Journal of
Low Power Electronics and Applications

Article

Reliability-Aware Resource Management in Multi-/Many-Core Systems: A Perspective Paper

Siva Satyendra Sahoo *,†, Behnaz Ranjbar *,† and Akash Kumar *

CFAED, Technische Universität Dresden, 01062 Dresden, Germany
* Correspondence: siva_satyendra.sahoo@tu-dresden.de (S.S.S.); behnaz.ranjbar@tu-dresden.de (B.R.); akash.kumar@tu-dresden.de (A.K.)
† These authors contributed equally to this work.

Abstract: With the advancement of technology scaling, multi/many-core platforms are getting more attention in embedded systems due to the ever-increasing performance requirements and power efficiency. This feature size scaling, along with architectural innovations, has dramatically exacerbated the rate of manufacturing defects and physical fault-rates. As a result, in addition to providing high parallelism, such hardware platforms have introduced increasing unreliability into the system. Such systems need to be well designed to ensure long-term and application-specific reliability, especially in mixed-criticality systems, where incorrect execution of applications may cause catastrophic consequences. However, the optimal allocation of applications/tasks on multi/many-core platforms is an increasingly complex problem. Therefore, reliability-aware resource management is crucial while ensuring the application-specific Quality-of-Service (QoS) requirements and optimizing other system-level performance goals. This article presents a survey of recent works that focus on reliability-aware resource management in multi-/many-core systems. We first present an overview of reliability in electronic systems, associated fault models and the various system models used in related research. Then, we present recent published articles primarily focusing on aspects such as application-specific reliability optimization, mixed-criticality awareness, and hardware resource heterogeneity. To underscore the techniques' differences, we classify them based on the design space exploration. In the end, we briefly discuss the upcoming trends and open challenges within the domain of reliability-aware resource management for future research.

Keywords: multi/many-core platforms; reliability; resource management; mixed-criticality

check for updates

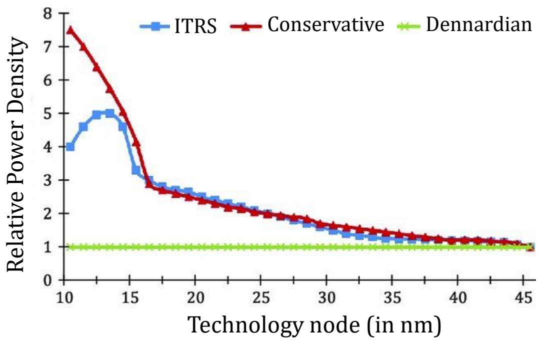
Citation: Sahoo, S.S.; Ranjbar, B.; Kumar, A. Reliability-Aware Resource Management in Multi-/Many-Core Systems: A Perspective Paper. *J. Low Power Electron. Appl.* **2021**, *11*, 7. https://doi.org/10.3390/jlpea11010007

Academic Editor: Amit Kumar Singh
Received: 17 December 2020
Accepted: 19 January 2021
Published: 25 January 2021

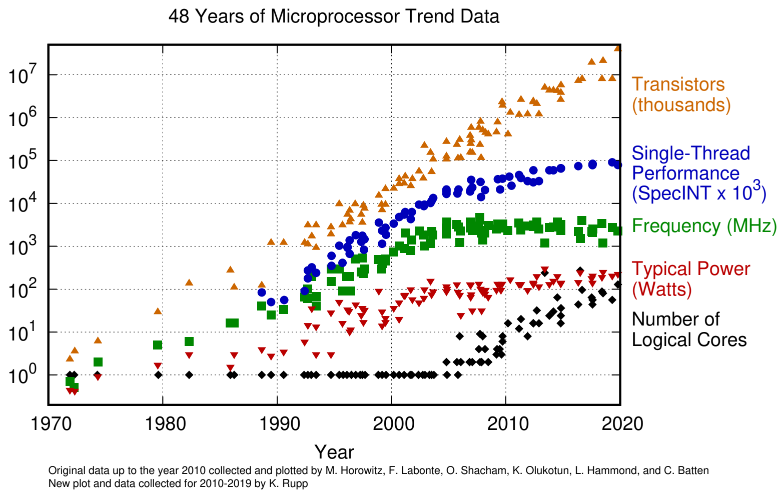
1. Introduction

From the colossus machines of 1943 [1], to the modern Internet of Thing (IoT) devices, there has been a massive growth in the variety of applications that use electronic computing systems. Every sector of our day-to-day lives—Consumer products, Telecommunication, Education, Agriculture, Healthcare, Automobiles, Military defence etc.—usually involves some form of information processing on electronic systems. While the scale of such information processing platforms can vary from small energy-harvesting IoT nodes to large data centres, the overall computing performance requirements for each of the application areas has undoubtedly increased in the last two decades. Prior to the 2000s, the major semiconductor manufacturers would answer the need for increasing performance requirements by technology scaling and micro-architectural enhancements only. Methods such as deep pipelining, increased cache sizes and complex dynamic Instruction Level Parallelism (ILP) were the primary tools as long as Dennard scaling could be achieved [2]. However, as shown in Figure 1a, the quest for higher clock frequency at lower technology nodes resulted in high power density and heat dissipation beyond the capacity of inexpensive cooling methods. This phenomenon, is often referred to as the power-wall [3]. Further, the continued technology scaling and micro-architectural innovations did not necessarily translate to faster memory technologies. As a result, the increasing gap between the compute clock frequency and the memory access frequency, referred to as the memory-wall, did not provide much benefits even at very high computation speeds. As a result, as shown in Figure 1b, the rate of improvement in single thread performance reduced considerably [4]. Further, the data dependencies of each application imposes implicit limits to the extent to which the application can exploit ILP. Traditional approaches to exploiting ILP such as deeper pipelines and branch prediction can exacerbate the power density problem while increasing the time and energy wastage due to branch mispredictions.

The cumulative effect of the three walls—memory, power and ILP—has led to diminishing returns from the efforts to provide performance scaling by increasing clock frequency of single core systems. As shown in Figure 1b, since 2005, the semiconductor industry has adopted the on-chip integration of multiple cores and processors as the weapon-of-choice for satisfying increasing computation complexity. In addition to using the ILP for each core, the multi-/many-core systems allow the software developer to exploit any form of thread-, task- and data-level parallelism in the application. Further, the technology scaling allows for additional application-specific cores to be integrated on the chip. The heterogeneity of the cores could be targeted for generic objectives such as power-performance trade-offs (e.g., big.LITTLE from ARM [5]) or for some specific computations (e.g., the CELL Broadband Engine [6]). However, the quest for increasing the number of on-chip transistors through technology scaling and improving performance of each core through architectural innovations have also increased the reliability issues across all electronic systems. Increased unreliability—either in terms of computation errors or the reduced lifetime of systems—has led to the increasingly complex problem of ensuring the reliable execution of applications on increasingly unreliable hardware [7].



(**a**)



(**b**)

Figure 1. Technology scaling and its impact. (**a**) Power density trends. (**b**) Impact of cheaper transistors [4].

1.1. Need of Reliability in Multi/Many-Core Systems

The increasing unreliability of electronic systems can be explained using the bath-tub curves shown in Figure 2. The reliability-specific life-cycle of typical electronic systems is characterized by three types of failures. The infant mortality, caused by premature failure of weak components as a result of manufacturing defects and burn-in testing, the constant failures due to random faults and the wearout-based faults due to aging. The solid curve in the figure shows the net effect of all three factors. With aggressive technology scaling, the rate of manufacturing defects has increased, resulting in higher infant mortality and higher susceptibility to aging-related faults. Similarly, the architectural innovations such as deeper pipelines and supply voltage scaling have reduced the clocking-window masking and electrical masking, respectively [8], thus increasing the constant failure rate due to random faults. Further, the parallel processing of multiple cores can increase the power dissipation resulting in accelerated aging. The net result of all these factors is the increased failure rate as shown by the dashed bath-tub curve in Figure 2. However, each of the these factors can manifest as degradation of different system-level performance metrics and the priority of each such metric may vary with each application.

For instance, in real-time systems the timeliness of execution has the highest priority. Similarly, in financial and scientific computations, the accuracy of calculations would be more important than the execution time. Additionally, in systems such as consumer products and space missions, extended lifetime of the system may have higher priority. Further, in a system executing multiple applications, each of the application may have varying criticality w.r.t. each reliability-related performance metric. In this scenario, the ideal solution would be to design a custom hardware platform for each application. However, from a market perspective, every application may not warrant the development of a custom hardware platform. Hence, standard multi-/many-core systems should be used for ensuring application-specific Quality of Service (QoS) requirements—both reliability-related and otherwise.

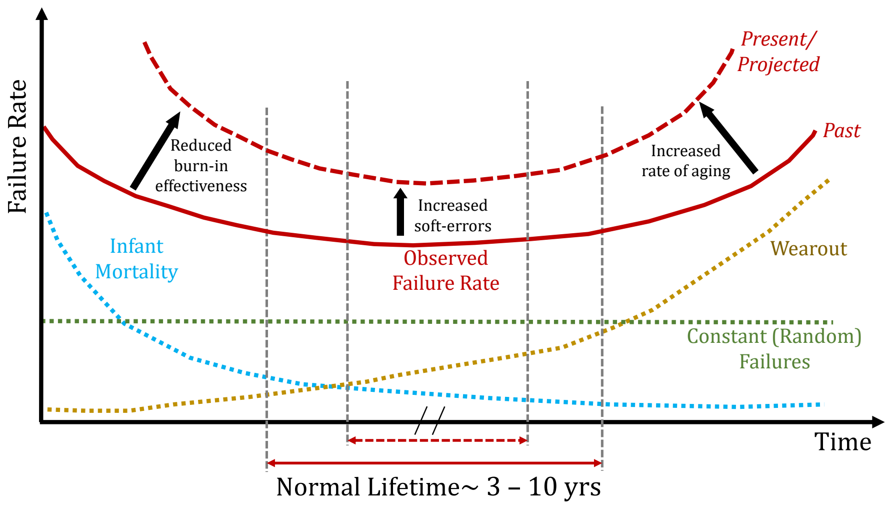


Figure 2. Increasing unreliability in electronic systems.

1.2. Reliability-Aware Resource Management in Multi-/Many-Core Systems

Given the variety of applications executed on multi-/many-core systems, optimal allocation of on-chip resources is an increasingly complex problem. We define this reliability-aware resource management problem as below:

Definition 1. *Reliability-aware resource management refers to the appropriate allocation of on-chip resources—computation, communication and memory—to applications/tasks executing on a multi-/many-core system while ensuring application-specific QoS and optimizing other system-level performance goals.*

The increasing unreliability in electronic systems has also resulted in a large body of research being devoted to ensuring reliable execution of applications. The research works related to solving the problem stated in Def. Figure 1 usually focus on one or more of following aspects:

1. Application Specificity: The varying priority among QoS metrics across different applications presents both a scope for application-specific optimization as well as challenges for ensuring application specific constraints.
2. Mixed criticality: Scheduling tasks with different criticality levels on a common platform is challenging, in which executing the tasks must be guaranteed in terms of both safety and real-time aspects to prevent the probability of failure and, consequently, catastrophic consequences.

3. Resource Heterogeneity: The heterogeneity of cores provides a scope for leveraging the availability of custom hardware implementations. Similarly, the availability of reconfigurable logic provides the scope for implementing accelerators on standard hardware platforms. However, such heterogeneity also introduces additional complexity for ensuring optimal resource sharing.
4. Design Space Exploration: All the above aspects introduce additional degrees of freedom in the design space. Consequently, the Design Space Exploration (DSE) for the joint optimization across all these aspects can result in an exponential increase in complexity.

In this article, we provide a survey of some of the more recent research works that explore these aspects. The rest of the article is organised as follows. We provide a brief overview of the relevant background and the taxonomy that is used through the rest of the article in Section 2. A generic system model and the generic problem statement along with the classification of the approaches is presented in Sections 3 and 4, respectively. We provide a detailed survey of related works across Sections 5–7. We present a brief discussion of emerging approaches to reliability management in Section 8. Finally, we conclude the article with a summary in Section 9.

2. Background and Taxonomy for Reliability Management Methodologies

The research works surveyed in this article aim to improve one or more aspects of reliability using different techniques of resource management under varying scenarios. The scenario can vary depending upon the executing application(s) and the resources available on the hardware platform. Similarly, the resource management could be aimed at maximizing different types of reliability and may be achieved by implementing various DSE methods. Figure 3 shows the taxonomy for the various terms that will be used frequently in the rest of the article for reviewing the related works. The tree-like structure in Figure 3 is used to categorize and show the relationships among these different aspects addressed by the related works discussed in this article. The figure also serves as a checklist for determining the scope and the assumptions used in each reviewed article. The colour coding of the rectangular boxes relates to the major aspects of reliability management as discussed in Section 1.2. The orange boxes show the terms related to reliability-types and causes. The gray and green boxes correspond to application and architecture scenarios, respectively. The blue boxes show the terms related to DSE. The current and the next section provide the background of and relationship (shown as arrows) among the terms shown in Figure 3.

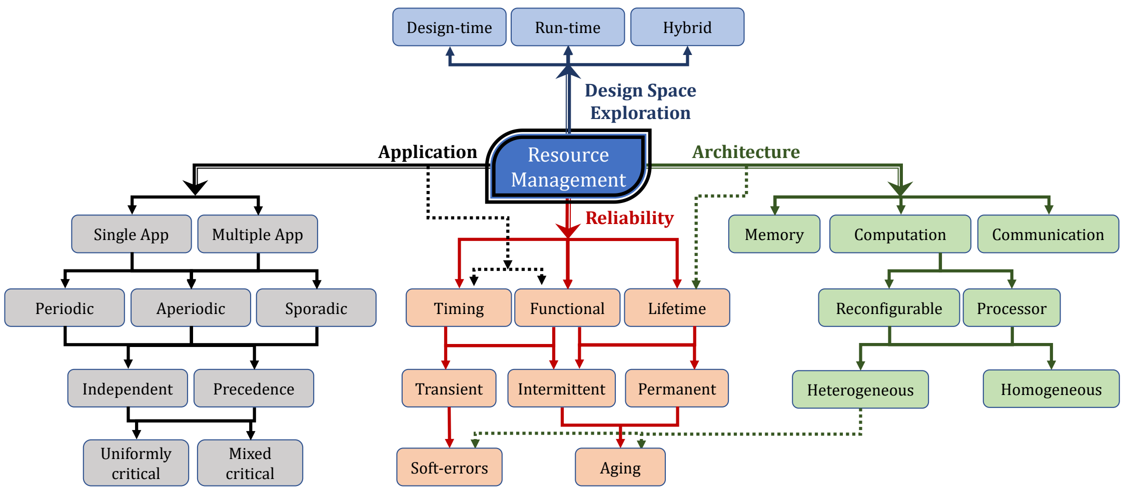


Figure 3. Taxonomy: The terms used in the article are categorized under four aspects of reliability-aware resource management—Application, Architecture, Reliability and Design Space Exploration.

2.1. Reliability in Electronic Systems

The rise in fault-rates in electronic systems has led to adverse effects on system performance in more than one way. Direct effects include application failures in terms of incorrect functionality and inability of the system to complete execution within the specified timing constraints. Similarly, indirect effects include over-designing for fault-mitigation—e.g., the high power dissipation of Triple Modular Redundancy (TMR)—leading to accelerated aging in the system and multiple re-executions of some critical tasks—for reducing chances of error—resulting in deadline violations. Given the variations in application-specific requirements across different application domains, the reliability-related QoS can be categorized as discussed next.

2.1.1. Lifetime Reliability

The expected operational life of the system can be characterized by its lifetime reliability. Depending upon whether the system is repairable, and the cost of such repairs, metrics such as Mean Time To Failure (MTTF), Mean Time To Crash (MTTC) and Availability can be used to characterize the system's lifetime reliability. MTTF refers to the expected time to the first observed failure in the system. In healthcare applications and consumer electronics, the need for predictable and extended MTTF can be the primary objective. Similarly, MTTC refers to the expected operational time for the point at which the system does not have sufficient resources for ensuring the expected behavior and is usually applicable for repairable systems. In applications with long mission times such as space exploration, repairing the failure mechanism is used to extend the MTTC. However, repair-time plays a critical role in high-availability applications such as automated control of power generation.

The reduced lifetime in electronic systems usually occurs due to the aging caused by electrical stress [9]. Most research works focus on improving the lifetime reliability by one/more of the following methods—reducing continuous computation on the processing elements, reducing the power dissipation, improving heat dissipation, reducing computations involved in executing the application etc. The availability of multiple processing elements enables improving the lifetime reliability by utilizing such techniques more effectively. For instance, the electrical stress on a single processing element can be reduced by using different resources for different execution instances. However, selecting the appropriate optimization technique along with the optimal configuration poses a significant research problem.

2.1.2. Timing Reliability

The performance of the system in terms of the expected behavior concerning the timeliness of execution completion can be expressed as its timing reliability. It is used only in terms of real-time systems and depending upon the criticality of the application, can be expressed in terms of Worst-case Execution Time (WCET), Mean Time between Failures (MTBF), Probability of Completion and Average Makespan [10–12]. WCET is usually used in hard real-time systems such as pacemakers and automobile safety features where any instance of missing a deadline can have fatal consequences. MTBF, frequently used in the context of repairable systems, can also be used for expressing the timing reliability in firm real-time systems such as manufacturing assembly lines, where infrequent failures can be tolerated, provided sufficient availability is ensured. Average makespan and probability of completion are usually used in soft real-time systems such as streaming devices and gaming consoles where frequent deadline misses can be tolerated as long as they do not affect user experience.

As more and more applications that require fast reaction times implement some level of automaton, for example autonomous driving, timing reliability can be a prime QoS metric for a large number of systems. Usual methods for improving timing reliability include faster execution, isolation of critical computation and communication etc. The spatial parallelism in many/multi-core systems can be used to exploit the inherent parallelism in an application and reduce the execution latency. However, distributing the computation across multiple processing elements introduces design complexities in terms of isolation and communication latency.

2.1.3. Functional Reliability

With the rising constant failure rates during the normal life of the system, the chances of such failures manifesting as incorrect computations has also increased. Hence, in applications that require high levels of computational accuracy such as financial transactions in point-of-sales systems or ATMs, scientific applications, the corresponding QoS can be expressed in terms of functional reliability. It concerns the correctness of the results computed by a system operating in a fault-inducing environment. The functional reliability can be quantified by the probability of no physical fault-induced errors occurring during application execution or the MTBF. Improving the functional reliability usually involves implementing one/more among–Spatial, Temporal and Informational redundancy. In many/muti-core systems, additional processing elements can be used to implement spatial redundancy effectively without considerable overheads on the execution latency. However, the increased power dissipation overheads and reduced spatial parallelism for computation resulting from this approach can adversely affect the other reliability metrics.

2.2. Fault Model

The reliability-specific events in a system can be classified as one of–failure, error and fault [13]. An application failure refers to an event where the service delivered by the system deviates from the expected service defined by the application requirements. An error refers to the deviation of the system from a correct service state to an erroneous one. Faults refer to the adjudged or hypothesized cause of the error. The physical faults in a system can be further classified into the following types, based on their frequency and persistence of occurrence.

1. *Transient faults* occur at a particular time, remain in the system for some period and then disappear. Such faults are initially dormant but can become active at any time. Examples of such faults occur in hardware components which have an adverse reaction to some external interference, such as electrical fields or radioactivity.
2. *Intermittent faults* show up in systems from time to time due to some inherent design issue or aging. An example is a hardware component that is heat sensitive—it works for some time, stops working, cools down and then starts to work again.
3. *Permanent faults* such as a broken wire or a software design error show a more persistent behavior than intermittent faults—start at a particular time and remain in the system until they are repaired.

The reliability-aware resource management presented in this article concerns the mitigation of physical faults only. Such physical faults, if unmasked, lead to errors which in turn may lead to application failures. The manifestation of all types of physical faults can be studied under two broad categories—Soft-errors and Aging. The effects of soft-errors caused by transient faults are considered for functional reliability analysis. Similarly, the aging-related intermittent and permanent faults are used in the analysis for lifetime reliability. Timing reliability issues occur usually due to aging-induced slower execution and additional computations that are employed to mitigate soft-errors.

2.2.1. Soft-Errors

Soft-error refer to non-reproducible hardware malfunctions caused by transient faults. The additional charge induced by external interference (such as alpha particles from radioactive impurities in chip packaging materials [14], and neutrons generated by cosmic radiation's interaction with the earth's atmosphere [15]) can sometimes (when $> Q_{crit}$) lead to changing the logic value of the affected nodes in the system. While in memory elements the changed value is retained until the next refresh, in combinational circuits the computations are affected only if the wrong value is latched by a memory element. The probability of such computational errors is reduced by either of the three masking effects—Logical masking, Electrical masking and Latching-window masking. In the quest for faster faster systems, although logical masking has remained unchanged, the deeper pipelines and aggressive voltage scaling have reduced latching-window masking and electrical masking, respectively, leading to increased Soft Error Rate (SER).

2.2.2. Aging

The term aging broadly refers to the degradation of semiconductor devices due to continued electrical stress that may lead to timing failures and reduced operational life of the integrated circuits (ICs). The primary physical phenomena causing aging are listed next [16].

1. Bias Temperature Instability (BTI) results in an increase in the threshold voltage, V_{th} due to the accumulation of charge in the dielectric material of the transistors [17]. The use of high-k dielectrics in lower technology nodes has resulted in an increased contribution of BTI to aging.
2. Hot Carrier Injection (HCI) occurs when charge carriers with higher energy than the average stray out of the conductive channel between the source and drain and get trapped in the insulating dielectric [18]. Eventually it leads to building up electric charge within the dielectric layer, increasing the voltage needed to turn the transistor on.
3. Time Dependent Dielectric Breakdown (TDDB) comes into play when a voltage applied to the gate creates electrically active defects within the dielectric, known as traps, that can join and form an outright short circuit between the gate and the current channel. Unlike the other aging mechanisms, which cause a gradual decline in performance, the breakdown of the dielectric can lead to the catastrophic failure of the transistor, causing a malfunction in the circuit.
4. Electromigration (EM) [19] occurs when a surge of current knocks metal atoms loose and causes them to drift along with the flow of electrons. The thinning of the metal increases the resistance of the connection, sometimes to the point that it can become an open circuit. Similarly, the accumulation of the drifted material can lead to electrical shorts.

3. System Model

The research works reviewed in this article, in general, aim at improving the reliability of the system. These improvements could be aimed at mitigating one or more types of faults across a subset of all the components of the system. In order to evaluate the impact of the proposed improvements, each of the works makes certain assumptions regarding the system components. We provide an overview of the system components under two models—Architecture and Application.

3.1. Architecture Model

The architecture model encapsulates all the features and the assumptions regarding the hardware platform. Figure 4 shows the components of a typical architecture model. The fault mechanisms discussed in Section 2.2 can affect different aspects of processing-computation, memory and communication.

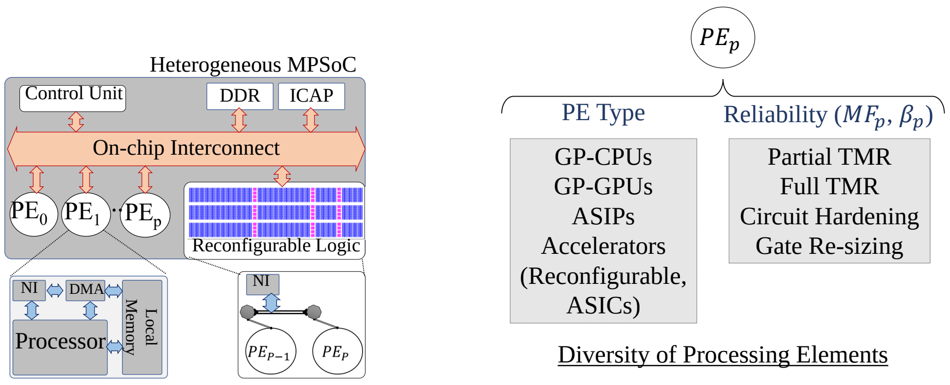


Figure 4. Architecture model.

3.1.1. Computation

As shown in Figure 4 we will use the term Processing Element (PE) to refer to each core/processor/accelerator implemented on the reconfigurable logic of the architecture. Soft-errors and the stuck-at faults due to aging directly affect the factional reliability. Similarly aging-related delays can result in timing violations. A more indirect impact of the timing and lifetime reliability can be observed as a result of fault-mitigation measures implemented for reducing errors in computation. Spatial redundancy measures such as TMR and Dual Modular Redundancy (DMR) result in higher power dissipation, thereby accelerating aging. Similarly, temporal and information-redundancy based methods introduce additional computations for processing the same workload and can led to degradation of both timing and lifetime reliability. The architecture may be homogeneous or heterogeneous w.r.t. the extent of spatial redundancy and other hardening techniques implemented in each of the PEs. Other types of heterogeneity may be due to the result of varying implementations for low-power design (ex. Big-Little from ARM), ISAs, ASIPs, etc.

3.1.2. Communication

As shown in Figure 4, we use the term on-chip interconnect to denote the set of communication-related components on the hardware architecture. Although the implementation of the on-chip interconnect may vary among bus-based, etc., with the rising number of PEs, Network-on-Chip (NoC)-based communication is being increasingly used in multi-/many-core systems [20]. Reliability of the on-chip interconnects involves ensuring error-free inter PE communication by using a combination of different types of redundancy methods across multiple layers of the Open Systems Interconnect (OSI) model. A detailed account of various reliability issues in on-chip interconnects and their mitigation can be found in [21,22]. Resource management of communication elements usually involves allocating links and routers to communicating tasks. Depending upon the reliability requirements and the criticality, the allocation algorithm may choose to provide different types of redundancy methods.

3.1.3. Memory

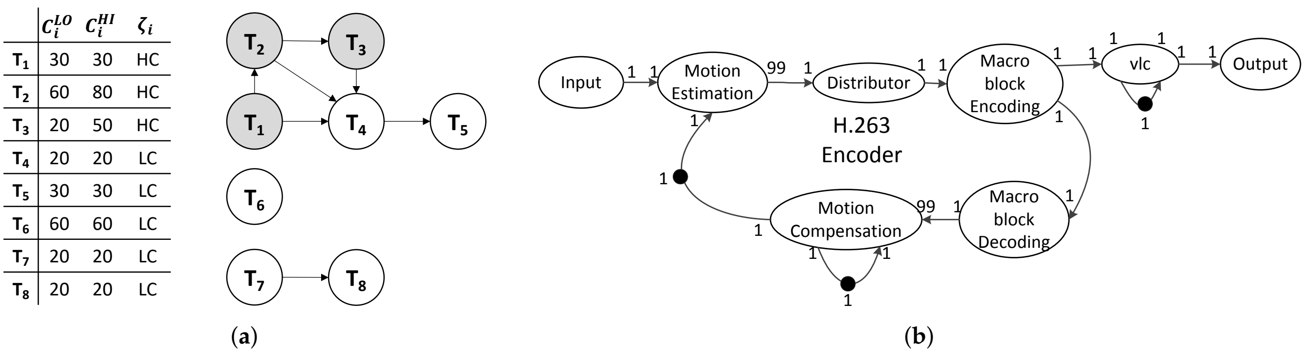
Similar to computation, soft-errors and stuck-at faults may result in incorrect values being read from the memory elements on the hardware platform, leading to reduced functional reliability. Information redundancy in the form of additional bits for Error Checking and Correcting (ECC) is commonly used for both Static Random Access Memory (SRAM)-based caches and Dynamic Random Access Memory (DRAM)-based main memory. Hamming [23] or Hsiao [24] code based Single-bit-Error-Correcting (SEC) and Double-bit-Error-Detecting (DED) codes are usually sufficient for most systems. More robust methods like Double-bit-Error-Correcting (DEC) and Triple-bit-Error-Detecting (TED) codes can be used for higher resilience against random bit errors. Storage overhead and power are the cost factors associated with design of a resilient memory system. Flexible error protection methods can enable adaptation to system-level requirements, both at design-time as well as run-time. ECC granularity and fault-coverage provide tunable parameters, while the memory controller acts as the tuning knob for varying error protection levels based on system requirements.

3.2. Application Model

In general, one application or multiple applications within a system can be executed on a platform according to its mission. These applications consist of different tasks whose properties are dependency, periodicity, execution time, and criticality. We introduce each property of these tasks briefly.

3.2.1. Task Dependencies

Tasks within an application can be dependent or independent to each other. Independent tasks mean that there is no precedence or communication among them, while the dependency represents the flow of data between tasks and induces a partial order on the task set. Such precedence relations are usually described through a Directed Acyclic Graph (DAG), where tasks are represented by nodes, and precedence relations by arrows. Figure 5a represents a combination of dependent (as DAG) and independent tasks. Synchronous Data Flow Graph (SDFG) is another common application representation that models cyclic dependency. This task model is used in streaming multimedia applications, in which support for pipe-lined execution are needed [12,25,26]. Figure 5b shows an example of SDFG for the H.263 encoder application [25,26], in which nodes are called actors. Each actor is executed by reading data from its inputs and writing the results as a token on the output port.



	C_i^{LO}	C_i^{HI}	ζ_i
T_1	30	30	HC
T_2	60	80	HC
T_3	20	50	HC
T_4	20	20	LC
T_5	30	30	LC
T_6	60	60	LC
T_7	20	20	LC
T_8	20	20	LC

Figure 5. Examples of task dependency. (**a**) DAG of UAV application [27]. (**b**) SDFG of H.263 encoder application [26].

3.2.2. Application/Tasks' Periodicity

Another property of tasks is their execution periodicity (i.e., the time of activation to be executed) that can be periodic, aperiodic, or sporadic. A periodic task consists of an infinite sequence of jobs, that are regularly activated at each period (i.e., the time to complete one iteration). An aperiodic task also consists of an infinite sequence of jobs, but their activations are not regularly interleaved. Sporadic tasks are aperiodic tasks where consecutive jobs are separated by minimum initiation time interval.

3.2.3. Application/Tasks' Criticality

In general, all tasks running on a common platform, may not be equally critical (i.e., not uniform criticality) for doing a correct service. Avionics, automotive and medical devices are examples of these systems. Indeed, due to the various safety demand for tasks, within an application, tasks can have different reliability requirements and criticality levels [28]. The systems with different criticality tasks are called mixed-criticality. In this regard, a set of industrial standards, e.g., DO-178B [29], has been introduced with five levels of safety, i.e., A, B, C, D, and E, (A and E provide the highest and the lowest levels of safety, respectively). A failure occurring in tasks with different criticality levels has a different impact on the system, which are shown in Table 1. The Probability-of-Failure-per-Hour (PFH) values (adopted by safety standards for tasks' safety measurement) have been determined for all the criticality levels to guarantee system safety.

In these mixed-criticality systems, the correct execution of tasks with higher criticality levels and QoS of tasks with lower criticality levels are considered, especially for service-oriented systems. Furthermore, mixed-criticality tasks may be real-time, i.e., the high-criticality tasks must be executed correctly before their deadlines to not cause catastrophic consequences. For tasks with lower-criticality levels, guaranteeing the deadlines is commonly regarded as a QoS parameter. Therefore, to ensure the correct execution of high-criticality tasks in any situation and the minimum QoS of low-criticality tasks while the system computation time is beneficially utilized, different WCET estimations, pessimistic (C_i^{HI}) and optimistic (C_i^{LO}), are used. As a result, different operational modes, according to the number of WCET estimations are considered for these systems. Figure 5a shows an example of mixed-criticality applications, Unmanned Air Vehicle (UAV), where some tasks are dependent on other tasks. In this application, T_1, T_2 and T_3 are the high-criticality (HC) tasks which are responsible for the collision avoidance, navigation, and stability of the system. Failure in the execution of these tasks may lead to system failure and cause irreparable damage to the system. Besides, low-criticality (LC) tasks ($\{T_4, \dots, T_8\}$ in the figure) are responsible for recording sensors data, GPS coordination, and video transmissions, to help the system carry out its mission successfully. Furthermore, as shown in this figure, since the correct execution of high-criticality tasks is crucial, two different WCETs are computed for them.

Table 1. DO-178B safety requirement [29].

x	A	B	C	D	E
PFH_x	$<10^{-9}$	$<10^{-7}$	$<10^{-5}$	$\geq 10^{-5}$	-
Failure Condition	Catastrophic	Hazardous	Major	Minor	No Effect

From the mixed-criticality system operation perspective, the system starts execution in the low-criticality mode in which all tasks are expected to finish their execution before their optimistic WCET (C_i^{LO}). If the execution of at least one high-criticality task exceeds its C_i^{LO}, while the correct output of the task is not ready, the system switches to the high-criticality mode and all task are expected to finish their execution before their pessimistic WCET (C_i^{HI}). Hence, since the requested demand for the system computation time is increased in this mode, some low-criticality tasks may be dropped to guarantee the correct execution of high-criticality tasks.

4. Reliability Management in Multi/Many-Core Systems

4.1. Problem Statement

The related research problem involves the optimization of the allocation of the available hardware resources to the computation, communication and storage requirements of an application(s) while satisfying the reliability and criticality constraints. We formulate an optimisation problem that serves as a generic framework, and the research works mentioned in this article are discussed with reference to it.

Application: Each application is represented as a tuple $G_g(\mathbb{T}_g, \mathbb{E}_g)$ containing the set of nodes $\mathbb{T}_g$ representing tasks/actors, and a set of directed edges $\mathbb{E}_g$, representing the precedence and the communication between the tasks. Similarly, a scenario denoting the applications executing in parallel can be represented by a set of applications $S_s = \{G_1, G_2, \dots\}$.

Architecture: Similar to an application, a mesh NoC-based hardware platform can be represented by a tuple $H(\mathbb{SW}, \mathbb{C})$. $\mathbb{SW}$ is the set of switches and $\mathbb{C}$ represents the connections among the switches. Each switch $Sw_s \in \mathbb{SW}$ can be attached to one or more cores. Assuming P_s cores are connected to Sw_s, the total number of cores in $H(\mathbb{SW}, \mathbb{C})$ is $|\mathbb{P}| = \sum |P_s|_{Sw_s \in \mathbb{SW}}$, where the set $\mathbb{P}$ denotes the collection of cores.

Allocation: Resource management involves timely allocation of appropriate hardware resources for all the tasks and communication in the scenario. We represent the resource allocation by two sets—(1) Task allocation: $T_{alloc} = \{(T_t, P_t, St_t), \forall T_t \in \mathbb{T}\}$, where $\mathbb{T}$ is the set of all tasks in the scenario, and (2) Edge allocation: $E_{alloc} = \{(E_e, C_e, St_e), \forall E_t \in \mathbb{E}\}$, where $\mathbb{E}$ is the set of all edges in the scenario. For a feasible solution, the sets T_{alloc} and E_{alloc} must satisfy certain constraints. The constraints may be of the following types:

- Precedence constraints: Any task must start execution only after all its preceding tasks and incoming communication is complete.
- Scheduling constraints: Properties such as each task/actor assigned to a single core, not more than a single task executing on a single core at a time etc. must be satisfied.
- Criticality constraints: PFH and reliability requirements should be corresponded with the criticality levels.

Performance Estimation: Varying the resource allocation results in varying system-level performance. Different methods of estimating these performance metrics—both analytical and empirical—as a function of T_{alloc} and E_{alloc} have been used across the works discussed in this article. We use the notation shown in Equation (1) to denote these methods. The estimated performance is used during the decision-making for reliability resource management. While the priority of each metric varies with application, Equation (2) shows the generic optimization problem. The terms $w_{\langle m \rangle}$ determine the application-specific priority of the system-level metrics ($\langle m \rangle$). Similarly, the terms $c_{\langle m \rangle}$ are used to denote the existence of constraints due to application-specific QoS requirements. The terms $\mathbb{T}_{alloc}$ and $\mathbb{E}_{alloc}$ represent the set of all possible task and edge allocations, respectively.

$$\begin{aligned}
\text{System-level Performance Estimation: Timing Reliability: } \mathcal{T}_{sys} &= funcR_T(T_{alloc}, E_{alloc}) \\
\text{Functional Reliability: } \mathcal{F}_{sys} = funcR_F(T_{alloc}, E_{alloc}); \text{Lifetime Reliability: } \mathcal{L}_{sys} &= funcR_L(T_{alloc}, E_{alloc}) \\
\text{Power Dissipation: } \mathcal{W}_{sys} = funcP(T_{alloc}, E_{alloc}); \text{Energy Consumption: } \mathcal{J}_{sys} &= funcE(T_{alloc}, E_{alloc})
\end{aligned} \tag{1}$$

$$\begin{aligned}
\underset{\forall T_{alloc} \in \mathbb{T}_{alloc}, \forall E_{alloc} \in \mathbb{E}_{alloc}}{\text{minimize}} \quad & \{w_{\mathcal{T}} \mathcal{T}_{sys}, w_{\mathcal{F}} \mathcal{F}_{sys}, w_{\mathcal{L}} \mathcal{L}_{sys}, w_{\mathcal{W}} \mathcal{W}_{sys}, w_{\mathcal{J}} \mathcal{J}_{sys}\} \\
\text{s.t.}, \mathcal{T}_{sys} &\geq c_{\mathcal{T}}\ \mathcal{T}_{SPEC}; \\
\mathcal{F}_{sys} \geq c_{\mathcal{F}}\ \mathcal{F}_{SPEC};\ \mathcal{L}_{sys} &\geq c_{\mathcal{L}}\ \mathcal{L}_{SPEC}; \\
\mathcal{J}_{sys} \leq c_{\mathcal{J}}\ \mathcal{J}_{SPEC};\ \mathcal{W}_{sys} &\leq c_{\mathcal{W}}\ \mathcal{W}_{SPEC};
\end{aligned} \tag{2}$$

4.2. Classification of Solution Approaches

The research articles discussed in this survey employ various methods for finding the optimal resource allocation. Further, each article focuses on optimizing a subset of the system-level performance metrics shown in Equation (2). In addition to providing an overview of the various approaches and prioritization (among different metrics) presented in each of the articles, we also classify them based on the criterion shown in Figure 6. Some of the criteria, such as fault-types, application model, architecture model, and criticality have been covered in the earlier sections. Additionally the articles can be classified on the experimental evaluation methodology. While some works use analytical methods to estimate the effectiveness of their proposed methods, others use simulation based approaches. In most of the cases real-world applications, in terms of standard benchmark suites have been used for the experiments. Along with reliability, other system level metrics such as power dissipation, energy consumption, throughput and thermal limits have been reported from the experiments as well. Further, some works—especially for reconfigurable systems—use resource utilization as an optimization objective.

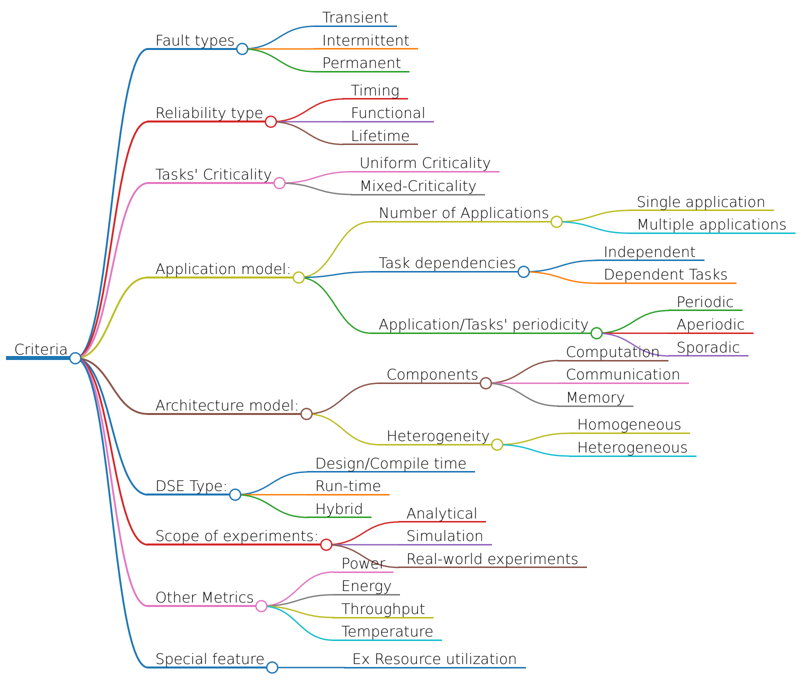


Figure 6. Classification criterion.

The DSE approach used in the research works can be categorized under (1) Design-/compile-time, (2) Run-time, and (3) Hybrid. While in most cases the methods used in each article can be classified under one of the three types, in some cases more than one of the approaches have also been used—typically for different objectives.

- Design-/compile-time: In this approach all the design decisions and the related optimizations are performed before the system is deployed. As shown in Figure 7, the related analysis is made under the design-time assumptions regarding both the application workload and the system's hardware resource availability. This approach allows the designers to generate highly optimized solutions. However, it also limits the adaptability of the system to dynamic operating conditions.
- Run-time: This approach involves implementing all resource allocation decisions only after the deployment of the system. This allows the system to adapt to varying operating conditions—both external and internal. External variations might include changing external radiation, changing workload resulting in varying QoS requirements etc. Internal variations include the changing performance/availability of the cores due to aging, as shown in Figure 7, low energy availability in mobile systems etc. In the run-time DSE approach the dynamic adaptability comes at the cost of result quality. Since all the resource management decisions are determined at run-time, it may result in sub-optimal solutions due to computation and availability constraints.
- Hybrid: The hybrid approach attempts to combine the best of both design-/compile-time and run-time approaches. It usually involves analysing most of the possible run-time operating conditions, finding the optimal solution for each condition of design time and storing the optimal solution to be used for run-time adaptation. As shown in Figure 7, the design-time analysis could involve determining the possible scenarios and determining the optimal solution for each scenario for varying core availability that might change during run-time due to aging and the resulting physical faults.

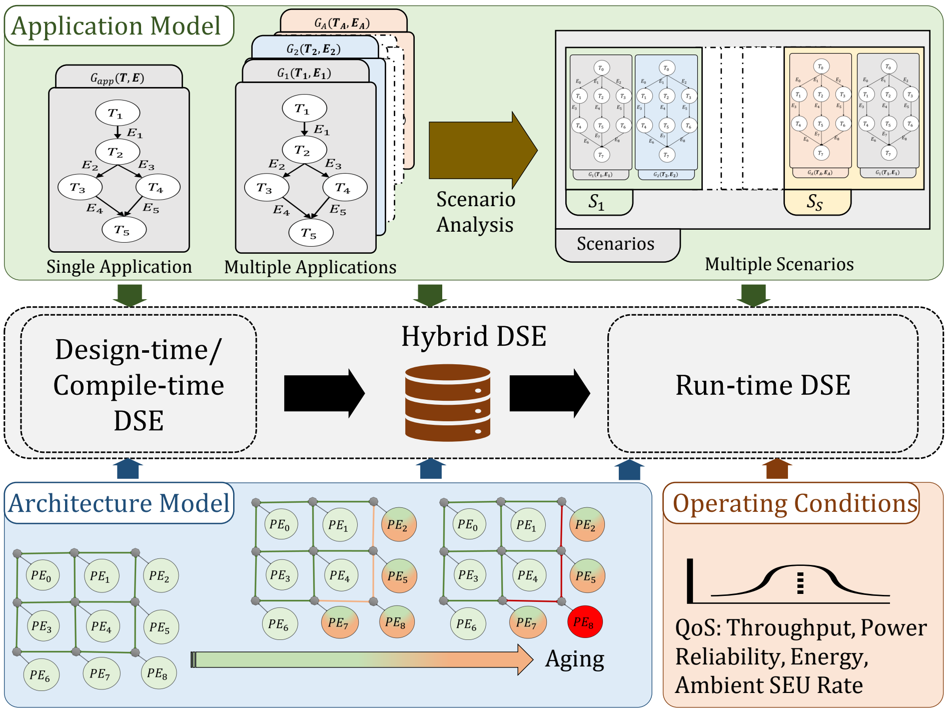


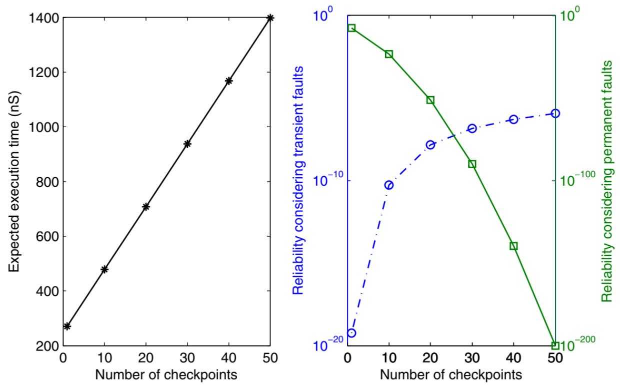
Figure 7. DSE approaches: Design-time/Compile-time, Run-time and Hybrid.

5. Lifetime Reliability Management in Multi/Many-Core Processors

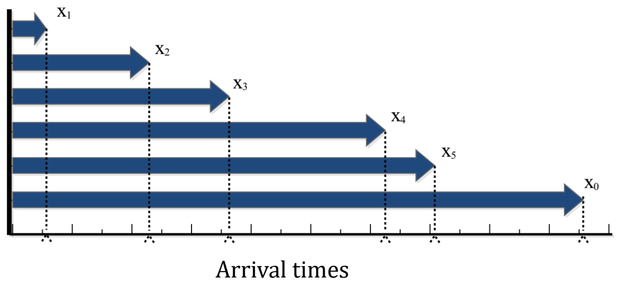
Resource management for improving lifetime reliability may involve direct approaches such as allocating spare cores in case of faults or indirect approaches such as wear-leveling and thermal management to delay the onset of faults. We summarize the related works under three categories; design-time approaches (D.T), run-time approaches (R.T), and hybrid (H), which we discuss in detail as follows. Table 2 lists the works in lifetime reliability management of these three categories for both uniform and mixed-criticality task models in detail. In this table, we also present the considered criteria in the state-of-the-arts, such as fault model (Transient, Intermittent, permanent), system model (Components and Heterogeneity), and application model (Dependency-Independent (I), Dependent (D), Periodicity-Periodic (P), Sporadic (S), Aperiodic (A)), that have been explained in detail, in Section 2.2, Section 3.1, and Section 3.2, respectively.

5.1. Design-Time Strategies

A purely design-time DSE approach to improving the system's lifetime reliability usually involves significant analytical estimation of workload stress. Furthermore, the scope of design-time analysis is limited by the assumptions of complete knowledge of the applications that will be executed on the hardware platform. Similarly, the estimation usually employs assumptions about the variability of the execution time of the various tasks in the application. The case for resource allocation explicitly targeting lifetime improvements is presented by [30]. Hartman er al. showed the improvement in system lifetime when task-mapping was optimized directly for reducing aging rather than trying to reduce the thermal stress in the system. The authors used an Ant Colony Optimization (ACO)-based design-time optimization of lifetime and compared the results with randomized task-mapping and task-mapping optimized for reducing the temperature using Simulated Annealing (SA). The authors attribute the improvements in the ACO-based approach to the negligence of the temperature optimization approach to factors such as supply voltage, current density and the aging-related interaction between the application and the architecture. Ref. [31] improved upon the lifetime optimization approach with task-mapping by determining the appropriate allocation of computation, communication and memory resources during the design of an NoC-based system for an application. Specifically, the authors have proposed the search for the optimal allocation of execution slack, storage slack and communication architecture during the hardware platform design. The authors extended this approach in [32] to perform joint optimization of system lifetime and the yield. While the optimization for lifetime involves slack distribution such that the system can survive most wear-out induced failures, the yield optimization involves surviving the most defect-induced failures in the system. The optimization methodologies presented in [30–32] use system-level simulation for estimating the lifetime as a result of the design decisions. In [33], Ma et al. presented a Multi-Armed Bandit (MAB)-based exploration for allocating less simulations for sampling of weaker solutions. The authors compared their approach against regular Monte-Carlo Simulations (MCS) in the optimization of lifetime-aware Multi-Processor System-on-Chip (MPSoC) design and report similar quality of results as MCS with up to 5.26× fewer samples. A more analytical approach, proposed in [34], for the estimation of system-level lifetime reliability in multi-core systems have found use in multiple research works that rely on design-time optimization for the DSE problem. In [35], Das et al., used the average execution time of each task, and the corresponding wear-out due to EM, to estimate the systems MTTF for varying task-mapping configurations and different Dynamic Voltage and Frequncy Scaling (DVFS) modes. The net aging effect on each core has been modelled as the average across all the tasks mapped to the core in every period (of periodic applications). The aging estimation methodology used in [35] is based on the techniques presented by [34]. Further, [26] use similar lifetime estimation approach to present a DSE methodology for showing the trade-off between permanent and transient fault-tolerance. Specifically, the authors explore the effect of temporal redundancy on the EM-related wear-out failures. Figure 8a shows the results from [26], depicting the effect of increasing number of checkpoints in the tasks of an application, on the average (expected) execution time and transient- and permanent-fault reliability. A more indirect approach of improving system lifetime by reducing the core temperatures is presented in [36], where the authors explore both the temporal and spatial effects of task-mapping on the peak temperature of a core. Recently, [37] have proposed a more generic DSE framework for joint optimization across varying types of redundancy methods and multiple design objectives. The authors presented the benefits of using a cross-layer optimization approach and proposed improved Multi-Objective Evolutionary Algorithms (MOEA)-based search methods for the large design space. Most design-time DSE works for lifetime reliability use the wear-out estimation for electromigration. Such EM-related wear-out effects are particularly disruptive for on-chip communication components. To this end, [38] proposed a dual physical channel switch architecture that was designed to improve the system's lifetime in the presence of permanent faults. The design-time analysis and optimization for mixed-criticality systems introduces the additional complexity of considering multiple criticality levels across the tasks. Related research for mixed criticality systems are discussed next.

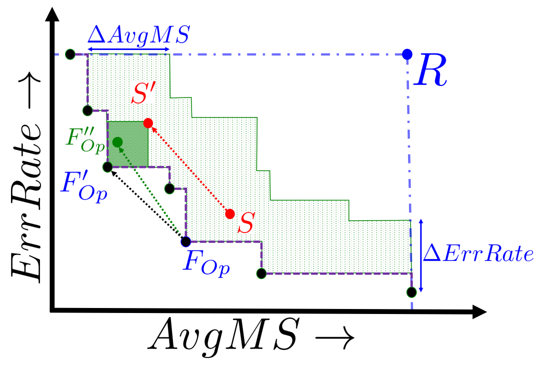


(**a**)



$$U\_val = \frac{\sum x_i/n - x_0/2}{x_0\sqrt{1/(12n)}}$$

(**b**)



(**c**)

Figure 8. Design space exploration for lifetime reliability. (**a**) Design-time DSE results in [26]. (**b**) Aging estimation for run-time DSE [39]. (**c**) Hybrid DSE [40].

Table 2. Summary of state-of-the-art approaches in lifetime reliability aware resource management.

	Fault Model			**App. Model**			**System Model**				**Imp.**		**DSE**	**Technique**
	Transient	**Intermittent**	**Permanent**	**Criticality**	**Dependency**	**Periodicity**	**Communication**	**Computation**	**Memory**	**Heterogeneity**	**Real Board**	**Real App.**		
Hartman'10 [30]	×	×	✓	×	D	P	×	✓	×	Het.	×	✓	D.T	Task Mapping
Meyer'10 [31]	×	×	✓	×	D	P	✓	✓	✓	Het.	×	✓	D.T	Slack Allocation
Meyer'14 [32]	×	×	✓	×	D	P	✓	✓	✓	Het.	×	✓	D.T	Slack Allocation, Yield
Ma'17 [33]	×	×	✓	×	D	P	✓	✓	×	Het.	×	✓	D.T	MAB Simulation
Das'14 [35]	✓	×	✓	×	D	P	×	✓	×	Hom.	×	✓	D.T	Task Mapping
Das'13 [26]	✓	×	✓	×	D	P	×	✓	×	Het.	×	✓	D.T	Task Mapping
Das'14 [36]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	D.T	Task Mapping
Sahoo'20 [37]	✓	×	✓	×	D	P	×	✓	×	Het.	×	×	D.T	Task Mapping
Kakoee'11 [38]	✓	×	✓	×	×	×	✓	×	×	×	×	✓	D(R).T	Hardware Redundancy
Hartman'12 [41]	×	×	✓	×	D	P	✓	✓	×	Het.	×	✓	R.T	Task (Re)-Mapping
Duque'15 [42]	×	✓	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Sahoo'16 [39]	×	✓	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Rathore'19 [43]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Venkataraman'15 [44]	×	×	✓	×	D	P	×	✓	×	Hom.	×	×	R.T	Hardware Migration
Wang'19 [45]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Haghbayan'16 [46]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Haghbayan'17 [47]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Rathore'18 [48]	×	×	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Raparti'17 [49]	×	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	R.T	(3D) Task Mapping
Bauer'15 [50]	✓	✓	✓	×	D	P	✓	✓	×	Het.	×	✓	R.T	Multi-layer
Das'13 [51]	×	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Das'16 [52]	×	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Das'13 [53]	×	✓	✓	×	D	P	×	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Bolchini'13 [54]	×	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Namazi'19 [55]	✓	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Kriebel'16 [56]	✓	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Task (Re)-Mapping
Nahar'15 [57]	✓	×	✓	×	D	P	✓	✓	×	Hom.	×	✓	H	Redundancy
Sahoo'19 [40]	✓	×	✓	×	D	P	×	✓	×	Het.	×	×	H	Task (Re)-Mapping
Axer'11 [58]	✓	×	✓	✓	I	P	×	✓	×	Hom.	×	×	D.T	Redundancy
Pathan'17 [59]	✓	×	✓	✓	I	S	×	✓	×	Hom.	×	×	D.T	Redundancy
Safari'20 [60]	✓	×	✓	✓	D	P	×	✓	×	Hom.	×	×	D.T	Redundancy
Saraswat'09 [61]	✓	×	✓	✓	I	P	✓	✓	✓	Het.	×	✓	R.T	Task Re-Mapping
Liu'13 [62]	×	×	✓	✓	I	P	×	✓	×	Hom.	×	×	R.T	Task Re-Mapping
Ranjbar'19 [63]	×	×	×	✓	D	P	×	✓	×	Hom.	×	✓	R.T	Thermal Management
Iacovelli'18 [64]	×	×	✓	✓	I	P	×	✓	×	Hom.	×	×	H	Task (Re)-Mapping
Bayati'16 [65]	×	×	✓	✓	I	S	×	✓	×	Hom.	×	×	H	Task (Re)-Mapping

Using redundancy in multi-core systems is one of the most useful techniques to manage permanent faults and consequently, lifetime reliability at design-time, used in [58–60]. Ref. [58] have presented a reliability analysis to tolerate soft errors by using checkpointing and redundancy techniques. The application is modeled as a set of independent tasks that consist of fault-tolerant tasks, which are more critical, and non-fault-tolerant tasks. In this paper, the lifetime reliability is represented by MTTF, and to improve the reliability, in addition to checkpointing, redundancy is used that each task is mapped two times perhaps on the same core (time redundancy) or different cores (hardware redundancy). To evaluate their method and show the improvement of lifetime reliability, the results are compared to a reference Monte-Carlo simulation. The work in [59] has presented a resource-efficient scheduling algorithm for independent safety-critical sporadic tasks. In this algorithm, first, the number of sufficient backups (multiple instances) for each task is determined to guarantee timing and lifetime reliability. Then, to reduce the processing resources consumption, the number of active backups for tasks are determined and the other backups are counted as passive. Hence, active backups can execute in parallel with primary tasks, while the passive backups would be executed in the case of occurring faults in active backups. In the end, the effectiveness of their proposed method is shown by using an example application.

In [60], researchers have presented an approach for dependent mixed-criticality tasks running on homogeneous multi-core processors, in which parallelism and redundancy policy have been applied for fault-tolerance. They enhance the reliability at design-time by using spare cores and tolerate both transient and permanent faults. Hence, they just improve the reliability of tasks with higher criticality by running task replica on the spare core. To allocate tasks on cores, first, all tasks are mapped to primary cores, and if the deadline of at least one high-criticality task is missed, the parallelism policy is used. It means the low-criticality tasks that can be scheduled concurrently are re-mapped one by one to spare cores until high-criticality tasks' deadlines are met. If there is still a high-criticality task whose deadline is missed, the reduction policy is used in which the QoS of low-criticality tasks is aggravated by reducing the worst-case execution time of low-criticality tasks on primary cores. To enhance reliability, they schedule the replicas of high-criticality tasks in the spare core by postponing the execution of replicas as much as possible. After allocating tasks on primary and spare cores, the free slack in each core is used to minimize the energy consumption by changing the V-f levels, such that the reliability constraints of criticality tasks would not be violated. Eventually, the presented method has been evaluated in simulation by using some random task graphs.

5.2. Run-Time Strategies

Run-time optimization of system lifetime usually involves periodically assessing the aging-level of each core and other components in the hardware platform and modifying the workload distribution in order to achieve wear-leveling. Wear-out estimation using special hardware structures and corresponding task-remapping to improve the system lifetime was proposed by [41]. The authors in [41] assumed the presence of wear sensors in every PE in the architecture and showed that the usage of such sensors instead of temperature sensors can improve the lifetime of a system by up to 14.6% compared to a thermal optimization approach. A heuristic scoring system, based on the weighted score from the wear sensors of the various architecture elements, is used for evaluating the candidate mapping solutions at run-time and perform the task-remapping accordingly. There have been more recent works that leverage the frequency of the occurrence of intermittent faults for wear-out estimation instead of using special hardware structures such as those used by [41]. For instance, ref. [42] presented a averaging window-based approach, that computes the average number of intermittent faults observed in each core over a fixed number of past execution cycles (moving window), to determine the wear-out level in each core. They ranked the cores in terms of their wear-out level to remap the tasks of an application accordingly. However, the evaluation in [42] involved experiments with the Infant Mortality stage of the system lifecycle (Figure 2), which primarily represents the initial failures due to manufacturing defects and burn-in tests, and not the wear-out region that represents the aging of electronic components. Ref. [39] improved upon this approach by using the Centroid test [66] over the arrival times of the intermittent faults in each core, as shown in Figure 8b. The centroid test presents a better statistical estimate of the wear-out of each core compared to a moving window approach. While the moving window approach suffers from a form of short-term memory, the centroid test is more reflective of the trends of aging seen by any arbitrary PE. Using this approach, considerable improvements over [42] were reported in [39]. Both [39,42] used the partial repairable nature of aging mechanism such as Negative Bias Temperature Instability (NBTI) to improve the lifetime both in terms of MTTF and MTTC. However, both [39,42] present the results with known workloads and do not consider the optimization for new application execution requests. Ref. [43] presented an aging-aware run-time task-remapping methodology that can cater to new application requests in addition to using a Reinforcement Learning (RL)-based online adaptation method for ensuring performance and improving lifetime. The authors present a learning based approach to keep track of the interactions between the tasks and cores to learn the process variations and the aging effects in the many-/multi-core system in order to determine the optimal operating frequency for each core. The authors show their approach to be scalable with the number of cores in the architecture. A novel approach to reducing the overheads during task-remapping due to permanent or intermittent faults was presented by [44]. The authors proposed a hardware-based task-migration technique that could reduce the re-mapping latency by up to 6.5×. A host of such run-time estimation and wear-leveling across multiple layers and different architectures are presented in [50]. Most works focused towards improving system lifetime treat the performance metrics such as throughput and latency as constraints and the lifetime as the optimization objective. However, ref. [45] presented a resource management approach that aims at optimizing the throughput under user-specified system lifetime constraints. The authors presented a run-time DSE methodology that implements a borrowing strategy to differentiate the resource allocation for compute- and communication-intensive applications. Specifically, the proposed method relaxes the short-term lifetime reliability constraints to improve throughput for communication-intensive applications while ensuring long-term lifetime of the system.

Most of the articles discussed in this section do not show the scalability of their proposed methods for many number of cores in the architecture. The scaling performance of these methods is especially important in the dark silicon era [67]. Dark Silicon refers to the phenomenon where, with each technology generation, the thermal and power limits of the system reduces the fraction of transistors that can operate at maximum frequency. Therefore, appropriate run-time resource management becomes crucial for enabling the useful integration of a large number of cores in the architecture. In [46], Haghbayan et al. present a methodology for lifetime-aware run-time task-mapping in many-core systems for the dark silicon era. The proposed technique involved running a long-term reliability analysis unit to track the aging of the cores along with a short-term re-mapping unit. The re-mapping unit utilizes the information from the analysis unit to provide longer recovery times to highly aging cores in the system. In [47], Haghbayan et al. extended this approach for the joint optimization of performance and lifetime reliability. Additionally, they adopted a hierarchical approach to task re-mapping where the first stage determined the appropriate region in the hardware and the second stage determined the appropriate PEs in the selected region to be used for mapping the application. A similar hierarchical approach to lifetime-aware run-time mapping of applications to many-core systems is presented by [48], where the authors intersperse the selected region with dark cores to maintain the thermal limits and reduce accelerated aging. An aging-aware run-time task-mapping for 3D NoC-based systems is presented in [49]. In this work, Raparti et al. consider the additional aging effects faced due to the 3D nature of the hardware platform. The higher current densities and limited number of power pins in such structures warrant special considerations for the EM-related aging of the power delivery network along with the aging of the PEs in the system.

Next, we discuss some related research in the context of mixed criticality systems. As we mentioned, one of the techniques to guarantee lifetime reliability is using task re-mapping at run-time. As shown in Table 2, Refs. [61,62] have used this technique that we explain each work in detail. Ref. [61] proposed a heuristic to manage the transient and permanent faults for mixed-criticality systems that tasks can be hard or soft (a task is said to be hard if missing its deadline may cause catastrophic consequences and, also a task is said to be soft if missing the deadline cause a performance degradation [68]). In this heuristic, checkpointing and roll-back recovery is used to tolerate transient faults and also task re-mapping to tolerate permanent faults. Meeting the deadlines of hard tasks and maximizing the QoS of soft tasks through task re-mapping are the targets of the heuristic. In the case of a permanent failure for a core, first the hard tasks and then soft tasks re-mapped to other healthy cores. Besides, researchers in [62] have proposed an algorithm for independent periodic mixed-criticality tasks to minimize the number of task re-mapping, while the most critical applications continue to meet their deadlines in homogeneous multi-core systems. When a core fails due to the permanent faults, first high-criticality tasks are re-mapped to other healthy cores, then, low-criticality tasks running on the processor may be re-mapped to other processors or even dropped to increase performance. Indeed, the algorithm makes a trade-off between the number of task re-allocations and the system's performance. The efficiency of the algorithm has been evaluated through simulation with a random task generation.

On the other hand, in [63,69], online peak power, and thermal management heuristic has been proposed, in which the re-mapping technique is used in the case of available dynamic slack to re-map a ready task from the hot core to a core with a lower temperature to manage the system's maximum temperature which improves the lifetime reliability. In addition, researchers have also proposed an approach to assign available dynamic slack to an appropriate task among k look-ahead tasks, which has more impact on system power and maximum temperature and reduce the V-f levels. The proposed method has been evaluated for dependent periodic mixed-criticality tasks (both real-life and random task set generation) in simulation. Figure 9 depicts an example of the thermal hotspot mitigation result of the system based on the proposed method in [63] and a state-of-the-art [27]). As shown, the proposed method can help in balancing the difference in temperature between the cores, which results in lifetime reliability improvement in the long-term.

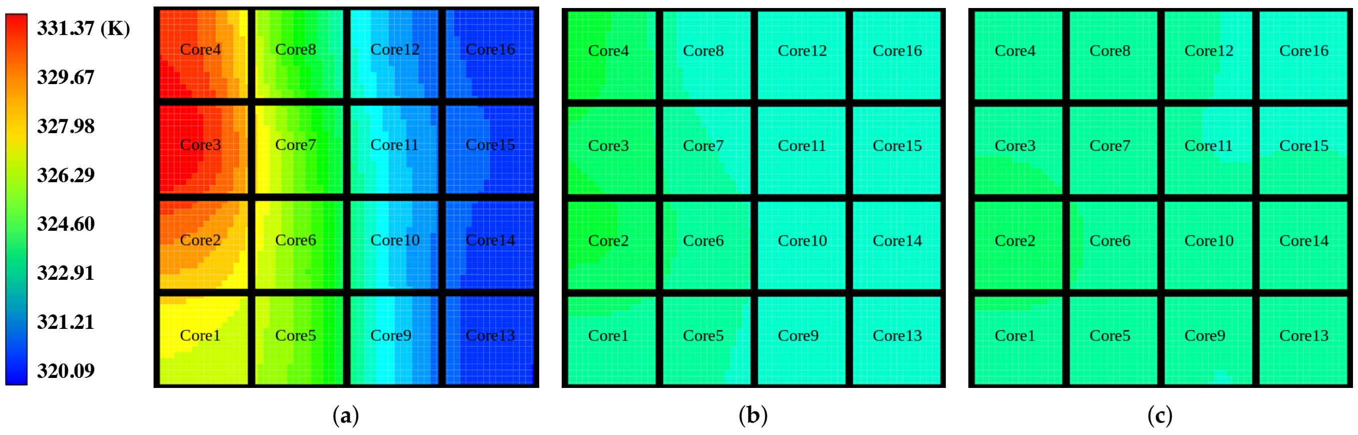


Figure 9. Temperature profiles of different approaches. (**a**) [27]. (**b**) [63], k = 1. (**c**) [63], k = 4.

5.3. Hybrid Strategies

The hybrid DSE approach to improving lifetime reliability usually involves searching and storing multiple system configurations for various fault/aging-scenarios at design-/compile-time that the system can be reconfigured to during run-time. Correspondingly, ref. [51] presented a methodology for reliability-driven task-mapping for MPSoCs that could be used for multiple applications as well. Their compile-time analysis used convex optimization to find the optimal task-mapping for different system states resulting from permanent faults to one or more cores. The run-time optimization in [51] involved considering the aging of the NoC for determining the appropriate task-mapping to be selected dynamically. The experimental evaluation in [51], involved testing the proposed methods for both DAG and SDFG representation of synthetic and real-world applications. Ref. [52] used a similar approach with the added design objective of energy consumption and the consideration of the thermal effect of adjoining cores on the reliability of any arbitrary core. In another similar work, ref. [53] presented hybrid DSE approaches that considered the occurrence of both permanent and intermittent faults during design-time analysis. The run-time optimization in [53] takes into account the energy consumption of task-remapping during the selection of the appropriate dynamic system configuration.

Similarly, Ref. [54] presented a task-remapping methodology for mitigation of core aging and reduction of communication energy. In addition to selecting and storing a set of Pareto-front points obtained from design-time optimization for performance and reliability, they proposed a fast heuristics-based run-time task-remapping for reducing energy consumption. A similar two-pronged approach to improve system lifetime is proposed in [55]. Here, the authors use the offline optimization to counter the effect of transient faults and the online re-mapping is aimed at migration-reduced adaptation to permanent faults. Similarly, in [56], Kriebel et al. propose an aging-aware resource management in the context of Redundant Multi-Threading (RMT). RMT allows the mitigation of soft-errors by executing copies of a task as two parallel threads. However, RMT can also lead to more aging of the core due to higher utilization. Kriebel et al. store multiple compiled versions of the application exhibiting varying vulnerability to soft-errors, aging and execution time. At run-time, the appropriate version is selected to be executed on an appropriate PE, along with the decision to enable/disable RMT, depending upon the vulnerability and aging-impact of the application. All of the works implementing run-time adaptation to failing PEs implement some form of task-migration. However, ref. [57] proposed a spatial redundancy based task-mapping approach that aims to remove the task-migration overhead completely by replicating the execution of tasks. In Nahar and Meyer [57], provide tolerance to both transient and permanent faults by ensuring that no single failure affects more than one copy of the redundant task. The authors report considerable improvements in the fault-tolerant lifetime of the system compared to traditional spatial redundancy-based methods such as TMR and DMR.

Most of the works in hybrid DSE store either the complete set or a subset of the Pareto-front points that are used during run-time. Recently [40] proposed a methodology where additional non-Pareto points are also stored by the system. These additional points were useful in providing configurations that resulted in lower reconfiguration time at the cost of slightly sub-optimal performance. Figure 8c shows the rationale behind this approach. In the figure, the system has to reconfigure due to changing requirements ($S \rightarrow S'$) at run-time. If only Pareto-front points were stored, it would result in the system switching from F_{Op} to F'_{Op} However there might be non-dominating points (such as F''_{Op}) that satisfy the new requirements while costing lower reconfiguration than F'_{Op}. Thereby, storing additional points within $\Delta ErrRate$ and $\Delta AvgMS$ around the Pareto-front points may result in better dynamic adaptation.

There are a few works [64,65], in which the researchers have taken advantage of both run-time and design-time phases to improve the lifetime reliability in mixed-criticality systems. In [64], researchers have considered an independent periodic task model that tasks with different criticality levels run on a homogeneous multi-core processor. The algorithm has two steps, task partitioning between cores and core utilization optimization. So, tasks are sorted in decreasing order of criticality and then assigned to the cores with the least load allocated. Indeed, the design-time algorithm adapts the resource shortage at run-time. Now, in run-time phase, if a permanent failure happens in a core, the algorithm must re-map tasks to other cores. In the case of not having enough space on the remaining cores, the heuristic drops first tasks with the least criticality and utility. In addition, the work in [65] proposed a Mixed Integer Linear Programming (MILP) based design space exploration process to design a reliable mixed-criticality system. At design-time, tasks are mapped, and the task schedulability in each core is tested by MILP. At run-time, to support the tasks running on cores from permanent faults, the re-mapping technique is used in run-time phase. Therefore, each high-criticality task would be run in two processors, primary and backup. When the primary processor fails, the high-criticality tasks on the failed processor are re-mapped to the predefined backup processor, and all low-criticality tasks assigned in the primary failed processor are dropped. Hence, this algorithm just supports a single processor failure.

5.4. Critique and Perspectives

Almost all the works discussed for improving lifetime reliability use one or more from a very limited set of techniques. These techniques involve either a reactive re-mapping of tasks in the event of a fault, or, pro-active distribution of workload for reducing the electrical stress on the individual cores to achieve wear-leveling. Most articles typically use an additional design objective along with system lifetime to present their novel methodology. While this does result in solving a problem of higher complexity, it does not necessarily contribute to improving system lifetime. Similarly most works listed in Table 2 ignore the reliability of communication and memory structures of the hardware platform. With increasing usage of NoCs for many-/multi-core systems, the reliability of communication elements should find more focus in research. Further, with emerging technologies shifting the focus to memory systems—for both storage and computation—reliability of memory elements needs more research. Finally, the evaluation methodology for lifetime reliability optimizations should include real world benchmarks that include more relevant applications from the domain of machine learning, computer vision etc. Further, lifetime reliability optimization for mixed criticality applications needs more direct approaches to improve system lifetime compared to the more prevalent indirect approach of thermal management.

6. Timing and Functional Reliability Management in Multi/Many-Core Processors

In this section, we aim to study the state-of-the-art works, which manage the timing or functional reliability in multi/many-core processors. In general, most papers have employed redundancy techniques, such as timing, hardware and information to guarantee the reliability in the systems. In the following, we present the existing criticality-aware and non-criticality-aware approaches in three categories of design-time, run-time, and hybrid strategies. Table 3 lists the works based on the criteria in detail. In the end of this section, we discuss about the critique and perspectives on the presented research works.

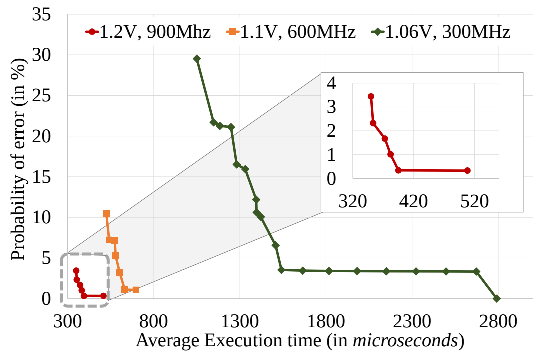
6.1. Design-Time Strategies

Some of works discussed under the lifetime reliability optimization earlier also analyze for functional and/or timing reliability as a design objective. For instance, ref. [26] used checkpointing with rollback recovery for mitigating the effect of transient faults on functional reliability. The analysis in [26] is aimed at determining the appropriate number of checkpoints in each constituent task of the application for providing sufficient functional correctness. Similarly in [35], Das et al. included the impact of DVFS on soft-error rate while varying the number of replications for each task. In a similar approach, ref. [37] proposed a methodology for an early stage evaluation of the impact of using multiple types of redundancies on the application's timing and functional reliability. In their analysis, ref. [37] integrated the effect of DVFS, imperfect fault-mitigation and implicit fault-masking across multiple layers. Figure 10a,b show the effect of DVFS and varying implicit masking on the average execution time and probability of error of a single task, respectively. Ref. [37] used a Markov Chain-based model to estimate the average execution time and the probability of error. A similar modelling approach for estimating the probability of task completion within a deadline was also presented recently by [70]. A task-mapping and priority assignment for similar deadline miss ratio-constrained systems has been proposed by [71]. In [71], the authors provide a design-time optimization for a heterogeneous architecture and use the stochastic execution time of tasks to optimize the task-mapping. A collection of methods for improving the fractional and timing reliability by using various mitigation methods across different layers, and with varying resource requirements, can be found in [50,72,73]. Similarly, research works targeting improved functional and timing reliability of on-chip communication include [38,74–76].

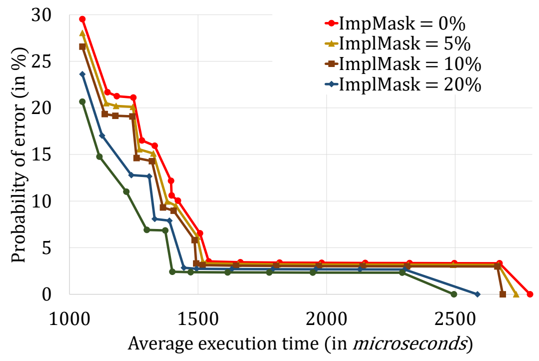
Table 3. Summary of state-of-the-art approaches in timing/functional reliability aware resource management.

	Fault Model			**App. Model**			**System Model**				**Imp.**		**DSE**	**Technique**
	Transient	**Intermittent**	**Permanent**	**Criticality**	**Dependency**	**Periodicity**	**Communication**	**Computation**	**Memory**	**Heterogeneity**	**Real Board**	**Real App.**		
Manolache’08 [71]	×	×	×	×	D	P	✓	✓	×	Het.	×	✓	D.T	Task Mapping
Das’14 [35]	✓	×	✓	×	D	P	×	✓	×	Hom.	×	✓	D.T	Task-Replication
Bauer’15 [50]	✓	✓	✓	×	D	P	✓	✓	×	Het.	×	✓	R.T	Hardware blackundancy
Das’13 [26]	✓	×	✓	×	D	P	×	✓	×	Het.	×	✓	D.T	Checkpointing
Sahoo’20 [37]	✓	×	✓	×	D	P	×	✓	×	Het.	×	×	D.T	Cross-layer Redundancy
Frantz’07 [74]	✓	×	×	×	×	×	✓	×	×	×	×	✓	D.T	HW/SW Redundancy
Lehtonen’07 [75]	✓	✓	✓	×	×	×	✓	×	×	×	×	✓	D.T	Fault-spec. Opt.
Vitkovskiy’10 [76]	×	×	✓	×	×	×	✓	×	×	×	×	✓	D(R).T	Latency Reduction
Kakoee’11 [38]	✓	×	✓	×	×	×	✓	×	×	×	×	✓	D(R).T	Hardware Redundancy
Duque’15 [42]	×	✓	✓	×	D	P	×	✓	×	Hom.	×	✓	R.T	Task (Re)-Mapping
Wells’08 [77]	×	✓	×	×	I	S	×	✓	×	Hom.	×	✓	R.T	Hardware Redundancy
Lehtonen’10 [78]	×	×	✓	×	×	×	✓	×	×	×	×	✓	R.T	Online Testing
Yu’10 [79]	✓	×	✓	×	×	×	✓	×	×	×	×	✓	R.T	Hardware Redundancy
Rehman’16 [80]	✓	×	×	×	D	A	×	✓	×	Hom.	×	✓	H	Cross-layer
Weichslgartner’18 [81]	×	×	×	×	D	A	✓	✓	✓	Het.	×	✓	H	Task Mapping
Sahoo’19 [40]	✓	×	✓	×	D	P	×	✓	×	Het.	×	×	H	Cross-layer Redundancy
Pourmohseni’19 [82]	×	×	×	×	D	P	✓	✓	×	Het.	×	✓	H	Task Mapping
Pathan’17 [59]	✓	×	✓	✓	I	S	×	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Safari’20 [60]	✓	×	✓	✓	D	P	×	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Safari’19 [83]	✓	×	×	✓	I	P	×	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Rambo’17 [84]	✓	×	×	✓	I	P	✓	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Bolchini’13 [85]	✓	×	×	✓	D	P	✓	✓	×	Het.	×	×	D.T	Hardware Redundancy
Kang’14 [86]	✓	×	×	✓	D	P	✓	✓	×	Hom.	×	✓	D.T	Hardware Redundancy
Kang’14a [87]	✓	×	×	✓	D	P	✓	✓	×	Het.	×	✓	D.T	Hardware Redundancy
Choi’18 [88]	✓	×	×	✓	D	P/S	×	✓	×	Hom.	×	✓	D.T	Hardware Redundancy
Jiang’18 [89]	✓	×	×	✓	D	P	✓	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Zeng’16 [90]	✓	×	×	✓	I	S	×	✓	×	Hom.	×	✓	D.T	Hardware Redundancy
Caplan’17 [91]	✓	×	×	✓	I	S	×	✓	×	Hom.	×	×	D.T	Hardware Redundancy
Axer’11 [58]	✓	×	✓	✓	I	P	×	✓	×	Hom.	×	×	D.T	Timing Redundancy
Saraswat’09 [61]	✓	×	✓	✓	I	P	✓	✓	✓	Het.	×	✓	D.T	Timing Redundancy
Saraswat’10 [92]	✓	×	×	✓	I	P	✓	✓	✓	Het.	×	×	D.T	Timing Redundancy
Bagheri’14 [93]	✓	×	×	✓	D	-	✓	✓	×	Hom.	×	×	D.T	Timing Redundancy
Kajmakovic’19 [94]	✓	×	×	✓	×	×	×	×	✓	×	✓	×	D.T	Information Redundancy
Liu’19 [95]	✓	×	×	✓	D	P	✓	✓	×	Het.	×	×	D.T	Rel.-Aware Mapping
Thekkilakattil’14 [96]	✓	×	×	✓	I	P	×	✓	×	Hom.	×	×	H	Mapping & Redundancy
Koc’19 [97]	×	×	×	✓	D	P	×	✓	×	Het.	×	×	H	Mapping & Redundancy

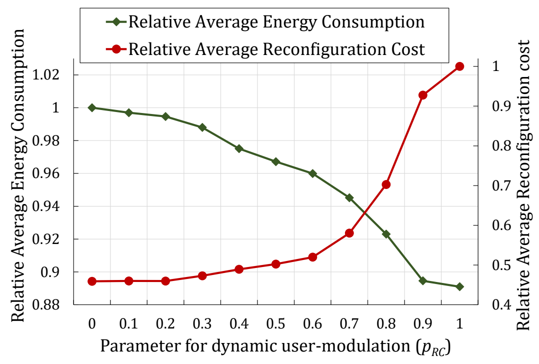
Next, we study the prior works that managed the timing or functional reliability of mixed-criticality systems in the design-time phase. Most of the papers that exploited the multi/many-core processors achieve reliability improvement by using redundancy technique, such as hardware redundancy (using replica, which is Active, Passive, or Hybrid), timing redundancy using re-execution after error detecting and check-pointing with roll-back recovery, and information redundancy by using the addition of redundant information to data to mitigate soft error. A summary of the literature based on their exploiting techniques are detailed as follows.



(**a**)



(**b**)



(**c**)

Figure 10. DSE for timing and functional reliability. (**a**) Reliability and DVFS [37]. (**b**) Implicit masking [37]. (**c**) Hybrid DSE results [40].

6.1.1. Multi-Core Platform Used for Spatial and Temporal Redundancy

Two types of redundancy, which are used in the most papers [59,60,83–91] for tolerating faults and consequently, improving reliability are timing and hardware. Some papers have used the feature of multi-core platforms and applied just the hardware redundancy, i.e., replication [59,60,83–85]. As we mentioned in Section 5, authors in [59,60] have used backup redundancy to improve timing and lifetime reliabilities, while the deadline of tasks is guaranteed. In [83], a scheme has been presented to minimize energy, guarantee reliability by computing the optimum number of replication for each criticality task, and maximize QoS when the systems switch to the high-criticality mode. To map tasks on cores, first high-criticality tasks and their replicas and then, low-criticality tasks are mapped on cores that are selected based on worst-fit decreasing and first-fit decreasing. Safari et al. claimed that worst-fit decreasing is the best policy from the energy-awareness perspective. The effect of this algorithm is validated through simulation based on random task generation. Besides, ref. [84] have presented a replica-aware co-scheduling method for mixed-criticality systems by exploiting cross-layer fault tolerance mechanisms. This method has supported network-on-chip communication delay and replication management overheads. In addition, ref. [85] have presented a methodology to map and schedule tasks on heterogeneous multi-core processors and optimize overall performance. In this methodology, different fault management techniques (Fault Detection/Tolerance) are exploited on different portions of the task graph. Indeed, for some parts of a task graph that needed to be tolerated against faults, different reliability improvement techniques such as replication would be exploited based on architecture features.

On the other hand, there are some works [86–91] that considered both hardware and timing (re-execution) redundancies to tolerate faults before tasks' deadlines and improve timing reliability. In [86,87], an offline heuristic for mapping optimization has been proposed for dependable tasks with different reliability requirements and tolerated transient faults and, consequently, guaranteed the tasks' reliability before their deadlines. The number of re-execution or replication for each task is defined based on its criticality. Besides, ref. [88] have presented a framework to find an optimal mapping of dependent mixed-criticality tasks on multi-core processors, while the QoS is increased in the faulty state and also the power consumption in the normal and faulty states are minimized. To tolerate the transient faults based on the probability distribution of fault occurrences, re-execution and replication are used, and also, the algorithm is designed to endure the maximum number of faults. Choi et al. use the genetic algorithm to find the optimum mapping while all objectives are guaranteed. To investigate the fault tolerance techniques on message communication between dependent tasks, ref. [89] have proposed an optimization to map tasks, minimize the scheduling length and application security vulnerability by considering fault-tolerant constraints. Two techniques of re-execution and active replication are used to tolerate faults of tasks.

Besides, ref. [90] find the optimum number of replication and re-execution for each criticality task to guarantee the reliability of tasks based on the DO-178B safety requirements, in which PFH is defined as their metric. In [91], the authors have used this metric to guarantee the reliability of high-criticality tasks in the case of fault occurrence in different situations. In that paper, an efficient mapping and scheduling algorithm based on the genetic algorithm is proposed on heterogeneous multi-core platforms, while transient faults are tolerated by using on-demand redundancy. In on-demand redundancy, three types of Dual Modular Redundancy, Triple Modular Redundancy, and Passive Replication are supported. In this algorithm, all low-criticality tasks must be executed in a normal situation. If the system is overloaded or a fault occurs, these tasks can be dropped to guarantee the correct execution of high-criticality tasks. Hence, the QoS of low-criticality tasks is one of the objectives of this algorithm that would be maximized.

6.1.2. Timing Redundancy with Check-Pointing and Rollback Recovery

The authors in [58,61,92,93] have improved the reliability by using Check-pointing and rollback recovery for mixed-criticality systems in multi-core processors. Using the Check-pointing technique helps to tolerate transient faults and guarantee reliability. At design-time, authors find the optimum number of check-points during the execution of tasks, such that the deadline of tasks would be guaranteed. In the case of faults occurring during run-time phase, the faulty task is recovered from the previous check-point and continues its execution. In [92], a Tabu search-based approach for task mapping is presented, in which the deadlines of the hard tasks (higher criticality tasks) are guaranteed, even in the case of transient faults, and the QoS for the soft tasks (tasks with lower criticality) is maximized. To improve the timing reliability of tasks with higher criticality, check-pointing with rollback recovery is used. The optimum number of check-points is calculated by considering the overheads of establishing the check-point, error detection, and recovery, while the task deadlines are guaranteed. The proposed algorithm has been evaluated with several random and real-life benchmarks. In [93], a framework of dependable NoC-based multiprocessor is designed for mixed-criticality DAG task models, in which the inter-task communication has been considered. Bagheri and Jervan [93] guarantee the deadline of just high-criticality tasks even in the presence of transient faults. Transient faults are tolerated by check-pointing, and also, its timing overhead is considered as part of WCET.

6.1.3. Information Redundancy, Mitigating Soft Errors by Using Parity

From the perspective of reliability improvement of memories, ref. [94] have considered soft errors and used redundant parity bit to detect and recover the data utilizing the parity bit. Considering mixed-criticality for memories allows us to improve the system reliability and increase the protection of more criticality memory parts. In this work, faulty data would be corrected in the minimum time to maximize the performance and minimize the run-time memory overhead. Indeed, this parity approach to detect soft errors is explored to use in the design-time phase. In the case of detecting fault at run-time, faulty data recover by copying healthy data. Hence, the Kajmakovic et al. have not used any additional hardware components to tolerate faults.

6.1.4. Task Reliability-Aware Mapping

Another technique to guarantee the reliability of tasks is finding an efficient mapping of mixed-criticality tasks on heterogeneous multi-core platforms to ensure the timing reliability and minimize the probability of fault occurrence. In [95], a heuristic has been proposed that the reliability requirements are satisfied, and also the deadline miss ratio of high-criticality tasks is minimized. In this heuristic, the authors find the optimum mapping and scheduling of tasks with different criticality and reliability requirements on multi-cores with different reliability levels to reduce the probability of transient fault occurrence. The proposed heuristic efficiency has been validated through simulation and shows an outstanding reduction in the deadline miss ratio.

6.2. Run-Time Strategies

Run-time adaptation for improving functional and timing reliability usually involves varying the redundancy levels and using faster cores, respectively. While achieving better functional reliability with a purely run-time approach may be achieved by avoiding faulty cores and/or replicating task execution (both spatially and/or temporally), guaranteeing/improving timing reliability requires much more analysis and is better achieved with a hybrid approach. For instance, multiple research works involve detection of permanent faults in cores and re-mapping tasks to healthier cores [39,41,42]. Similarly, ref. [77] proposed multiple mitigation approaches for intermittent faults including pausing execution, using spare cores and avoiding faulty cores to allow for self-repair. Similarly, ref. [50] proposed multiple resource management methods that include both proactive (avoiding hot-spots) and reactive (online testing and error detection) methods. Some of the approaches proposed in [50] also include using TMR for improving functional reliability. Similar redundancy based improvement of on-chip communication by using information encoding and spare wires are proposed by [75]. A novel methodology for online testing, detection and bypassing of permanent faults in NoCs is presented in [78].

From the run-time criticality-aware reliability management perspective, ref. [98] have recently investigated MC systems. In this paper, authors have presented the Information Processing Factory (IPF) paradigm to achieve long-term dependability for MC systems, where a 5-layer hierarchical organization has been introduced. IPF is introduced as a self-aware and self-organizing system used to manage resources by decomposing approach, planning, and confining them during run-time. The IPF can also detect and predict potential hazards and handle these upcoming risks in different layers. As a result, in this introduced framework, the requirements of safety-critical functions are always met at run-time. In the end, the proposed method efficiency has been evaluated by showing achieving the reliability levels (functional reliability) and MTTF as lifetime reliability.

6.3. Hybrid Strategies

The design-/compile-time analysis for improving functional reliability involves determining the different levels of error tolerance provided by varying levels of redundancy. For timing reliability, the analogous stage involves finding the various resource allocation configurations that ensure the timing requirements of the application(s). Ref. [80] proposed a cross-layer reliability approach in single-processor systems, where multiple executables for each task, with varying execution time and error probabilities, were generated at compile time. The run-time allocation involved dynamically selecting the appropriate version of each task, depending upon the available slack w.r.t. the deadline. Ref. [81] also tackled the problem of achieving predictable execution time in MPSoCs with a hybrid approach. The authors presented a compile time DSE methodology for generating clustered tasks and constraint graphs based on the timing requirements of the application (modelled as a DAG). The run-time management involved mapping the clustered tasks and edges on available cores and interconnects of the hardware platform. A similar approach to mapping of hard real-time applications on many-core system using a hybrid approach was presented by [82]. A novel methodology for using hybrid DSE to adapt to varying QoS requirements was proposed by [40]. In addition to ensuring timing and functional reliability, the run-time process proposed in [40] allowed the user to dynamically select the priority of energy consumption and reconfiguration cost during run-time adaptation. Figure 10c shows the different trade-offs obtained for a sample application by varying the user-controlled parameter p_{RC}.

From the perspective of criticality-aware strategies, some papers have improved the timing reliability for mixed-criticality tasks in both design-time and run-time phases [96,97]. For example, ref. [96] have presented an approach to map and schedule tasks with different criticality levels on multi-core processors in which the timing constraints of high-criticality tasks are guaranteed at design-time even in the case of fault occurrence and also, the flexibility for the low-criticality tasks are ensured. To tolerate the faults and improve reliability, the timing redundancy technique, re-execution, is used for high-criticality tasks at design-time and low-criticality tasks at run-time. Besides, the work [97] has focused on finding the best mapping of mixed-criticality tasks to minimize execution latency by considering the reliability of both system and high-criticality tasks at design-time. At run-time, when the system switches to the high-criticality mode, high-criticality tasks are re-mapped to the highly reliable cores to be executed before their deadlines and, if possible, low-criticality tasks are scheduled without exceeding the minimum latency.

6.4. Critique and Perspectives

Most works in this category have evaluated their proposed approaches in simulation, which can be seen in Table 3. Although simulation can validate the proposed method good enough, it is not sufficient due to some reasons such as overheads at run-time, like communication, and fault detection and tolerance; then the proposed method may not be applicable in some cases. Researchers in [69] have discussed that if the timing overheads, such as the delay for changing the V-f levels of cores, are not considered while scheduling the tasks, it may cause deadline violation and consequently, catastrophic consequences may happen. Therefore, evaluating the proposed methods based on real applications on a real platform is needed, which has not been considered in most existing works.

Besides, as mentioned in this section, most of the works have used redundancy techniques to guarantee timing or functional reliability. Since the system may execute safely without fault occurrence, these techniques such as hardware and timing redundancy may waste the system's resources.Therefore, investigating the run-time approaches is needed to efficiently use the computational resources and optimize the other objectives.

In addition, intermittent faults are one of the common faults in embedded systems. The reason for these faults can be inherent design issue or unstable hardware. Due to the behavior of repeated conditions of causing faults, errors may occur. Investigating the intermittent faults can help the system be optimized more efficiently at run-time, which has not been considered in previous works.

From the criticality-aware reliability management perspective, most of the existing works have guaranteed the timing reliability only for the tasks with higher criticality tasks in any system operational modes. However, in some mixed-criticality embedded systems, such as avionics, low-criticality tasks, are mission-critical, and guaranteeing the reliability requirement of both low- and high-criticality tasks are crucial. Most of the existing approaches cannot be applied to these systems; thus, new studies to guarantee each criticality level's reliability in each operational mode are required in multi-core mixed-criticality systems.

As can be illustrated from Table 3, most of the works have not considered the communication and memory of multi/many-core processors and only focused on guaranteeing the reliability in computational parts. However, the memories and data sharing can be the bottleneck to guarantee the reliability of applications in multi-core platforms. Designing the system for reliability management and analyzing it at run-time by having a comprehensive view on the whole system resources are required in multi/many-core platforms.

7. Reliability Management in Reconfigurable Architectures

Some of the related works discussed till now assume the availability of reconfigurable logic on the hardware platform and tackle the related problem of resource management by allocating accelerators to a select subset of tasks of an application (hardware/software partitioning). Therefore the DSE methods presented in works such as [26,40,50] can be directly used. However, in this section we survey the works that assume the complete architecture comprising of reconfigurable hardware logic, specifically for Field Programmable Gate Array (FPGA)s.

Improving functional reliability in FPGAs has mostly been focused on improving the reliability of the configuration bits. Traditional methods methods such as ECC [99], scrubbing [100] and hardware checkpointing [101] have been used to provide protection from transient faults in the configuration memory. Additionally, circuit design methods that involve TMR has also been employed for enabling the usage of FPGAs in high-radiation environments [102]. However, there has been a growing trend of lifetime reliability improvement in FPGA-based systems. For instance, ref. [103] proposed various phenomenon-specific methods, tailored for each failure mechanism, to mitigate aging in FPGAs. Similarly, ref. [104] proposed multiple generic electrical stress hot-spot reduction techniques for FPGAs. Ref. [105] presented a stress-aware run-time wear-leveling approach that leverages Dynamic Partial Reconfiguration (DPR) in FPGA-based systems. In [106], Zhang et al. used module diversification, to generate multiple accelerator designs with spatially varying aging effects. These diverse modules were used to leverage DPR by periodically swapping accelerators that use different CLBs of the FPGA fabric. A novel approach combining module diversification and dynamic adaptation to varying aging-effects at run-time was proposed by [107]. Similarly, ref. [108] presented a reliability-aware floorplanning methodology along with delay-based aging estimation and run-time reconfiguration. A joint mitigation methodology using DPR, aimed at both soft errors and permanent faults in FPGAs was proposed by [109]. The authors presented reconfiguration as a solution to both types of faults, which can be a costly approach. Almost all the research works employing DPR assume using homogeneous Partially Reconfigurable Region (PRR) (comprising of equivalent amount of FPGA resources). However, as shown in Figure 11, Sahoo et al. [110,111] proposed a hardware/hardware partitioning methodology that allows using application specific heterogeneous PRRs, that provided the scope for improving both the latency (average makespan) and reliability (MTTF) in DPR-based systems.

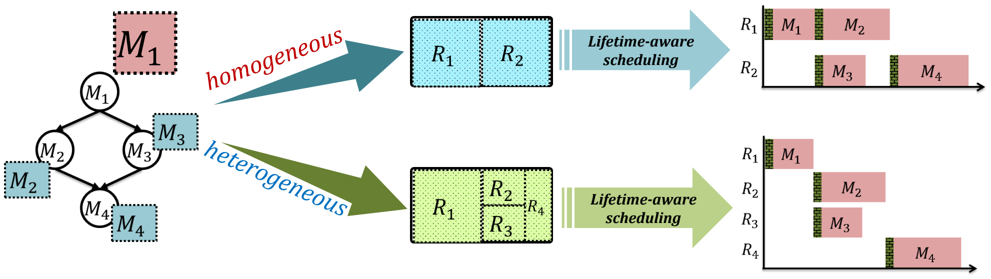


Figure 11. Reliability-aware HW/HW partitioning [110].

A few papers such as [112–116], have addressed reconfigurable processors in a system with different criticality tasks to improve timing reliability. In [112,113], Santos, et al. have proposed a new efficient scrubbing mechanism to increase the system's reliability by considering the criticality and timing of the hardware task execution. Hence, scrubbing mechanism takes advantages of FPGA reconfiguration, and verify the reconfiguration periodically to tolerate faults like Single Event Upset (SEU) in the Static-RAM (SRAM) [112,117]. Researchers in [112] have proposed a static heuristic to schedule the tasks based on their criticality level, running on reconfigurable embedded systems to maximize each task's reliability. In addition, they have presented a dynamic scrubbing mechanism in [113] to have high reliability by using the windows and fixed priority scheduling. They also consider the reliability as well as the criticality level for the tasks. They have claimed that they significantly reduce the amount of memory required to store the scrubbing schedule. Ref. [114] have presented an efficient resource management mechanism and then architecture to provide fault tolerance in the context of a time-triggered NoC-based mixed-criticality system that invokes reconfiguration. Their method establishes the fault-recovery and efficient resource utilization in Mixed-Criticality Networks-on-Chip (MCNoCs) by monitoring the resource requests and reconfiguring them based on a recovery strategy. Besides, ref. [115] have proposed a reliability driven scheduling approach for mixed-criticality tasks by handling periodic, aperiodic, and sporadic tasks on FPGAs against hardware trojan horse attacks. In this approach, redundancy is used to increase reliability on task criticality level, offline, and then attempt to prevent faulty data propagation in the run-time phase. Ref. [116] have detected an error by using the Secure Hash Algorithm and corrected them by using parity based two-dimensional erasure code, while the performance is reduced, which consists of time error detection and correction. This method has taken the execution period and criticality into account to correct faulty data.

8. Upcoming Trends and Open Challenges

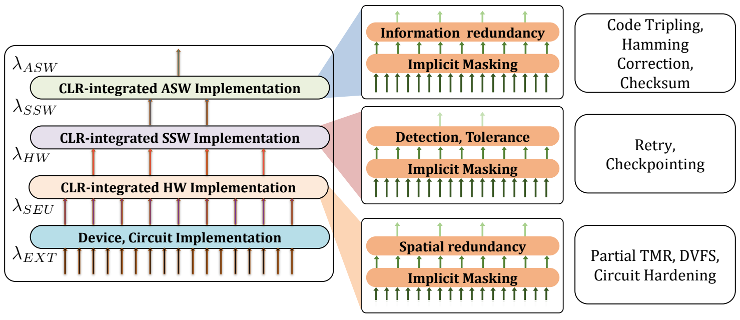
In this section, we briefly address the upcoming trends and challenges relevant to reliability-aware resource management in multi/many-core systems.

- Cross-layer Reliability: Most of the research articles discussed in this survey employ/select different redundancy-based methods to improve the system's reliability. Similarly, there is an increasing trend of using multiple layers of the system stack in the design for reliability [73,118]. This is unlike the traditional approach of mitigating each fault-mechanism at the hardware layer and providing a fault-free abstraction to the other layers. Although this phenomenon-based approach makes the design process simpler for the non-hardware layers, the high cost of hardware-based fault-mitigation can make this approach infeasible for resource-constrained systems. In contrast, the cross-layer approach involves multiple layers sharing the fault-mitigation activities during run-time [119]. Similarly, various methods of leveraging at cross-layer reliability at design-time have been proposed [50].
  One of the major advantages of the cross-layer approach is the inherent suitability for application-specific optimizations. Since the overheads of fault-tolerance varies with the type of redundancy being used, application-specific tolerances to degradation in some form of reliability can be used to improve other reliability metrics.

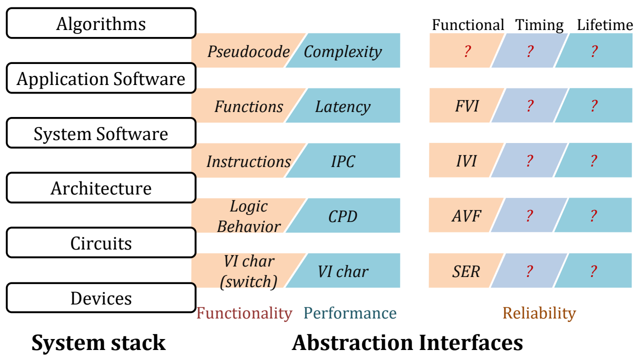
Further, with the cross-layer approach, the implicit masking of multiple layers can be used to provide low-cost fault-tolerance [120]. However, the joint optimization across multiple layers increases the design space considerably. Recently there have been multiple works that try to provide efficient DSE for cross-layer reliability for various system-level design tasks such as task mapping [37], hardware-hardware partitioning [121], run-time adaptation [40], hardware design [72] etc. However, most of the works assume rather simplistic reliability models such as the one shown in Figure 12a where each layer is limited to a specific type of redundancy [37,40,121]. A more holistic approach to the design of cross layer reliability is necessary for more realistic reliability models that integrate multiple reliability methods at each layer. As shown in Figure 12b having reliability interfaces, similar to those used for functionality and performance, can enable far better DSE than current state-of-the-art works. An interface for functional reliability was proposed by [80] that used Architectural Vulnerability Factor (AVF), Instruction Vulnerability Index (IVI) and Function Vulnerability Index (FVI) for characterising different implementations of an embedded processor, instruction set and function libraries, respectively. However, similar interfaces for timing and lifetime reliability need to be developed for designing efficient cross-layer reliability.

- Self-Aware System Design: In general, design-time approaches are applied to optimize resource usage and guarantee the reliability in the worst-case scenarios. However, due to the various run-time behaviors of applications and fault occurrence, we cannot efficiently manage the reliability, especially lifetime reliability and system utilization. Therefore, run-time system monitoring and optimization are essential to control and have a reliable operation of applications, especially mixed-criticality applications, and efficient resource management of multi/many-core platforms [122]. IPF paradigm is recently used to manage the system dynamically, according to the changes in system and workload [98,123]. This self-aware paradigm improves the reliability and resource utilization by combining different techniques in different hierarchical layers. As a result, the online optimization based on the current state and variations in applications and system by monitoring the hardware and software components, and using the IPF to conquer the complexity is essential, especially for safety-critical systems.
- Reliable Communication and Data Sharing: Safety and dependability are critical issues in designing the mixed-criticality systems on multi/many-core platforms, in which data are shared between concurrent execution of tasks with different criticality [124]. The strict control of data (critical and non-critical), communication, sharing, and storage in such systems for safety assurance, e.g., in medical devices, is crucial. Most state-of-the-art works have concentrated on the reliability management of tasks in processors of multi/many-core systems regardless of safe data sharing among communication and memories. As a result, safe mixed-criticality system design considering all system resources, like communications, and memory access, and processors are needed.

In addition to the trends discussed above many new approaches to computing have emerged over the past few years. These emerging technologies have brought forward novel opportunities and challenges to reliability-aware resource management. For example, Approximate Computing (AxC) has emerged as a new computing paradigm that offers the promise of low-power and faster execution [125]. While AxC might hold promise for improving timing and lifetime reliability, the deliberate introduction of computational inaccuracies requires careful design for dependable systems. Similarly, the introduction of post-CMOS transistors for next-generation computing requires a thorough reliability analysis of emerging devices. Finally, we are already witnessing AI-based resource management in multi-/many-core systems for both design-time optimization and run-time adaptation to varying operating conditions [40,126,127].


λ_{ASW}
CLR-integrated ASW Implementation
λ_{SSW}
CLR-integrated SSW Implementation
λ_{HW}
CLR-integrated HW Implementation
λ_{SEU}
Device, Circuit Implementation
λ_{EXT}
Information redundancy
Implicit Masking
Detection, Tolerance
Implicit Masking
Spatial redundancy
Implicit Masking
Code Tripling, Hamming Correction, Checksum
Retry, Checkpointing
Partial TMR, DVFS, Circuit Hardening

(**a**)



(**b**)

Figure 12. Cross-layer design approach to reliability. (**a**) Redundancy methods across layers. (**b**) Interfaces for cross-layer design approach.

9. Conclusions

With increasing susceptibility of modern electronic systems to physical faults, reliability-aware resource management is a topical research problem. While technology scaling and architectural innovations such as 3D integration allows us to integrate more and more cores in a system, extracting reliable performance from increasingly unreliable semiconductor heightens the need for resource management in multi-/many-core systems. To this end, this article presents our perspective on the related research. To begin, we have provided a detailed overview of the various phenomena that have contributed to the growing need for reliability in modern electronic systems. Then, we provided a taxonomy along with a detailed background of the various aspects of reliability improvements that are explored in related research works. Specifically we looked at the different types of reliability—lifetime, timing and functional (the different fault models and the system model) application and architecture. We formulated a generic problem statement for reliability-aware resource management that lets us determine the scope and classify the methods adopted in each of the related works. A survey of the related works is then presented, categorized under the type of DSE approach—design/compile-time, run-time and hybrid–adopted for each type of reliability. We have also presented a brief survey related research works targeted for FPGA-based systems. The presented survey focuses on the application-specific reliability, mixed-criticality awareness and hardware resource heterogeneity. In the end, we have provided a brief discussion on the upcoming trends in reliability-aware resource management and the challenges therein, to encourage further research in this topic.

Author Contributions: Conceptualization, A.K.; writing—original draft preparation, S.S.S. and B.R.; writing—review and editing, S.S.S., B.R. and A.K.; supervision, A.K.; project administration, A.K.; funding acquisition, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence Center for Advancing Electronics Dresden (CFAED) at the Technische Universität Dresden.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACO	Ant Colony Optimization
AVF	Architectural Vulnerability Factor
AxC	Approximate Computing
BTI	Bias Temperature Instability
DAG	Directed Acyclic Graph
DEC	Double-bit-Error-Correcting
DED	Double-bit-Error-Detecting
DMR	Dual Modular Redundancy
DPR	Dynamic Partial Reconfiguration
DRAM	Dynamic Random Access Memory
DSE	Design Space Exploration
DVFS	Dynamic Voltage and Frequncy Scaling
ECC	Error Checking and Correcting
EM	Electromigration
FPGA	Field Programmable Gate Array
FVI	Function Vulnerability Index
HCI	Hot Carrier Injection
ICs	integrated circuits
ILP	Instruction Level Parallelism
IoT	Internet of Thing
IPF	Information Processing Factory
IVI	Instruction Vulnerability Index
MAB	Multi-Armed Bandit
MCS	Monte-Carlo Simulations
MILP	Mixed Integer Linear Programming
MOEA	Multi-Objective Evolutionary Algorithms
MPSoC	Multi-Processor System-on-Chip
MTBF	Mean Time between Failures
MTTC	Mean Time To Crash
MTTF	Mean Time To Failure
NBTI	Negative Bias Temperature Instability
NoC	Network-on-Chip
PE	Processing Element
PFH	Probability-of-Failure-per-Hour
PRR	Partially Reconfigurable Region
QoS	Quality of Service
RL	Reinforcement Learning
RMT	Redundant Multi-Threading
SA	Simulated Annealing
SDFG	Synchronous Data Flow Graph
SEC	Single-bit-Error-Correcting
SER	Soft Error Rate
SEU	Single Event Upset
SRAM	Static Random Access Memory
TDDB	Time Dependent Dielectric Breakdown
TED	Triple-bit-Error-Detecting
TMR	Triple Modular Redundancy
WCET	Worst-case Execution Time

References

1. Coombs, A.W.M. The Making of Colossus. *Ann. Hist. Comput.* **1983**, *5*, 253–259. [CrossRef]
2. Dennard, R.H.; Gaensslen, F.H.; Rideout, V.L.; Bassous, E.; LeBlanc, A.R. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE J. Solid State Circuits* **1974**, *9*, 256–268. [CrossRef]
3. Patterson, D.A.; Hennessy, J.L. *Computer Organization and Design ARM Edition: The Hardware Software Interface*; Morgan Kaufmann: Burlington, MA, USA, 2016.
4. Rupp, K. 42 Years of Microprocessor Trend Data. Available online: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/ (accessed on 12 December 2020).
5. ARM. big. LITTLE Technology: The Future of Mobile. Available online: https://img.hexus.net/v2/press_releases/arm/big.LITTLE.Whitepaper.pdf (accessed on 12 December 2020).
6. Chen, T.; Raghavan, R.; Dale, J.N.; Iwata, E. Cell Broadband Engine Architecture and its first implementation–A performance view. *IBM J. Res. Dev.* **2007**, *51*, 559–572. [CrossRef]
7. Borkar, S. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* **2005**, *25*, 10–16. [CrossRef]
8. Shivakumar, P.; Kistler, M.; Keckler, S.W.; Burger, D.; Alvisi, L. Modeling the effect of technology trends on the soft error rate of combinational logic. In Proceedings of the International Conference on Dependable Systems and Networks, Washington, DC, USA, 23–26 June 2002; pp. 389–398. [CrossRef]
9. Nightingale, E.B.; Douceur, J.R.; Orgovan, V. Cycles, Cells and Platters: An Empirical Analysisof Hardware Failures on a Million Consumer PCs. In Proceedings of the Sixth Conference on Computer Systems, Salzburg, Austria, 10–13 April 2011; ACM: New York, NY, USA, 2011; pp. 343–356. [CrossRef]
10. Liu, J.W.S. *Real-Time Systems*; Prentice Hall: Upper Saddle River, NJ, USA, 2000.
11. Halang, W.A.; Gumzej, R.; Colnaric, M.; Druzovec, M. Measuring the performance of real-time systems. *Real Time Syst.* **2000**, *18*, 59–68. [CrossRef]
12. Das, A.K.; Kumar, A.; Veeravalli, B.; Catthoor, F. *Reliable and Energy Efficient Streaming Multiprocessor Systems*; Springer: Berlin/Heidelberg, Germany, 2018.
13. Avizienis, A.; Laprie, J.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* **2004**, *1*, 11–33. [CrossRef]
14. May, T.C.; Woods, M.H. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. Electron Devices* **1979**, *26*, 2–9. [CrossRef]
15. Ziegler, J.F.; Lanford, W.A. Effect of Cosmic Rays on Computer Memories. *Science* **1979**, *206*, 776–788. [CrossRef]
16. Keane, J.; Kim, C.H. An odomoeter for CPUs. *IEEE Spectr.* **2011**, *48*, 28–33. [CrossRef]
17. Zhang, J.F.; Eccleston, W. Positive bias temperature instability in MOSFETs. *IEEE Trans. Electron Devices* **1998**, *45*, 116–124. [CrossRef]
18. Takeda, E.; Suzuki, N.; Hagiwara, T. Device performance degradation to hot-carrier injection at energies below the Si-SiO$_2$energy barrier. In Proceedings of the 1983 International Electron Devices Meeting, Washington, DC, USA, 5–7 December 1983; pp. 396–399. [CrossRef]
19. Black, J.R. Electromigration: A brief survey and some recent results. *IEEE Trans. Electron Devices* **1969**, *16*, 338–347. [CrossRef]
20. Benini, L.; De Micheli, G. Networks on chips: A new SoC paradigm. *Computer* **2002**, *35*, 70–78. [CrossRef]
21. Radetzki, M.; Feng, C.; Zhao, X.; Jantsch, A. Methods for Fault Tolerance in Networks-on-chip. *ACM Comput. Surv.* **2013**, *46*, 8:1–8:38. [CrossRef]
22. Postman, J.; Chiang, P. A Survey Addressing On-Chip Interconnect: Energy and Reliability Considerations. Availabe online: https://www.hindawi.com/journals/isrn/2012/916259/ (accessed on 12 December 2020).
23. Hamming, R.W. Error detecting and error correcting codes. *Bell Syst. Tech. J.* **1950**, *29*, 147–160. [CrossRef]
24. Hsiao, M.Y. A Class of Optimal Minimum Odd-weight-column SEC-DED Codes. *IBM J. Res. Dev.* **1970**, *14*, 395–401. [CrossRef]
25. Das, A.; Kumar, A. Fault-aware task re-mapping for throughput constrained multimedia applications on NoC-based MPSoCs. In Proceedings of the IEEE International Symposium on Rapid System Prototyping (RSP), Tampere, Finland, 11–12 October 2012.
26. Das, A.; Kumar, A.; Veeravalli, B. Aging-aware hardware-software task partitioning for reliable reconfigurable multiprocessor systems. In Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), Montreal, QC, Canada, 29 September–4 October 2013; pp. 1–10. [CrossRef]
27. Medina, R.; Borde, E.; Pautet, L. Availability enhancement and analysis for mixed-criticality systems on multi-core. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018.
28. Ranjbar, B.; Safaei, B.; Ejlali, A.; Kumar, A. FANTOM: Fault Tolerant Task-Drop Aware Scheduling for Mixed-Criticality Systems. *IEEE Access* **2020**, *8*, 187232–187248. [CrossRef]
29. Johnson, L.A. DO-178B, Software considerations in airborne systems and equipment certification. *Crosstalk Oct.* **1998**, *199*, 11–20.

30. Hartman, A.S.; Thomas, D.E.; Meyer, B.H. A Case for Lifetime-Aware Task Mapping in Embedded Chip Multiprocessors. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES), CODES/ISSS '10, Scottsdale, AZ, USA, 24–29 October 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 145–154. [CrossRef]
31. Meyer, B.H.; Hartman, A.S.; Thomas, D.E. Cost-effective slack allocation for lifetime improvement in NoC-based MPSoCs. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 8–12 March 2010; pp. 1596–1601. [CrossRef]
32. Meyer, B.H.; Hartman, A.S.; Thomas, D.E. Cost-Effective Lifetime and Yield Optimization for NoC-Based MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* **2014**, *19*. [CrossRef]
33. Ma, C.; Mahajan, A.; Meyer, B.H. Multi-armed bandits for efficient lifetime estimation in MPSoC design. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1540–1545. [CrossRef]
34. Xiang, Y.; Chantem, T.; Dick, R.P.; Hu, X.S.; Shang, L. System-level reliability modeling for MPSoCs. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES), Scottsdale, AZ, USA, 24–29 October 2010; pp. 297–306.
35. 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 the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; European Design and Automation Association: Leuven, Belgium, 2014.
36. Das, A.; Kumar, A.; Veeravalli, B. Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6. [CrossRef]
37. Sahoo, S.S.; Veeravalli, B.; Kumar, A. CL(R)Early: An Early-stage DSE Methodology for Cross-Layer Reliability-aware Heterogeneous Embedded Systems. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 19–22 July 2020; pp. 1–6. [CrossRef]
38. Kakoee, M.R.; Bertacco, V.; Benini, L. ReliNoC: A reliable network for priority-based on-chip communication. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Grenoble, France, 14–18 March 2011; pp. 1–6. [CrossRef]
39. Sahoo, S.S.; Kumar, A.; Veeravalli, B. Design and evaluation of reliability-oriented task re-mapping in MPSoCs using time-series analysis of intermittent faults. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 798–803.
40. Sahoo, S.S.; Veeravalli, B.; Kumar, A. A Hybrid Agent-Based Design Methodology for Dynamic Cross-Layer Reliability in Heterogeneous Embedded Systems. In Proceedings of the Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
41. Hartman, A.S.; Thomas, D.E. Lifetime Improvement through Runtime Wear-Based Task Mapping. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES), CODES + ISSS '12, Tampere, Finland, 7–12 October 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 13–22. [CrossRef]
42. Duque, L.A.R.; Diaz, J.M.M.; Yang, C. Improving MPSoC reliability through adapting runtime task schedule based on time-correlated fault behavior. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2015; pp. 818–823.
43. Rathore, V.; Chaturvedi, V.; Singh, A.K.; Srikanthan, T.; Shafique, M. Life Guard: A Reinforcement Learning-Based Task Mapping Strategy for Performance-Centric Aging Management. In Proceedings of the Design Automation Conference (DAC), Las Vegas, NA, USA, 2–6 June 2019; pp. 1–6.
44. Venkataraman, S.; Santos, R.; Kumar, A.; Kuijsten, J. Hardware task migration module for improved fault tolerance and predictability. In Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Samos, Greece, 19–23 July 2015; pp. 197–202. [CrossRef]
45. Wang, L.; Lv, P.; Liu, L.; Han, J.; Leung, H.; Wang, X.; Yin, S.; Wei, S.; Mak, T. A Lifetime Reliability-Constrained Runtime Mapping for Throughput Optimization in Many-Core Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2019**, *38*, 1771–1784. [CrossRef]
46. Haghbayan, M.; Miele, A.; Rahmani, A.M.; Liljeberg, P.; Tenhunen, H. A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 854–857.
47. Haghbayan, M.; Miele, A.; Rahmani, A.M.; Liljeberg, P.; Tenhunen, H. Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era. *IEEE Trans. Comput.* **2017**, *66*, 1599–1612. [CrossRef]
48. Rathore, V.; Chaturvedi, V.; Singh, A.K.; Srikanthan, T.; Rohith, R.; Lam, S.; Shafique, M. HiMap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 991–996. [CrossRef]
49. Raparti, V.Y.; Kapadia, N.; Pasricha, S. ARTEMIS: An Aging-Aware Runtime Application Mapping Framework for 3D NoC-Based Chip Multiprocessors. *IEEE Trans. Multi-Scale Comput. Syst.* **2017**, *3*, 72–85. [CrossRef]
50. Bauer, L.; Henkel, J.; Herkersdorf, A.; Kochte, M.A.; Kühn, J.M.; Rosenstiel, W.; Schweizer, T.; Wallentowitz, S.; Wenzel, V.; Wild, T.; et al. Adaptive multi-layer techniques for increased system dependability. *It-Inf. Technol.* **2015**, *57*, 149–158. [CrossRef]

51. Das, A.; Kumar, A.; Veeravalli, B. Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; pp. 689–694. [CrossRef]
52. Das, A.; Kumar, A.; Veeravalli, B. Reliability and Energy-Aware Mapping and Scheduling of Multimedia Applications on Multiprocessor Systems. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 869–884. [CrossRef]
53. Das, A.; Kumar, A.; Veeravalli, B. Communication and migration energy aware design space exploration for multicore systems with intermittent faults. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; pp. 1631–1636. [CrossRef]
54. Bolchini, C.; Carminati, M.; Miele, A.; Das, A.; Kumar, A.; Veeravalli, B. Run-time mapping for reliable many-cores based on energy/performance trade-offs. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), New York, NY, USA, 2–4 October 2013; pp. 58–64. [CrossRef]
55. Namazi, A.; Safari, S.; Mohammadi, S.; Abdollahi, M. SORT: Semi Online Reliable Task Mapping for Embedded Multi-Core Systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.* **2019**, *4*. [CrossRef]
56. Kriebel, F.; Rehman, S.; Shafique, M.; Henkel, J. AgeOpt-RMT: Compiler-Driven Variation-Aware Aging Optimization for Redundant Multithreading. In Proceedings of the Design Automation Conference (DAC), DAC '16, Austin, TX, USA, 5–9 June 2016; Association for Computing Machinery: New York, NY, USA, 2016. [CrossRef]
57. Nahar, B.; Meyer, B.H. RotR: Rotational redundant task mapping for fail-operational MPSoCs. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), Amherst, MA, USA, 12–14 October 2015; pp. 21–28. [CrossRef]
58. Axer, P.; Sebastian, M.; Ernst, R. Reliability Analysis for MPSoCs with Mixed-Critical, Hard Real-Time Constraints. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES), Taipei Taiwan, 9–14 October 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 149–158. [CrossRef]
59. Pathan, R.M. Real-time scheduling algorithm for safety-critical systems on faulty multicore environments. *Real Time Syst.* **2017**, *53*, 45–81. [CrossRef]
60. Safari, S.; Hessabi, S.; Ershadi, G. LESS-MICS: A Low Energy Standby-Sparing Scheme for Mixed-Criticality Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 4601–4610. [CrossRef]
61. Saraswat, P.K.; Pop, P.; Madsen, J. Task Migration for Fault-Tolerance in Mixed-Criticality Embedded Systems. *SIGBED Rev.* **2009**, *6*. [CrossRef]
62. Liu, G.; Lu, Y.; Wang, S. An Efficient Fault Recovery Algorithm in Multiprocessor Mixed-Criticality Systems. In Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC)/ IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Zhangjiajie, China, 13–15 November 2013; pp. 2006–2013.
63. Ranjbar, B.; Nguyen, T.D.A.; Ejlali, A.; Kumar, A. Online Peak Power and Maximum Temperature Management in Multi-core Mixed-Criticality Embedded Systems. In Proceedings of the Euromicro Conference on Digital System Design (DSD), Chalkidiki, Greece, 28–30 August 2019; pp. 546–553.
64. Iacovelli, S.; Kirner, R.; Menon, C. ATMP: An Adaptive Tolerance-based Mixed-criticality Protocol for Multi-core Systems. In Proceedings of the International Symposium on Industrial Embedded Systems (SIES), Graz, Austria, 6–8 June 2018; pp. 1–9.
65. Al-bayati, Z.; Meyer, B.H.; Zeng, H. Fault-tolerant scheduling of multicore mixed-criticality systems under permanent failures. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Storrs, CT, USA, 19–20 September 2016; pp. 57–62.
66. O'Connor, P.P.; Kleyner, A. *Practical Reliability Engineering*, 5th ed.; Wiley Publishing: Hoboken, NJ, USA, 2012.
67. Esmaeilzadeh, H.; Blem, E.; Amant, R.S.; Sankaralingam, K.; Burger, D. Dark silicon and the end of multicore scaling. In Proceedings of the 2011 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; Association for Computing Machinery: New York, NY, USA, 2011.
68. Buttazzo, G.C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011; Volume 24.
69. Ranjbar, B.; Nguyen, T.D.A.; Ejlali, A.; Kumar, A. Power-Aware Run-Time Scheduler for Mixed-Criticality Systems on Multi-Core Platform. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**. [CrossRef]
70. Sahoo, S.S.; Veeravalli, B.; Kumar, A. Markov Chain-based Modeling and Analysis of Checkpointing with Rollback Recovery for Efficient DSE in Soft Real-time Systems. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Frascati, Italy, 19–21 October 2020; pp. 1–6. [CrossRef]
71. Manolache, S.; Eles, P.; Peng, Z. Task Mapping and Priority Assignment for Soft Real-Time Applications under Deadline Miss Ratio Constraints. *ACM Trans. Embed. Comput. Syst.* **2008**, *7*. [CrossRef]
72. Cheng, E.; Mirkhani, S.; Szafaryn, L.G.; Cher, C.Y.; Cho, H.; Skadron, K.; Stan, M.R.; Lilja, K.; Abraham, J.A.; Bose, P.; et al. CLEAR: Cross-Layer Exploration for Architecting Resilience - Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores. In Proceedings of the Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016; Association for Computing Machinery: New York, NY, USA, 2016. [CrossRef]
73. Henkel, J.; Bauer, L.; Zhang, H.; Rehman, S.; Shafique, M. Multi-Layer Dependability: From Microarchitecture to Application Level. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 1–6. [CrossRef]

74. Frantz, A.P.; Cassel, M.; Kastensmidt, F.L.; Cota, E.; Carro, L. Crosstalk- and SEU-Aware Networks on Chips. *IEEE Des. Test* **2007**, *24*, 340–350. [CrossRef]
75. Lehtonen, T.; Liljeberg, P.; Plosila, J. Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. Available online: https://www.hindawi.com/journals/vlsi/2007/094676/ (accessed on 12 December 2020).
76. Vitkovskiy, A.; Soteriou, V.; Nicopoulos, C. A fine-grained link-level fault-tolerant mechanism for networks-on-chip. In Proceedings of the IEEE International Conference on Computer Design, Amsterdam, The Netherlands, 3–6 October 2010; pp. 447–454. [CrossRef]
77. Wells, P.M.; Chakraborty, K.; Sohi, G.S. Adapting to Intermittent Faults in Multicore Systems. *SIGOPS Oper. Syst. Rev.* **2008**, *42*, 255–264. [CrossRef]
78. Lehtonen, T.; Wolpert, D.; Liljeberg, P.; Plosila, J.; Ampadu, P. Self-Adaptive System for Addressing Permanent Errors in On-Chip Interconnects. *IEEE Trans. Very Large Scale Integr. Syst.* **2010**, *18*, 527–540. [CrossRef]
79. Yu, Q.; Ampadu, P. Transient and Permanent Error Co-management Method for Reliable Networks-on-Chip. In Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip, Grenoble, France, 3–6 May 2010; pp. 145–154. [CrossRef]
80. Rehman, S.; Chen, K.; Kriebel, F.; Toma, A.; Shafique, M.; Chen, J.; Henkel, J. Cross-Layer Software Dependability on Unreliable Hardware. *IEEE Trans. Comput.* **2016**, *65*, 80–94. [CrossRef]
81. Weichslgartner, A.; Wildermann, S.; Gangadharan, D.; Glaß, M.; Teich, J. A Design-Time/Run-Time Application Mapping Methodology for Predictable Execution Time in MPSoCs. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*. [CrossRef]
82. Pourmohseni, B.; Wildermann, S.; Glaß, M.; Teich, J. Hard Real-Time Application Mapping Reconfiguration for NoC-Based Many-Core Systems. *Real Time Syst.* **2019**, *55*, 433–469. [CrossRef]
83. Safari, S.; Ansari, M.; Ershadi, G.; Hessabi, S. On the Scheduling of Energy-Aware Fault-Tolerant Mixed-Criticality Multicore Systems with Service Guarantee Exploration. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2338–2354. [CrossRef]
84. Rambo, E.A.; Ernst, R. Replica-Aware Co-Scheduling for Mixed-Criticality. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*; Leibniz International Proceedings in Informatics (LIPIcs); Bertogna, M., Ed.; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2017; Volume 76, pp. 20:1–20:20. [CrossRef]
85. Bolchini, C.; Miele, A. Reliability-Driven System-Level Synthesis for Mixed-Critical Embedded Systems. *IEEE Trans. Comput.* **2013**, *62*, 2489–2502. [CrossRef]
86. Kang, S.; Yang, H.; Kim, S.; Bacivarov, I.; Ha, S.; Thiele, L. Reliability-aware mapping optimization of multi-core systems with mixed-criticality. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–4.
87. Kang, S.h.; Yang, H.; Kim, S.; Bacivarov, I.; Ha, S.; Thiele, L. Static Mapping of Mixed-Critical Applications for Fault-Tolerant MPSoCs. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 1–6. [CrossRef]
88. Choi, J.; Yang, H.; Ha, S. Optimization of Fault-Tolerant Mixed-Criticality Multi-Core Systems with Enhanced WCRT Analysis. *ACM Trans. Des. Autom. Electron. Syst.* **2018**, *24*. [CrossRef]
89. Jiang, W.; Hu, H.; Zhan, J.; Jiang, K. Work-in-Progress: Design of Security-Critical Distributed Real-Time Applications with Fault-Tolerant Constraint. In Proceedings of the International Conference on Embedded Software (EMSOFT), Torino, Italy, 30 September–5 October 2018; pp. 1–2.
90. Zeng, L.; Huang, P.; Thiele, L. Towards the Design of Fault-Tolerant Mixed-Criticality Systems on Multicores. In Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), Pittsburgh, PA, USA, 2–7 October 2016; Association for Computing Machinery: New York, NY, USA, 2016. [CrossRef]
91. Caplan, J.; Al-bayati, Z.; Zeng, H.; Meyer, B.H. Mapping and Scheduling Mixed-Criticality Systems with On-Demand Redundancy. *IEEE Trans. Comput.* **2018**, *67*, 582–588. [CrossRef]
92. Saraswat, P.K.; Pop, P.; Madsen, J. Task Mapping and Bandwidth Reservation for Mixed Hard/Soft Fault-Tolerant Embedded Systems. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Stockholm, Sweden, 12–15 April 2010; pp. 89–98.
93. Bagheri, M.; Jervan, G. Fault-Tolerant Scheduling of Mixed-Critical Applications on Multi-processor Platforms. In Proceedings of the IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Milano, Italy, 25–29 August 2014; pp. 25–32.
94. Kajmakovic, A.; Diwold, K.; Kajtazovic, N.; Zupanc, R.; Macher, G. Flexible Soft Error Mitigation Strategy for Memories in Mixed-Critical Systems. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 28–31 October 2019; pp. 440–445.
95. Liu, Y.; Xie, G.; Tang, Y.; Li, R. Improving Real-Time Performance Under Reliability Requirement Assurance in Automotive Electronic Systems. *IEEE Access* **2019**, *7*, 140875–140888. [CrossRef]
96. Thekkilakattil, A.; Dobrin, R.; Punnekkat, S. Mixed criticality scheduling in fault-tolerant distributed real-time systems. In Proceedings of the International Conference on Embedded Systems (ICES), Coimbatore, India, 3–5 July 2014; pp. 92–97.
97. Koc, H.; Karanam, V.K.; Sonnier, M. Latency Constrained Task Mapping to Improve Reliability of High Critical Tasks in Mixed Criticality Systems. In Proceedings of the IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), British Columbia, VA, Canada, 17–19 October 2019; pp. 0320–0324.

98. Rambo, E.A.; Donyanavard, B.; Seo, M.; Maurer, F.; Kadeed, T.M.; De Melo, C.B.; Maity, B.; Surhonne, A.; Herkersdorf, A.; Kurdahi, F.; et al. The Self-Aware Information Processing Factory Paradigm for Mixed-Critical Multiprocessing. *IEEE Trans. Emerg. Top. Comput.* **2020**. [CrossRef]
99. LogiCORE IP. Soft Error Mitigation Controller v3. 4. Available online: https://www.xilinx.com/support/answers/54733.html (accessed on 12 December 2020).
100. Santos, R.; Venkataraman, S.; Kumar, A. Scrubbing Mechanism for Heterogeneous Applications in Reconfigurable Devices. *ACM Trans. Des. Autom. Electron. Syst.* **2017**, *22*. [CrossRef]
101. Koch, D.; Haubelt, C.; Teich, J. Efficient Hardware Checkpointing: Concepts, Overhead Analysis, and Implementation. In Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA), Monterey, CA, USA, 18–20 February 2007.
102. Lee, G.; Agiakatsikas, D.; Wu, T.; Cetin, E.; Diessel, O. TLegUp: A TMR code generation tool for SRAM-based FPGA applications using HLS. In Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, 30 April–2 May 2017; pp. 129–132.
103. Srinivasan, S.; Krishnan, R.; Mangalagiri, P.; Xie, Y.; Narayanan, V.; Irwin, M.J.; Sarpatwari, K. Toward Increasing FPGA Lifetime. *IEEE Trans. Dependable Secur. Comput.* **2008**, *5*, 115–127. [CrossRef]
104. Stott, E.; Cheung, P.Y.K. Improving FPGA Reliability with Wear-Levelling. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Chania, Crete, Greece, 5–7 September 2011; pp. 323–328. [CrossRef]
105. Angermeier, J.; Ziener, D.; Glaß, M.; Teich, J. Stress-Aware Module Placement on Reconfigurable Devices. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Chania, Crete, Greece, 5–7 September 2011; pp. 277–281. [CrossRef]
106. Zhang, H.; Bauer, L.; Kochte, M.A.; Schneider, E.; Braun, C.; Imhof, M.E.; Wunderlich, H.; Henkel, J. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In Proceedings of the IEEE International Test Conference (ITC), Anaheim, CA, USA, 6–13 September 2013; pp. 1–10. [CrossRef]
107. Zhang, H.; Kochte, M.A.; Schneider, E.; Bauer, L.; Wunderlich, H.; Henkel, J. STRAP: Stress-aware placement for aging mitigation in runtime reconfigurable architectures. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 2–6 November 2015; pp. 38–45. [CrossRef]
108. Ghaderi, Z.; Bozorgzadeh, E. Aging-aware high-level physical planning for reconfigurable systems. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, 25–28 January 2016; pp. 631–636. [CrossRef]
109. Dumitriu, V.; Kirischian, L.; Kirischian, V. Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-Organized Dynamic Partially Reconfigurable Systems. *IEEE Trans. Comput.* **2016**, *65*, 2835–2847. [CrossRef]
110. Sahoo, S.S.; Nguyen, T.D.A.; Veeravalli, B.; Kumar, A. Lifetime-aware design methodology for dynamic partially reconfigurable systems. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju Island , Korea, 22–25 January 2018; pp. 393–398.
111. Sahoo, S.; Nguyen, T.; Veeravalli, B.; Kumar, A. Multi-objective design space exploration for system partitioning of FPGA-based Dynamic Partially Reconfigurable Systems. *Integration* **2019**, *67*, 95–107. [CrossRef]
112. Santos, R.; Venkataraman, S.; Das, A.; Kumar, A. Criticality-aware scrubbing mechanism for SRAM-based FPGAs. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–8.
113. Santos, R.; Venkataraman, S.; Kumar, A. Dynamically adaptive scrubbing mechanism for improved reliability in reconfigurable embedded systems. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 7–11 June 2015; pp. 1–6.
114. Ahmadian, H.; Nekouei, F.; Obermaisser, R. Fault recovery and adaptation in time-triggered Networks-on-Chips for mixed-criticality systems. In Proceedings of the International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Madrid, Spain, 12–17 July 2017; pp. 1–8.
115. Guha, K.; Majumder, A.; Saha, D.; Chakrabarti, A. Reliability Driven Mixed Critical Tasks Processing on FPGAs Against Hardware Trojan Attacks. In Proceedings of the Euromicro Conference on Digital System Design (DSD), Prague, Czech Republic, 29–31 August 2018; pp. 537–544.
116. Mandal, S.; Sarkar, S.; Ming, W.M.; Chattopadhyay, A.; Chakrabarti, A. Criticality Aware Soft Error Mitigation in the Configuration Memory of SRAM Based FPGA. In Proceedings of the International Conference on VLSI Design and International Conference on Embedded Systems (VLSID), Delhi, India, 5–9 January 2019; pp. 257–262.
117. Heiner, J.; Sellers, B.; Wirthlin, M.; Kalb, J. FPGA partial reconfiguration via configuration scrubbing. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Prague, Czech Republic, 31 August–2 September 2009; pp. 99–104. [CrossRef]
118. Sahoo, S.S.; Veeravalli, B.; Kumar, A. Cross-layer fault-tolerant design of real-time systems. In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Storrs, CT, USA, 19–20 September 2016; pp. 63–68. [CrossRef]
119. Carter, N.P.; Naeimi, H.; Gardner, D.S. Design techniques for cross-layer resilience. In Proceedings of the Design, Automation Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 8–12 March 2010; pp. 1023–1028. [CrossRef]

120. Santini, T.; Rech, P.; Sartor, A.; Corrêa, U.B.; Carro, L.; Wagner, F.R. Evaluation of Failures Masking Across the Software Stack. Available online: http://www.median-project.eu/wp-content/uploads/12_II-1_median2015.pdf (accessed on 12 December 2020).
121. Sahoo, S.S.; Nguyen, T.D.A.; Veeravalli, B.; Kumar, A. QoS-Aware Cross-Layer Reliability-Integrated FPGA-Based Dynamic Partially Reconfigurable System Partitioning. In Proceedings of the International Conference on Field-Programmable Technology (FPT), Naha, Japan, 11–15 December 2018; pp. 230–233.
122. Götzinger, M.; Rahmani, A.M.; Pongratz, M.; Liljeberg, P.; Jantsch, A.; Tenhunen, H. The Role of Self-Awareness and Hierarchical Agents in Resource Management for Many-Core Systems. In Proceedings of the IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), Hanoi, Vietnam, 12–14 September 2018; pp. 53–60. [CrossRef]
123. Rambo, E.A.; Kadeed, T.; Ernst, R.; Seo, M.; Kurdahi, F.; Donyanavard, B.; de Melo, C.B.; Maity, B.; Moazzemi, K.; Stewart, K.; et al. The Information Processing Factory: A Paradigm for Life Cycle Management of Dependable Systems. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES), New York, NY, USA, 13–18 October 2019; pp. 1–10.
124. Burns, A.; Davis, R.I. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* **2017**, *50*. [CrossRef]
125. Mittal, S. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* **2016**, *48*. [CrossRef]
126. Das, A.; Shafik, R.A.; Merrett, G.V.; Al-Hashimi, B.M.; Kumar, A.; Veeravalli, B. Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems. In Proceedings of the Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 1–6. [CrossRef]
127. Zhang, J.; Sadiqbatcha, S.; Gao, Y.; O'Dea, M.; Yu, N.; Tan, S.X.D. HAT-DRL: Hotspot-Aware Task Mapping for Lifetime Improvement of Multicore System Using Deep Reinforcement Learning. In Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD, Virtual Event, Iceland, 16–20 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 77–82.