR when considering the above four failure mechanisms. This tool integrates three levels of models, i.e., device-, component-, and system-level models. At the device level, wear due to the above four mechanisms is modeled. The modeling tool accounts for the effect of using multiple devices in a component upon fault distributions, e.g., the effects of EM are most appropriately modeled using a lognormal distribution at the device level, but with a Weibull distribution for components containing many devices. Based on the device-level reliability models and temporal failure distributions, component-level MTTF is calculated [20]. Then, based on component-level reliability, the system-level MTTF is obtained by Monte Carlo simulation.

2 LTR and SER Optimization

This section introduces two approaches for LTR and SER optimization, and discusses the trade-off between them.

2.1 LTR Optimization

EM, SM, and TDDB wear rates depend exponentially on temperature. However, wear due to thermal cycling depends on the amplitude (i.e., the difference between the proximal peak and valley temperature), period, and maximum temperature of thermal cycles. Figure 3 summarizes some system MTTF data obtained from the system-level LTR modeling tool with default settings [20]. Figure 3a–c depicts the MTTF of an example system as a function of the amplitude, period, and peak

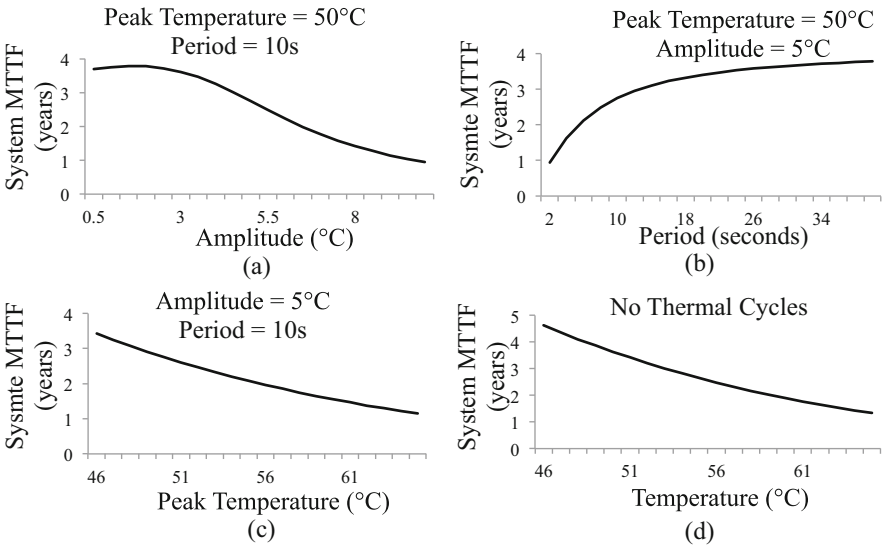

Fig. 3 System MTTF due to: (**a**) amplitude of thermal cycle; (**b**) period of thermal cycles; (**c**) peak temperature of thermal cycles; and (**d**) temperature without thermal cycles

temperature of thermal cycles, respectively. As a comparison, Fig. 3d shows the system MTTF due to temperature alone without thermal cycles. As can be seen from Fig. 3, system MTTF generally increases for lower temperatures and smaller thermal cycles.

A system's LTR is determined by its operating temperature and thermal cycles. Given that lower frequencies and voltages lead to higher utilization but lower temperatures, one method to improve system MTTF is to control core utilization. For example, we have developed a framework called Reliability-Aware Utilization Control (RUC) [21] to mitigate the effects of both operating temperature and thermal cycling. RUC consists of two controllers. The first controller reduces the peak temperature by periodically reducing core frequencies subject to task deadline requirements. Although frequent changes in core frequency helps to reduce peak temperature, they may increase the frequency of thermal cycling and reduce lifetime reliability. Hence, the second controller minimizes thermal cycling wear by dynamically adjusting the period of the first controller to achieve longer thermal cycles as well as lower peak temperature.

2.2 SER Optimization

Recovery allocation strategies and task execution orders can affect system-level soft-error reliability (as shown in Fig. 4). In this example, there are four tasks that

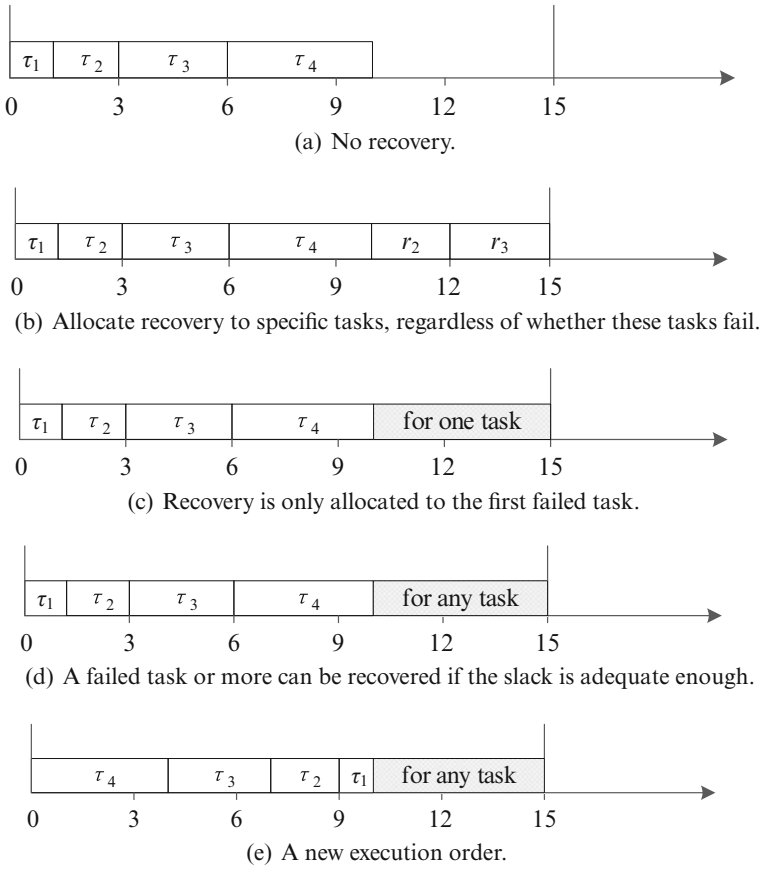

Fig. 4 Motivating examples illustrate different recovery allocation strategies and task execution order affect system-level SER. (**a**) No recovery. (**b**) Statically allocate recovery to specific tasks, regardless of whether these tasks fail. (**c**) Recovery is only allocated to the first failed task. (**d**) A failed task or more can be recovered if the slack is adequate. (**e**) A new task execution order

share a common period of 15 s. We further suppose the worst-case execution times of the tasks are 1, 2, 3, and 4 s. All tasks in the set execute at the highest core frequency. As indicated by the reliability model presented in Sect. 1, the SERs of the tasks are 0.904, 0.819, 0.741, and 0.670.

If no recovery is allowed as shown in Fig. 4a, the system-level SER, i.e., the probability that all tasks can complete successfully, is 0.368. Allowing recovery of some tasks increases SER. One method is to allocate recoveries to tasks offline [5]. Figure 4b represents a better solution for maximizing the system-level SER, in which tasks τ_2 and τ_3 have recoveries r_2 and r_3. In this case, the system-level SER is 0.547. Another approach allocates recovery online [22]. Figure 4c shows a scenario where the first failed task has a recovery [22]. The system-level SER is 0.686, which

is higher than that in Fig. 4b. However, although the slack is dynamically used in Fig. 4c, only one task can be recovered.

In the above online recovery allocation example, a failed task is recovered if the remaining slack is adequate, and tasks consume slack on a first-come, first-served basis (see Fig. 4d). For example, task τ_2 can recover even if τ_1 fails. However, task τ_3 cannot recover if both τ_1 and τ_2 fail because the remaining slack for τ_3 is only 2 s. Task τ_4 can recover only when all tasks succeed or only τ_1 fails. Hence, the probabilities of recovering τ_3 and τ_4 are 0.983 and 0.607, and the system-level SER is 0.716. Now, consider the impact of task scheduling on the system-level SER. Figure 4e represents a new schedule where the task's priority is the inverse of its execution time. In this case, the probabilities of recovering τ_1, τ_2, τ_3, and τ_4 are 0.792, 0.670, 0.670, and 1.000. In contrast with Fig. 4d, the task with the lowest SER, τ_4, can always be recovered, but the system-level SER is 0.692. Hence, a scheduling algorithm that simply improves the probability of recovery for some specific tasks may not be a good solution.

Based on these observations, we design an SER improvement framework [23] that statically schedules tasks and dynamically allocates recoveries. The framework is composed of a simple and fast scheduling algorithm for special task sets and a powerful scheduling algorithm for general task sets. For more details of the two scheduling algorithms, readers can refer to [23].

3 Trade-Off Between LTR and SER

Certain design decisions (e.g., task mapping and voltage scaling) may increase LTR but decrease SER, and vice versa. In other words, improving overall reliability requires trade-offs between LTR and SER. Recently, several efforts have focused on these trade-offs. Below, we describe two case studies in LTR and SER trade-off: (1) "big–little" type MPSoCs and (2) CPU–GPU integrated MPSoCs.

3.1 "Big–Little" MPSoCs

To address power/energy concerns, various heterogeneous MPSoCs have been introduced. A popular MPSoC architecture often used in power/energy-conscious real-time embedded applications is composed of pairs of high-performance (HP) cores and low-power (LP) cores. Such HP and LP cores present unique performance, power/energy, and reliability trade-offs. Following the terminology introduced by ARM, we refer to this as the "big–little" architecture. Nvidia's variable symmetric multiprocessing architecture is such an example [24].

Executing tasks on an LP core improves LTR by reducing temperature and improves SER through a higher core frequency. Although the primary goal of "big–little" MPSoCs is to reduce power consumption by executing a light workload on

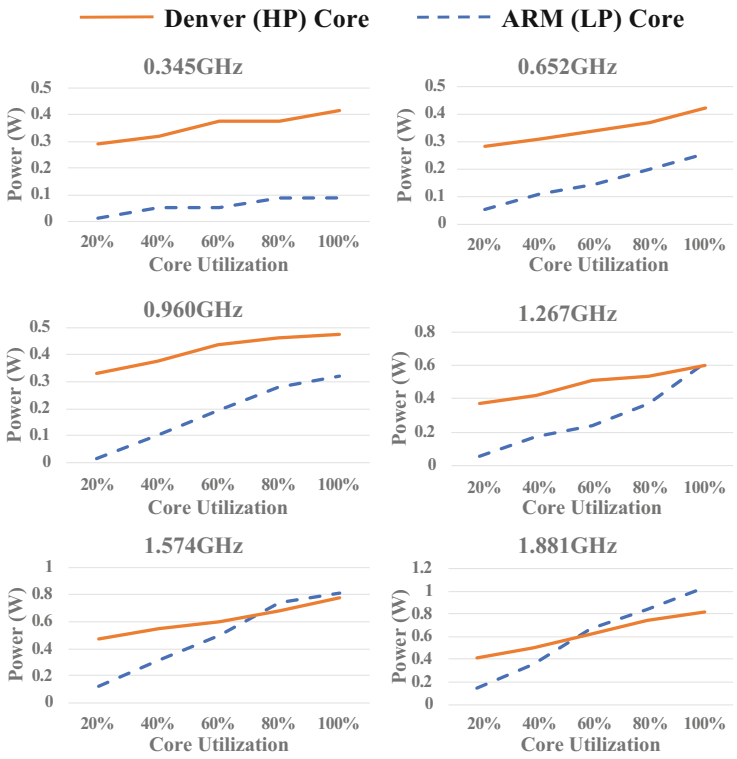

Fig. 5 The measured power consumptions of HP (Denver) core and LP (ARM) core on Nvidia's TX2 chip as functions of utilization and frequency

the LP cores, there are circumstances in which an LP core consumes more power than an HP core. Carefully characterizing the power consumption behavior of HP and LP cores is necessary. For example, the power consumption of the HP core and LP core on Nvidia's TX2[1] is shown in Fig. 5. The LP core consumes less power than the HP core only when the core frequency is low and the workload is light. One possible reason for this phenomenon is that the HP and LP cores have different microarchitectures, as is the case with the TX2. Another possible reason is that the transistors in the HP core and LP core have different threshold voltages. The LP core has low leakage power but requires high voltage to operate at higher frequencies. On the other hand, the HP core can work at high frequency with a low voltage.

The above observations reveal that in order to reduce power consumption of MPSoCs and improve reliability, it is necessary to fully account for the power

[1]Note that TX2 is composed of ARM Cortex A57 cores that support multithreading, and Nvidia' Denver cores for high single-thread performance with dynamic code optimization. Denver cores can be treated as HP cores and ARM cores can be treated as LP cores when running single-threaded applications.

features of heterogeneous cores and carefully map tasks to the most appropriate cores. A good guideline is to run tasks having short execution times on LP cores with low frequencies and tasks having long execution times on HP cores with high frequencies. The execution models of HP and LP cores must also be considered. For example, HP and LP cores on Nvidia's TK1 cannot execute at the same time. However, on Nvidia's TX2, HP and LP cores can work simultaneously. Although an HP core and an LP core can execute at different frequencies, all HP cores must share one frequency, as must LP cores. Hence, a strategy to improve reliability should migrate tasks dynamically and consider both the power features and execution models of HP and LP cores. Using this guideline, we have developed frameworks for different hardware platforms to improve soft-error reliability under lifetime reliability, power consumption, and deadline constraints [1, 11].

3.2 CPU–GPU Integrated MPSoCs

Thanks to the massively parallel computing capability offered by GPUs and the general computing capability of CPUs, MPSoCs with integrated GPUs and CPUs have been widely used in many soft real-time embedded applications, including mobile devices [25] and intelligent video analytics. For many such applications, SER due to transient faults and LTR due to permanent faults are major design concerns. A common reliability improvement objective is to maximize SER under an LTR constraint.

An application task set is used to illustrate how a task's execution time depends on whether it executes on the same core as the operating system. The varying execution times of tasks change the overall workload and operating temperature, influencing LTR and SER. Experiments were performed on Nvidia's TK1 chip (with CUDA 6.5) with default settings to measure task execution times. Six tasks from different benchmark suites were executed (see Table 1). Each task's increase in CPU time resulting from executing on a different core than the operating system is shown in Fig. 6 and the averages of additional GPU times are shown in Table 2. For all tasks, the additional CPU times can be significant and are input dependent. In contrast to the additional CPU time, the additional GPU time is negligible: the additional GPU times of all measured application tasks are less than 1% of the tasks' execution times. This increase can be ignored in most soft real-time applications. Similar phenomena can be observed for other platforms. On Nvidia's TX2 chip, the additional CPU times of application tasks are illustrated in Fig. 7.

The above observations imply that a task's CPU time increases if executed on a different core than the operating system, but its GPU time does not change. Since both LTR and SER increase with a lighter workloads, this observation reveals that we should consider what resources tasks use when assigning them to cores. Generally, the primary core, on which the operating system runs, should be reserved for application tasks that require GPU resources to complete.

Table 1 Application tasks used to measure additional execution times

Name	Description	Source
VectorAdd	Vector addition	CUDA samples [26]
SimpleTexture	Texture use	
MatrixMul	Matrix multiplication	
Gaussian	Gaussian elimination	Rodinia [27]
BFS	Breadth-first search	
Backprop	Back propagation	

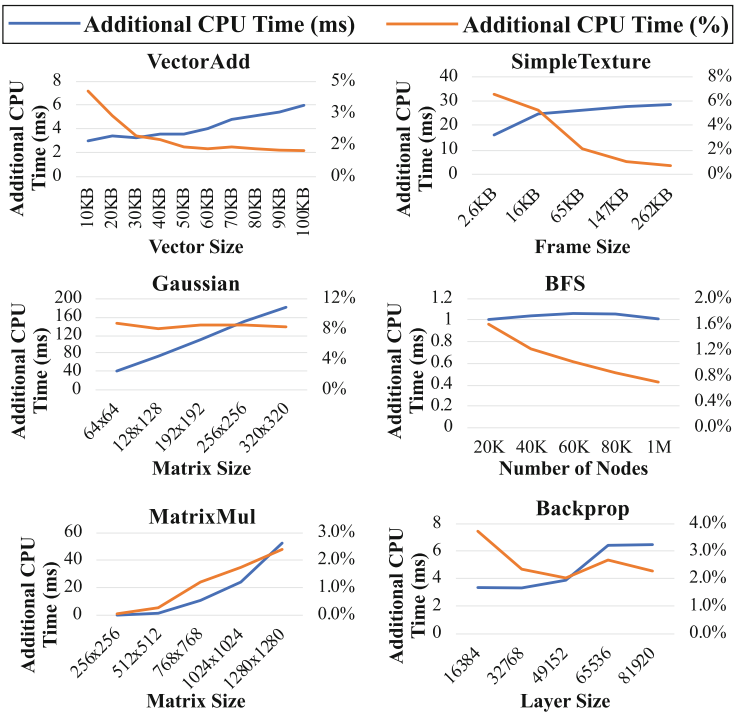

Fig. 6 Measured additional CPU times on TK1 for tasks executing on non-OS CPU cores

Table 2 Observed additional GPU time on TK1

	Additional GPU time	
Tasks	(ms)	(%)
VectorAdd	0.38	0.01
SimpleTexture	0.09	0.00
MatrixMul	0.22	0.11
Gaussian	0.38	0.00
BFS	0.003	0.20
Backprop	0.31	0.68

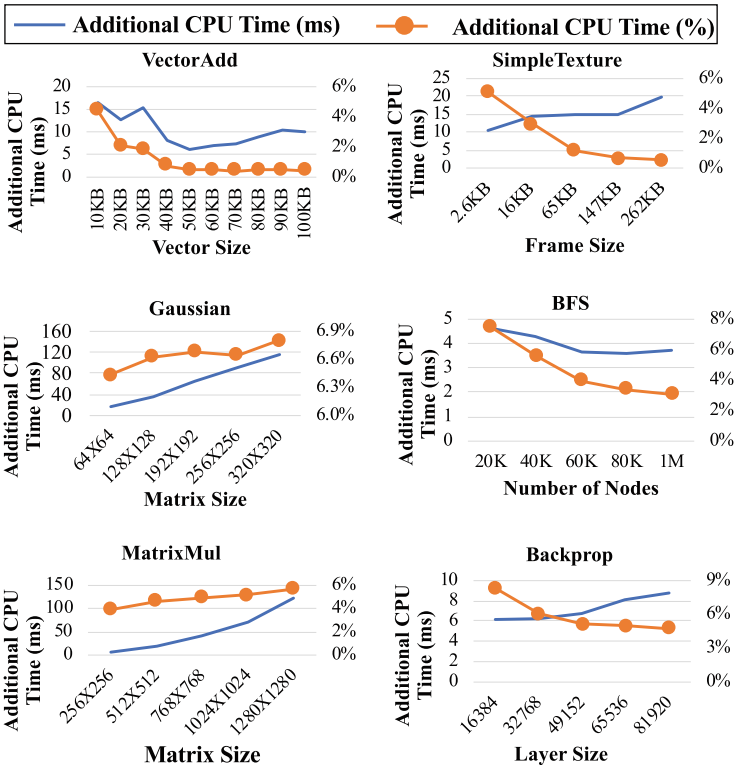

Fig. 7 Measured additional CPU times on TX2 for tasks executing on CPU cores that are different from the core where the operating system runs

4 Conclusion

Real-time embedded system soft-error and lifetime reliabilities are important. Generally, increasing a core's frequency, allocating recoveries and allowing replications improve soft-error reliability, but may increase operating temperature thereby reducing lifetime reliability. MPSoCs used in many applications are heterogeneous and integrate high-performance cores, low-power cores, and even GPUs. System designers should model the task-dependent power consumptions and execution times of the cores available to them, and use these models to solve the SER and LTR trade-off problem.

References

1. Ma, Y., Chantem, T., Dick, R.P., Wang, S., Hu, X.S.: An on-line framework for improving reliability of real-time systems on "big-little" type MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2017)
2. Henkel, J., et al.: Design and architectures for dependable embedded systems. In: Proceedings of the International Conference Hardware/Software Codesign and System Synthesis, October, pp. 69–78 (2011)
3. Haque, M.A., Aydin, H., Zhu, D.: On reliability management of energy-aware real-time systems through task replication. IEEE Trans. Parallel Distrib. Syst. **28**(3), 813–825 (2017)
4. Salehi, M., Tavana, M.K., Rehman, S., Shafique, M., Ejlali, A., Henkel, J.: Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems. IEEE Trans. VLSI Syst. **24**(7), 2426–2437 (2016)
5. Zhou, J., Hu, X.S., Ma, Y., Wei, T.: Balancing lifetime and soft-error reliability to improve system availability. In: Proceedings of Asia and South Pacific Design Automation Conference, January, pp. 685–690 (2016)
6. Zhou, J., Wei, T.: Stochastic thermal-aware real-time task scheduling with considerations of soft errors. J. Syst. Softw. **102**, 123–133 (2015)
7. Huang, L., Yuan, F., Xu, Q.: On task allocation and scheduling for lifetime extension of platform-based MPSoC designs. IEEE Trans. Parallel Distrib. Syst. **22**(12), 789–800 (2011)
8. Chantem, T., Xiang, Y., Hu, X.S., Dick, R.P.: Enhancing multicore reliability through wear compensation in online assignment and scheduling. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1373–1378 (2013)
9. Das, A., Kumar, A., Veeravalli, B.: Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems. In: Proceedings of Design, Automation and Test in Europe, March, pp. 689–694 (2013)
10. Das, A., Kumar, A., Veeravalli, B.: Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2014)
11. Ma, Y., Zhou, J., Chantem, T., Dick, R.P., Wang, S., Hu, X.S.: On-line resource management for improving reliability of real-time systems on "big-little" type MPSoCs. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **39**, 88–100 (2018)
12. Das, A., Kumar, A., Veeravalli, B., Bolchini, C., Miele, A.: Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1–6 (2014)
13. Aliee, H., Glaß, M., Reimann, F., Teich, J.: Automatic success tree-based reliability analysis for the consideration of transient and permanent faults. In: Proceedings of Design, Automation and Test in Europe, March, pp. 1621–1626 (2013)
14. Gupta, P., et al.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **32**(1), 8–23 (2012)
15. Harucha, P., Suensson. C.: Impact of CMOS technology scaling on the atmospheric neutron soft error rate. IEEE Trans. Nucl. Sci. **47**(6), 2586–2594 (2000)
16. Srinivasan, J., Adve, S., Bose, P., Rivers, J.: The impact of technology scaling on lifetime reliability. In: Proceedings of International Conference Dependable Systems and Networks, June, pp. 177–186 (2004)
17. Srinivasan, J., Adve, S., Bose, P., Rivers, J.: Exploiting structural duplication for lifetime reliability enhancement. In: Proceedings of International Symposium on Computer Architecture, June, pp. 520–531 (2005)
18. JEDEC Solid State Technology Association: Failure mechanisms and models for semiconductor devices. Joint Electron Device Engineering Council. Technical Report, August 2003, JEP 122-B
19. Ciappa, M., Carbognani, F., Fichtner, W.: Temperature-aware microarchitecture: modeling and implementation. IEEE Trans. Device Mater. Reliab. **3**(4), 191–196 (2003)

20. Xiang, Y., Chantem, T., Dick, R.P., Hu, X.S., Shang, L.: System-level reliability modeling for MPSoCs. In: Proceedings of International Conference Hardware/Software Codesign and System Synthesis, October, pp. 297–306 (2010)
21. Ma, Y., Chantem, T., Dick, R.P., Hu, X.S.: Improving system-level lifetime reliability of multicore soft real-time systems. IEEE Trans. VLSI Syst. **25**(6), 1895–1905 (2017)
22. Zhao, B., Aydin, H., Zhu, D.: Enhanced reliability-aware power management through shared recovery technology. In: Proceedings of International Conference Computer-Aided Design, November, pp. 63–70 (2009)
23. Ma, Y., Chantem, T., Dick, R.P., Hu, X.S.: Improving reliability for real-time systems through dynamic recovery. Proceedings of Design, Automation and Test in Europe, March, pp. 515–520 (2018)
24. Nvidia: Variable SMP (4-plus-1) a multi-core CPU architecture for low power and high performance. https://www.nvidia.com/content/PDF/tegra white papers (2018). Online. Accessed Oct 2018
25. Samsung: Samsung Exynos. https://www.samsung.com/semiconductor/minisite/exynos/ (2018). Online. Accessed Oct 2018
26. Nvidia: CUDA samples. http://docs.nvidia.com/cuda/cuda-samples/index.html (2018). Online. Accessed Oct 2018
27. University of Virginia: Rodinia: accelerating compute-intensive applications with accelerators. https://rodinia.cs.virginia.edu (2018). Online. Accessed Oct 2018