Dependable Software Generation and Execution on Embedded Systems

Florian Kriebel, Kuan-Hsun Chen, Semeen Rehman, Jörg Henkel, Jian-Jia Chen, and Muhammad Shafique

1 Overview

An overview of the chapter structure and the connection of the different sections is illustrated in Fig. 1. Soft error mitigation techniques like [17, 29] have shown that the software layer can be employed for enhancing the dependability of computing systems. However, to effectively use them, their overhead (e.g., in terms of power and performance) has to be considered. This also includes the option of adapting to different output accuracy requirements and inherent resilience against faults of different applications, for which appropriate metrics considering information from multiple system layers are required. Therefore, we start with a short overview of reliability and resilience modeling and estimation approaches, which not only focus on the functional correctness (like application reliability and resilience) but also consider the timeliness, i.e., determining the change of the timing behavior according to the run-time dependability, and providing various timing guarantees for real-time systems. They are used to evaluate the results of different dependable

F. Kriebel (✉) · M. Shafique
Technische Universität Wien (TU Wien), ECS (E191-02), Institute of Computer Engineering, Wien, Austria
e-mail: florian.kriebel@tuwien.ac.at; muhammad.shafique@tuwien.ac.at

K.-H. Chen · J.-J. Chen
Technische Universität Dortmund, Lehrstuhl Informatik 12, Dortmund, Germany
e-mail: kuan-hsun.chen@tu-dortmund.de; jian-jia.chen@cs.uni-dortmund.de

S. Rehman
Technische Universität Wien (TU Wien), ICT (E384), Wien, Austria
e-mail: semeen.rehman@tuwien.ac.at

J. Henkel
Karlsruhe Institute of Technology, Karlsruhe, Germany
e-mail: henkel@kit.edu

© The Author(s) 2021

J. Henkel, N. Dutt (eds.), *Dependable Embedded Systems*, Embedded Systems,
https://doi.org/10.1007/978-3-030-52017-5_6

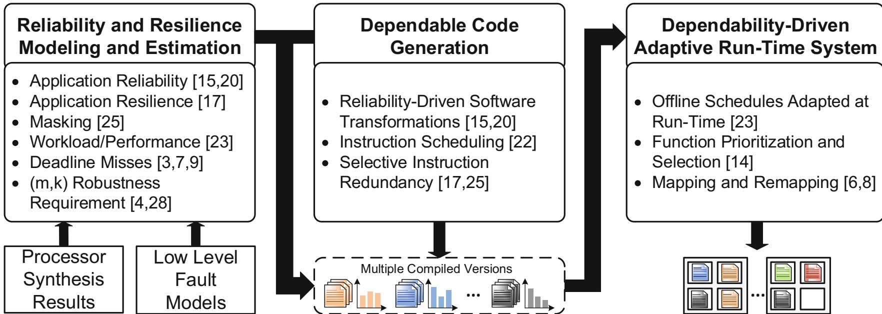


Fig. 1 Overview

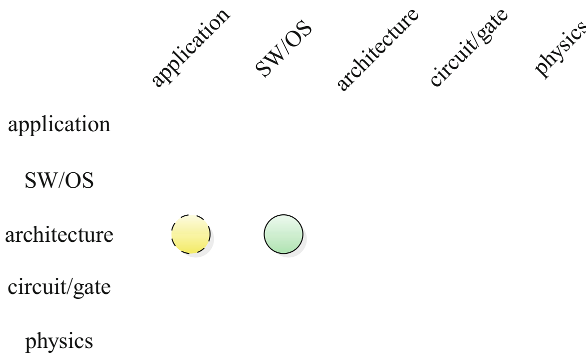


Fig. 2 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

code generation approaches, like *dependability-driven software transformations* and *selective instruction redundancy*. This enables generation of multiple compiled code versions of an application realizing different performance/energy vs. dependability trade-offs. The evaluation results and the different versions are then used by a *dependability-driven adaptive run-time system*. It considers *offline* and *online* optimizations, for instance, for selecting appropriate application versions and adapting to different workloads and conditions at run-time (like fault rate, aging, and process variation). Thereby, it finally enables a dependable execution of the applications on the target system.

As, however, not all systems are general-purpose, towards the end of the chapter an example design of a video processing system is included, which illustrates different approaches for application-specific dependability.

Embedding this chapter's content in the scope of this book and the overall projects [12, 14], the main contributions lie on the application, SW/OS, and architectural layers as illustrated in Fig. 2.

2 Dependability Modeling and Estimation

Modeling dependability at the software layer is a complex task as parameters and effects of different systems layers have to be taken into account. For an accurate yet fast evaluation of application dependability, information from lower system layers has to be considered, while abstracting it in a reasonable way to also allow for a fast estimation. For this purpose, different aspects have been separated into distinct metrics focusing on individual phenomena, as discussed below.

- The **Instruction Vulnerability Index (IVI)** [19, 25] focuses on the error probability of each instruction when being executed on different components/pipeline stages of a processor by analyzing their *spatial and temporal vulnerabilities*. This requires an analysis of *vulnerable bits* as well as *vulnerable time period*, i.e., the residence times of instructions in different components, while considering micro-architecture dependent information from the lower layers like the *area consumption* of different components and the *probability* that an error is observed at their output (see Fig. 1). The IVI of individual instructions can then be combined to estimate the vulnerability at higher granularity (e.g., Function Vulnerability Index—FVI). In this case, the susceptibility towards application failures and incorrect application outputs can be considered as well, for instance by classifying instructions into *critical* and *non-critical* ones, which is important if deviations in the application output can be tolerated.
- As not all errors occurring during the execution of an application become visible to the user due to *data flow and control flow masking*, the **Instruction Error Masking Index (IMI)** [31] provides probabilistic estimates whether the erroneous output of an instruction will be masked until the visible output of an application.
- The **Instruction Error Propagation Index (EPI)** [31] captures the effects of errors not being masked from the time of their generation until the final output of an application. It analyzes the propagation effects at instruction granularity and quantifies the impacts of the error propagation and how much it affects the final output of an application.
- Based on the information theory principles, the **Function Resilience** model [24] provides a probabilistic measure of the function's correctness (i.e., its output quality) in the presence of faults. In contrast to the *IVI/FVI*, it avoids exposing the application details by adopting a black-box modeling technique.
- The **Reliability-Timing Penalty (RTP)** [23] model jointly accounts for the *functional correctness* (i.e., generating the correct output) and the *timing correctness* (i.e., timely delivery of an output). In this work, we studied RTP as a linear combination of functional reliability and timing reliability, where the focus (functional or timing correctness) can be adjusted. However, it can also be devised through a non-linear model depending upon the design requirements of the target system.
- The **(m,k) robustness constraint** model [4, 35] quantifies the potential inherent safety margins of control tasks. In this work, several error-handling approaches

guarantee the minimal frequency of correctness over a static number of instances while satisfying the hard real-time constraints in the worst-case scenario.
- The **Deadline-Miss Probability** [3, 9, 34] provides a statistical argument for the probabilistic timing guarantees in soft real-time systems by assuming that after a deadline miss the system either discards the job missing its deadline or reboots itself. It is used to derive the **Deadline-Miss Rate** [7], which captures the frequency of deadline misses by considering the backlog of overrun tasks without the previous assumption of discarding jobs or rebooting the system.

A more detailed description of the different models as well as their corresponding system layers are presented in chapter "Reliable CPS Design for Unreliable Hardware Platforms".

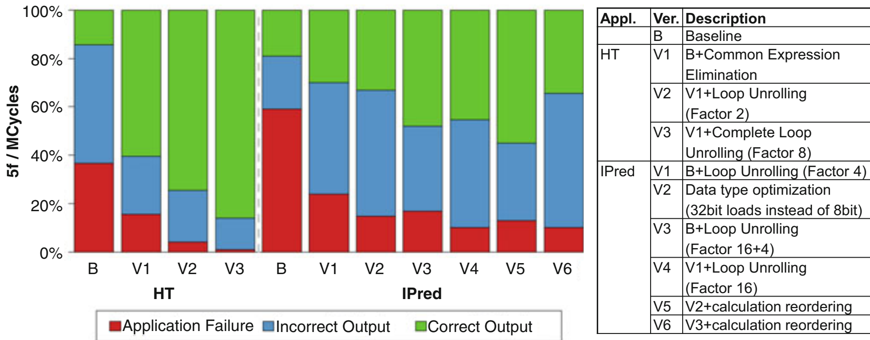
3 Dependability-Driven Compilation

Considering the models and therewith the main parameters affecting the dependability of a system, several mitigation techniques are developed, which target to improve the system dependability on the software layer. Three different approaches are discussed in the following.

3.1 Dependability-Driven Software Transformations

Software transformations like loop unrolling have mainly been motivated by and analyzed from the perspective of improving performance. Similarly, techniques for improving dependability at the software level have mainly focused on error detection and mitigation, e.g., by using redundant instruction executions. Therefore, the following dependability-driven compiler-based software transformations [19, 25] can be used to generate different application versions, which are identical in terms of their functionality but which provide different dependability-performance tradeoffs.

- *Dependability-Driven Data Type Optimization:* The idea is to implement the same functionality with different data types, targeting to reduce the number of memory load/store instructions (which are critical instructions due to their potential of causing application failures) and their predecessor instructions in the execution path. However, additional extraction/merging instructions for the data type optimization have to be taken care of when applying this transformation.
- *Dependability-Driven Loop Unrolling:* The goal is to find an unrolling factor (i.e., loop body replications), which minimizes the number of critical instructions/data (e.g., loop counters, branch instructions) that can lead to a significant deviation in the control flow causing application failures. This reduction, however, needs to be balanced, e.g., with the increase in the code size.



Appl.	Ver.	Description
	B	Baseline
HT	V1	B+Common Expression Elimination
	V2	V1+Loop Unrolling (Factor 2)
	V3	V1+Complete Loop Unrolling (Factor 8)
IPred	V1	B+Loop Unrolling (Factor 4)
	V2	Data type optimization (32bit loads instead of 8bit)
	V3	B+Loop Unrolling (Factor 16+4)
	V4	V1+Loop Unrolling (Factor 16)
	V5	V2+calculation reordering
	V6	V3+calculation reordering

Fig. 3 Fault injection results for two applications and the generated application versions (adapted from [25])

- *Reliability-Driven Common Expression Elimination and Operation Merging:* The idea of eliminating common expressions is to achieve performance improvement due to less instructions being executed and therefore less faults being able to affect an application execution. However, excessively applying this transformation might lead to register spilling or longer residence times of data in the registers. Therefore, it needs to be evaluated carefully whether eliminating a common expression leads to a vulnerability reduction or whether the redundancy implied by a re-execution provides a benefit.
- *Reliability-Driven Online Table Value Computation:* The goal of the online table value computation is to avoid long residence times of pre-computed tables in the memory, where the values can be affected by faults and can therefore affect a large set of computations. This needs to be traded off against the performance overhead (and therefore increased temporal vulnerability) of online value computation.

As the transformations listed above also imply certain side effects (e.g., increased code size, additional instructions), they need to be applied carefully. We evaluate the above techniques using an *instruction set simulator-based fault injection approach*, where faults can be injected in different processor components (e.g., register file, PC, ALU, etc.) considering their area. It supports injecting a single or multiple faults per experiment, where each fault can itself corrupt a single or multiple bits. The results for two example applications from the *MiBench* benchmark suite [13] are shown in Fig. 3. They illustrate the effectiveness of the proposed transformations, e.g., for the "HT" application by the reduction of the application failures and incorrect outputs generated when comparing the *Baseline* application version and *V3*.

Finally, the dependability-driven software transformations are not only useful as a standalone technique, but can also be combined with other error mitigation techniques. For example, by reducing the number of instructions accessing the

memory, they can help reduce the required checking instructions in [29], and thereby lead to a performance improvement.

3.2 Dependability-Driven Instruction Scheduling

Instruction scheduling can significantly affect the temporal vulnerability of instructions and data, as it determines their residence time in different processor components. To improve the dependability of an application, several problems have to be addressed, which usually do not have to be considered for a performance-oriented instruction scheduling:

1. *Critical instructions should not be scheduled after multi-cycle instructions* or instructions potentially stalling the pipeline as this increases their temporal vulnerability;
2. *High residence time* (and therefore temporal vulnerability) of data in registers/ memory;
3. *High spatial vulnerability*, e.g., as a consequence of using more registers in parallel.

Therefore, the dependability-driven instruction scheduling in [21, 22] estimates the vulnerabilities, and separates the instructions into *critical* and *non-critical* ones statically at compile-time before performing the instruction scheduling. Afterwards, it targets minimizing the application dependability by minimizing the spatial and temporal vulnerabilities while avoiding scheduling critical instructions after multi-cycle instructions to reduce their residence time in the pipeline. These parameters are combined to an evaluation metric called *instruction reliability weight*, which is employed by a *lookahead-based heuristic* for scheduling the instructions. The scheduler operates at the basic block level and considers the reliability weight of an instruction in conjunction with its dependent instructions to make a scheduling decision. In order to satisfy a given performance overhead constraint, the scheduler also considers the performance loss compared to a performance-oriented instruction scheduling.

3.3 Dependability-Driven Selective Instruction Redundancy

While the dependability-driven software transformations and instruction scheduling focus on reducing the vulnerability and critical instruction executions, certain important instructions might still have to be protected in applications being highly susceptible to faults. Therefore, it is beneficial to *selectively protect important instructions using error detection and recovery techniques* [24, 31], while saving the performance/power overhead of protecting every instruction.

To find the most important instructions, the error masking and error propagation properties as well as the instruction vulnerabilities have to be estimated. These results are used afterwards to prioritize the instructions to be protected, considering the performance overhead and the reliability improvement. For this, a *reliability profit function* is used, which jointly considers the protection overhead, error propagation and masking properties and the instruction vulnerabilities. The results of this analysis are finally used to select individual or a group of instructions, which maximize the total reliability profit considering a user-provided tolerable performance overhead.

4 Dependability-Driven System Software

Based on the dependability modeling and estimation approaches and the dependability-driven compilation techniques, *multiple code versions* are generated. These code versions exhibit distinct performance and dependability properties while providing the same functionality. They are then used by the **run-time system** for exploring different *reliability-performance trade-offs* by selecting appropriate application versions while adapting to changing run-time scenarios (e.g., different fault rates and workloads) for single- and multi-core systems.

4.1 Joint Consideration of Functional and Timing Dependability

The key requirement of many systems is producing correct results, where a (limited) time-wise overhead is oftentimes acceptable. However, for real-time (embedded) systems both the *functional dependability* (i.e., providing correct outputs even in the presence of hardware-level faults) and the *timing dependability* (i.e., providing the correct output before the deadline) play a central role and need to be considered jointly trading-off one against the other [23, 27]. To enable this, multiple system layers (i.e., compiler, offline system software, and run-time system software) need to be leveraged in a *cross-layer framework* to find the most effective solution [15]. For an application with multiple functions, the problem is to compose and execute it in way that jointly optimizes the functional and timing correctness. For this, the *RTP* (see Sect. 2) is used as an evaluation metric.

Figure 4a presents an overview of our approach. It is based on multiple function versions generated by employing the approaches described in Sect. 3, where additionally even different algorithms might be considered. As an example, a *sorting* application is illustrated in Fig. 4b, where the vulnerability of different algorithms and implementations as well as their execution times are compared showing different trade-offs. For generating the versions, a dependability-driven

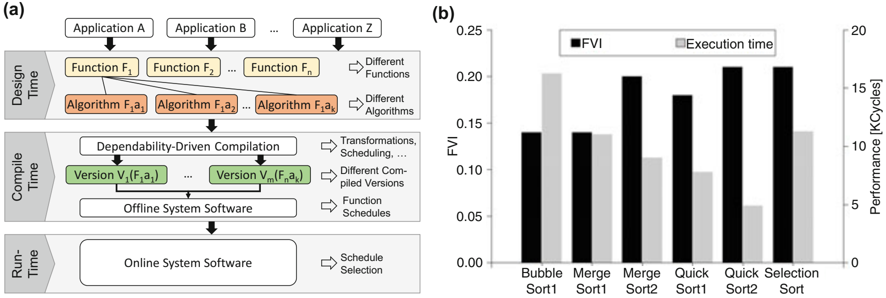


Fig. 4 (**a**) Overview of the design-time, compile-time, and run-time steps for generating different function/application versions. (**b**) Different algorithms and implementations for sorting (adapted from [23])

compilation process is used, given different implementations and a tolerable performance overhead to limit the design space. Then, only a limited number of versions from the *pareto-frontier* are selected, representing a wide spectrum of solutions.

In the next step, a *Dependability-Driven Offline System Software* generates **schedule tables** by minimizing the expected RTP. For the execution time, a probability distribution is considered, since it is not constant for all functions. For applications with only one function, the version minimizing the RTP (based on a weighting parameter) can be found by analyzing its probability for deadline misses and its reliability. For applications with multiple functions, it is required to consider that the selected version of a function is dependent on the functions executed earlier, e.g., if they finish early, a high-reliability version with a longer execution time can be selected. Therefore, a *dynamic version selection scheme* is adopted, where schedule tables are prepared offline and the scheduler selects appropriate function versions depending on the run-time behavior. Selecting a version for a particular function depends on both the functions executed *earlier*, and the functions executed *afterwards* (i.e. the predecessor and the successor functions in the execution path). The schedule tables are filled from the last function to be executed and remaining entries are added successively later, where the properties of earlier functions have to be explored and later functions can be captured by a lookup in the already filled parts of the table.

At run-time, a *Dependability-Driven Run-time System Software* selects an appropriate function version from the schedule table depending on the RTP. To execute the corresponding function, dynamic linking can be used. At the start of an application, the RTP is zero and the remaining time is the complete time until the deadline, as no function has been executed so far. With these parameters, the entry is looked up in the schedule table and the corresponding function version is executed. When one of the following functions need to be executed, the RTP observed so far is accumulated and the remaining time until the deadline is calculated. Afterwards, the corresponding table lookup is performed and a version is selected. To ensure

the correctness of the schedule tables, they should be placed in a protected memory part. As they, however, might become large, the size of the table can be reduced by removing redundant entries and entries where the RTP difference is too small. However, in this chapter, we assume that the system software is protected (for instance, using the approaches described in the OS-oriented chapters) and does not experience any failures.

In case the ordering of function executions is (partially) flexible, i.e., no/only partial precedence constraints exist, this approach can be extended by a function prioritization technique [27].

4.2 Adaptive Dependability Tuning in Multi-Core Systems

While Sect. 4.1 mainly focused on single-core systems and transient faults, the following technique will extend the scope towards multi-core systems and reliability threats having a permanent impact on the system (like process variation and aging). Thereby, different workloads on the individual cores might further aggravate the imbalance in core frequencies, which already preexists due to process variation. Consequently, *a joint consideration of soft errors, aging, and process variation is required* to optimize the dependability of the system. The goal is to achieve resource-efficient dependable application execution in multi-core systems under core-to-core frequency variation.

In a multi-core system, the software layer-based approaches can be complemented by *Redundant Multithreading (RMT)*, which is a hardware-based technique that executes redundant threads on different cores. An application can be executed with either Dual Modular Redundancy (DMR) or Triple Modular Redundancy (TMR). This broadens the mitigation solutions against the above-mentioned dependability threats, but also demands for the following problems to be solved [26].

1. The activation/deactivation of RMT has to be decided based on the properties (i.e., vulnerability, masking, performance) of the concurrently executing applications, the allowed performance overhead, and the error rate.
2. Mapping of (potentially redundant) threads to cores at run-time needs to consider the cores' states.
3. A reliable code version needs to be selected based on the performance variations of the underlying hardware and the application dependability requirements.

These problems are addressed by employing two key components: (1) a *Hybrid RMT-Tuning* technique, and (2) a *Dependability-Aware Application Version Tuning and Core Assignment* technique.

The **Hybrid RMT-Tuning** technique considers the performance requirements and vulnerability of the upcoming applications in combination with the available cores and history of encountered errors. It estimates the RTP of all applications, activates RMT for the one with the highest RTP in order to maintain the history, and

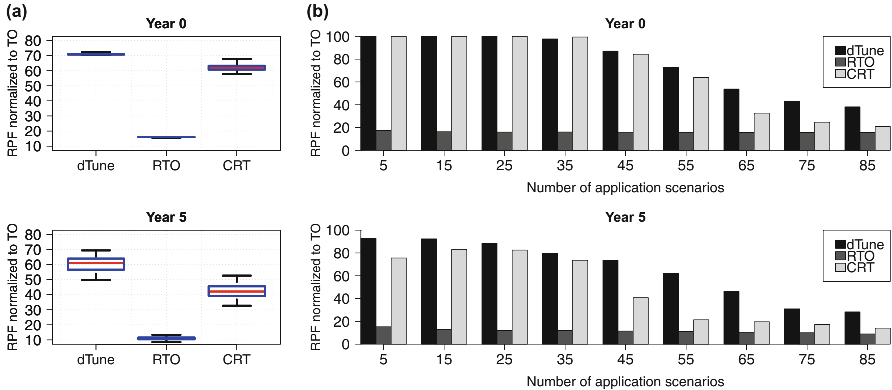


Fig. 5 (**a**) RPF improvements of *dTune*, *RTO*, and *CRT* normalized to *TO* for different aging years summarizing different chips and workloads. (**b**) RPF improvements detailing different workloads (adapted from [28])

takes RMT activation decisions based on the available cores and recent error history. For applications with RMT activated, the fastest compiled code version is selected.

After the RMT mode is decided for each application, the **Dependability-Aware Application Version Tuning and Core Assignment** is performed. It starts with an initial decision on the application version for applications where RMT has not been activated, considering their vulnerability and deadline. Then, the core allocation/mapping is performed, which takes the performance variations of individual cores (caused by process variation and aging) into account. It starts with the applications having the highest RTP and intends to allocate cores with similar performance properties to all redundant copies while also considering their distance. Finally, the application versions selected in the earlier step are tuned to improve the RTP further. Since the allocated core is now known, the potential for improving the dependability is evaluated considering the application's deadline.

Figure 5 shows the results of this approach (**dTune**) for different number of applications and different years. For the evaluation, a multi-core system with 10×10 ISA-compatible homogeneous RISC cores is used. These cores differ in their performance characteristics due to aging, where we consider NBTI-induced aging [1], and process variation, where the model of [18] is used. The comparison is done against three approaches: (1) *Chip-Level Redundant Threading (CRT)* which targets maximizing the reliability; (2) *Reliability-Timing Optimizing Technique (RTO)* jointly optimizing functional and timing dependability, but not using RMT; (3) *Timing Optimizing Technique (TO)* targeting to minimize the deadline misses. The evaluation is performed taking *TO* as a reference against which *dtune*, *CRT*, and *RTO* are compared with, where

$$\text{RPF} = 100 \times \left(1 - \frac{\sum_{t \in T} \text{RTP}(t)_Z}{\sum_{t \in T} \text{RTP}(t)_{TO}}\right). \tag{1}$$

Figure 5a shows an overview of the achieved improvements for different chip maps with process variations, scenarios of application mixes and aging years. *dTune* achieves better RPF-results compared to *TO*, *CRT*, and *RTO* for both aging years, as it jointly considers functional and timing dependability as well as the performance variation of the cores. For year 5, a wider spread of RPF-results is observed due to the decrease in processing capabilities of the chips. Figure 5b details the application workload, where it can be observed that *CRT* performs as good as *dTune* for a lower number of applications, but does not deal well with a higher number of applications due to focusing only on minimizing functional dependability.

The solution discussed above can further be enhanced by starting with a preprocessing for application version selection, as demonstrated in [5]. First, the version with the minimal reliability penalty achieving the tolerable miss rate (for applications not being protected by RMT) and the best performance (for applications protected with RMT) are selected. Afterwards, the application-to-core mapping problem is solved for the applications protected with RMT by assigning each of them the lowest-frequency group of cores possible. Then, the applications that are not protected with RMT are mapped to cores by transforming the problem to a minimum weight perfect bipartite matching problem, which is solved by applying the *Hungarian Algorithm* [16]. The decision whether to activate RMT or not is made by iteratively adapting the mode using a heuristic in combination with the application mapping approaches.

Nevertheless, solely adopting CRT to maximize the reliability is not good enough, since the utilization of the dedicated cores may be unnecessarily low due to low utilization tasks. If the number of redundant cores is limited, the number of tasks activating RMT is also limited. When the considered multi-core systems have multi-tasking cores rather than single thread-per-core (but homogeneous performance), the same studied problems, i.e., the activation of RMT, mapping of threads to cores, and reliable code version selection, can be addressed more nicely while satisfying the hard real-time constraints. The main idea is to *use Simultaneous Redundant Threading (SRT) and CRT at the same time or even a mixture of them called Mixed Redundant Threading (MRT)*. There are six redundancy levels characterized as a set of directed acyclic graphs (DAGs) in Fig. 6, where each node (sub-task) represents a sequence of instructions and each edge represents execution dependencies between nodes.

For determining the optimal selection of redundancy levels for all tasks, several dynamic programming algorithms are proposed in [8] to provide coarse- or fine-grained selection approaches while satisfying the feasibility under *Federated Scheduling*. In extensive experiments, the proposed approaches can generally outperform the greedy approach used in *dTune* when the number of available cores is too limited to activate CRT for all tasks. Since the fine-grained approach has more flexibility to harden tasks in stage-level, the decrease of the system reliability penalty is at least as good as for the coarse-grained approach. When the resources are more limited, e.g., less number of cores, the benefit of adopting the fine-grained approach is more significant.

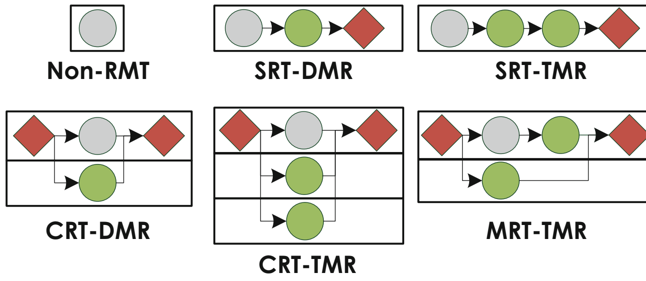


Fig. 6 DAG abstractions of the different redundancy levels, where the gray nodes are original executions and the green nodes are replicas. The red nodes represent the workload due to the necessary steps for forking the original executions and replicas, joining, and comparing the delivered results from DMR/TMR at the end of redundant multithreading. The directed edges represent the dependencies between nodes. Each block represents one core, i.e., the number of cores differs depending on the redundancy level

5 Resilient Design for System Software

Considering the adoption of error detection and recovery mechanisms due to the occurrence of soft errors from time to time, resilient designs for system software can be developed. (1) Execution versions can be determined to handle soft errors without over-provision while satisfying given robustness and timing constraints. (2) Dynamic timing guarantees can be provided without any online adaptation after a fault occurred. (3) Probabilistic analyses on deadline misses for soft real-time system. The detailed designs are presented in the following.

5.1 Adaptive Soft Error Handling

To avoid catastrophic events like unrecoverable system failures, software-based fault-tolerance techniques have the advantages in both the flexibility and application-specific assignment of techniques as well as in the non-requirement for specialized hardware. However, the main expenditure is the significant amount of time due to the additional computation incurred by such methods, e.g., redundant executions and majority voting, by which the designed system may not be feasible due to the overloaded execution demand. *Due to the potential inherent safety margins and noise tolerance, control applications might be able to tolerate a limited number of errors and only degrade its control performance.* Therefore, costly deploying full error detection and correction on each task instance might not be necessary.

To satisfy the minimal requirement of functional correctness for such control applications, (m, k) robustness constraint is proposed, which requires m out of any k consecutive instances to be correct. For each task an individual (m, k) constraint

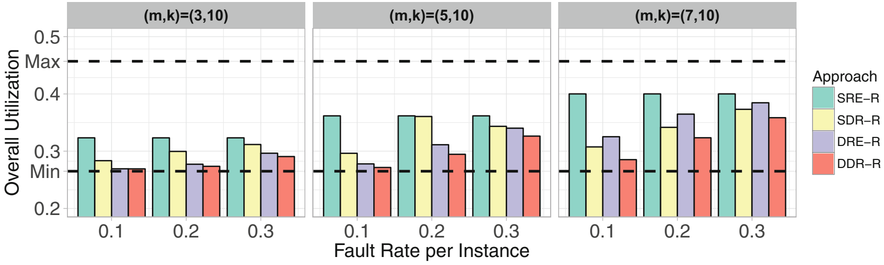


Fig. 7 Overall utilization after applying different compensation approaches on Task Path, where lower is better. Two horizontal and dashed bars represent the maximum (0.457) and the minimum utilization (0.265)

is possible to be given by other means analytically or empirically [35]. Without skipping any instances so likely achieving higher control performance, a static pattern-based approach [4] can be used to comply the reliable executions on the marked instances by following an (m, k)-pattern repeatedly to satisfy the given minimal requirement. To validate the schedulability, the multi-frame task model can then be applied to provide a hard real-time guarantee offline. A run-time adaptive approach [4] can further decide the executing version on the fly by enhancing the static pattern-based approach and monitoring the current tolerance status with sporadic replenishment counters. It is worth noting that the resulting distribution of execution jobs can still follow the (m, k) static patterns even in the worst case. Hence, the schedulability test for the static pattern-based approach can be directly used for the run-time adaptive approach as well.

Figure 7 shows the results for a self-balancing control application under different (m, k) requirements and varying fault rates. When the fault rate increases, the overall utilization of the run-time adaptive approach (DRE and DDR) also rises, since the requirement of reliable executions is increased within the application execution. Furthermore, the static pattern-based approaches (SRE and SDR) are always constant for a fixed (m, k) requirement, as the overall utilization is deterministic by the amount of job partitions. When the fault rate is as low as 10% and the (m, k) requirement is loose as (3, 10), the probability of activating reliable executions is rare, and, hence, the run-time adaptive approach can closely achieve the minimum overall utilization. Overall, the results suggest that the proposed approaches can be used to serve various applications with inherent fault-tolerance depending on their perspectives, thus *avoiding over-provision under robustness and hard real-time constraints*.

5.2 *Dynamic Real-Time Guarantees*

When soft errors are detected, the execution time of a real-time task can be increased due to potential recovery operations. Such recovery routines may make the system

very vulnerable with respect to meeting hard real-time deadlines. This problem is often addressed by aborting *not so important* tasks to guarantee the response time of the *more important* tasks. However, for most systems such faults occur rarely and the results of *not so important* tasks might still be useful, even if they are a bit late. This implicates to not abort these not so important tasks but keep them running even if faults occur, provided that the more important tasks still meet their hard real-time deadlines. To model this behavior, the idea of *Systems with Dynamic Real-Time Guarantees* [33] is proposed, which determines if the system can provide without any online adaptation after a fault occurred, either full timing guarantees or limited timing guarantees. Please note that, this study is highly linked to the topic of mixed-criticality systems [2]. We can imagine that the system is in the low-criticality mode if *full timing guarantees* are needed, and in the high-criticality mode if only *limited timing guarantees* are provided. However, in most of the related works, such mode changes are assumed to be known, without identifying the mode change. The system only switches from low-criticality to high-criticality mode once, without ever returning to the low-criticality mode. Moreover, the low-criticality tasks are considered to be either ignored, skipped, or run with best efforts as background tasks. Such a model has received criticism as system engineers claim that it does not match their expectations in Esper et al. [11], Ernst and Di Natale [10], and Burns and Davis [2].

Suppose that a task set can be partitioned into two subsets for *more important* and *not so important* tasks, and a fixed priority order is given. To test the schedulability of a preemptive task set with constrained deadlines under a fixed priority assignment, the typical **Time Demand Analysis (TDA)** as an exact test with *pseudo-polynomial run-time* can be directly applied. To determine the schedulability for a *System with Dynamic Real-Time Guarantees*, the following three conditions must hold:

- Full timing guarantees hold, if the given task set can be scheduled according to TDA when all tasks are executed in the normal mode.
- When the system runs with limited timing guarantees, all *more important* tasks will meet their deadlines if they can be proven to be scheduled by TDA while all tasks are executed in the abnormal mode.
- Each *not so important* task has bounded tardiness if the sum of utilization over all tasks in the abnormal mode can be less than or equal to one.

To decide such a fixed priority ordering for a given task set, the **Optimal Priority Assignment (OPA)** can be applied to find a feasible fixed priority assignment, since the above schedulability test is OPA compatible. It is proven that a feasible priority assignment for a *System with Dynamic Real-Time Guarantees* can be found if one exists by using the priority assignment algorithm presented in [33], which has a much better run-time than directly applying OPA.

As faulty-aware system design is desirable in the industrial practice, having an online monitor to reflect the system status is also important. This monitor should trigger warnings if the system can only provide limited timing guarantees, and display the next time the system will return to full timing guarantees. To achieve

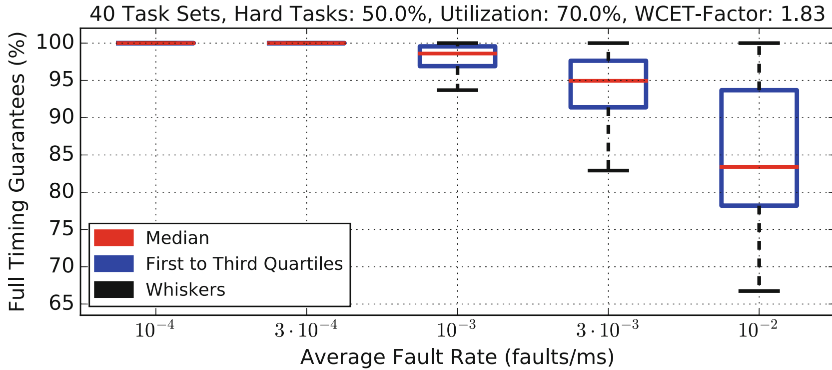


Fig. 8 Percentage of Time where *Full Timing Guarantees* can be given for task sets with utilization 70% in the normal mode under different fault rates. The median of the acceptance rates over 40 task sets is colored in red. The blue box represents the interval around this median that contains the inner 50% of those values while the whiskers display the range of the top/bottom 25% of those values

this, an approximation is needed to detect the change from *full timing guarantees* to *limited timing guarantees*, and for the calculation of an upper bound of the next time instance the system will return to full timing guarantees. To realize the routine of the online monitor, the system software has to ensure that the release pattern is still correct when a task misses its deadline and there is a helper function to keep tracking the number of postponed releases. How to enhance a real-time operating system for the previous two requirements is further discussed in [6].

Figure 8 shows the results with the percentage of time that the system was running with *full timing guarantees*. At a fault rate of 10^{-4} and 3×10^{-4} (faults/ms), the system always provides *full timing guarantees*. When the fault rate is increased, the average of the time where *full timing guarantees* are provided drops. For the worst-case values, the drop is faster but even in this case full timing guarantees are still provided $\approx$92.59% and $\approx$82.91% of the time for fault rates of 10^{-3} and 3×10^{-3}, respectively. This shows that even for the higher fault rates under a difficult setting, the system is still able to provide full timing guarantees for a reasonable percentage of time.

5.3 Probabilistic Deadline-Miss Analyses

When applying software fault-tolerant techniques, one natural assumption is that the *system functions normally most of time*. Therefore, it is meaningful to model the occurrence of different execution of a task by *probabilistic bounds on the worst-case execution time (WCETs)* due to potential recovery routines. This allows the system designer to provide probabilistic arguments, e.g., *Deadline-Miss Probability (DMP)*

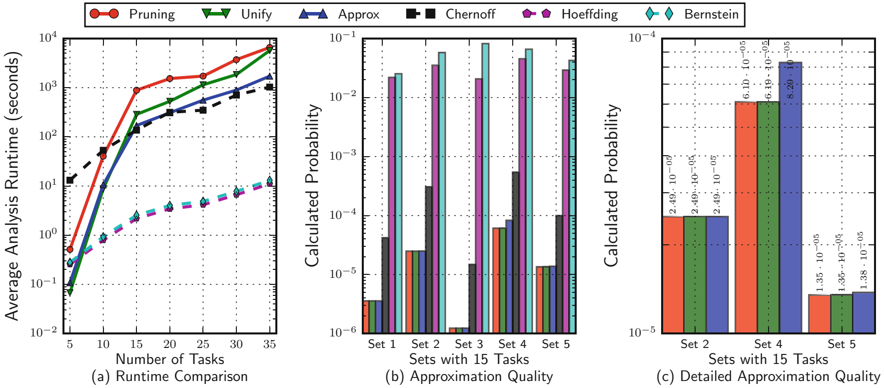


Fig. 9 (**a**) Average run-time with respect to task set cardinality. (**b**) Approximation quality for five task sets with Cardinality 15. (**c**) Detailed approximation quality for the convolution-based approaches

and *Deadline-Miss Rate*, as the statistical quantification to evaluate the proposed analyses scheduling algorithms, etc.

To derive the DMP, statistical approaches, i.e., *Probabilistic response time analysis* and *Deadline-misses probability analysis*, are usually taken into consideration. The state of the art of the probabilistic response time analysis is based on task-level convolution-based approaches [34]. Naturally, convolution-based approaches are computationally expensive to be applied when the number of tasks or jobs is large. Alternatively, *Deadline-Misses probability analysis* [3] is proposed, which can utilize analytical bounds, e.g., *Chernoff bounds* [3, 9], *Hoeffding's and Bernstein's inequalities* [34]. Please note that, the deadline-misses probability analysis is not better than the probabilistic response time analysis in terms of accuracy of the DMP. However, it is essentially much faster and has a better applicability in practice.

Figure 9 shows the results for randomly generated tasks sets with a normal-mode utilization 70%, fault rate 0.025, and for all tasks the execution time of abnormal mode is assumed to be two times of the normal mode. Three approaches based on the task-level convolution-based approaches [34], i.e., *Pruning, Unify, Approx*, result in similar values, roughly one order of magnitude better than *Chernoff* [3]. Although *Bernstein* [34] and *Hoeffding* [34] are orders of magnitude faster than the other approaches which are compatible with respect to the related run-time, the error of them is large compared to *Chernoff* by several orders of magnitude. The results suggest that, if sufficiently low deadline-miss probability can be guaranteed from analytical bounds, the task-level convolution-based approach then can be considered.

DMP and Deadline-Miss Rate are both important performance indicators to evaluate the extent of requirements compliance for soft real-time systems. However, the aforementioned probabilistic approaches all focus on finding the probability of the first deadline miss, and it is assumed that after a deadline miss the system either

discards the job missing its deadline or reboots itself. Therefore, the probability of one deadline miss directly relates to the deadline-miss rate since all jobs can be considered individually. If this assumption do not hold, the additional workload due to a deadline miss may trigger further deadline misses.

To derive a tight but safe estimation of the deadline-miss rate, an event-driven simulator [7] with a fault injection module can be used, which can gather deadline-miss rates empirically. However, the amount of time needed to perform the simulations is too large. Instead of simulating the targeted task set, an analytical approach [7] can leverage on the above probabilistic approaches that over-approximate the DMP of individual jobs to derive a safe upper bound on the expected deadline-miss rate.

6 Application-Specific Dependability

In this section, we focus on application-specific aspects on dependability improvement with the help of a case study on the *Context Adaptive Variable Length Coding (CAVLC)* used in the H.264 video coding standard [20, 30, 32]. It summarizes *how application-specific knowledge can be leveraged to design a power-efficient fault-tolerance technique for H.264 CAVLC.*

CAVLC is an important part of the coding process and is susceptible to errors due to its context adaptivity, multiple coding tables, and complex structure. It transforms an input with a fixed length to flexible-length code consisting of *codeword/codelength* tuples. The impact of a single error on the subjective video quality is illustrated in Fig. 10a, which shows a significant distortion in a video frame when the header of a macroblock (i.e., a 16 × 16 pixels block) is affected. Faults during the CAVLC can also propagate to subsequent frames or even lead to encoder/decoder crashes.

Consequently, it is required to address these problems during the CAVLC execution. To reduce the overhead compared to generic solutions, application-

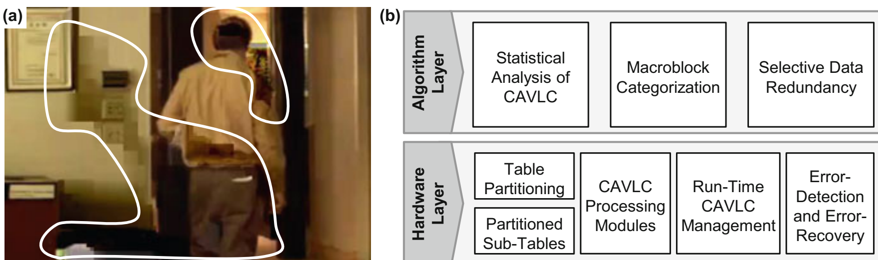


Fig. 10 (**a**) Example of a corrupted frame showing the effects of a single-bit error. (**b**) Overview of the contributions for the dependable CAVLC and the corresponding system layers (adapted from [32])

specific knowledge is considered. Specifically, Fig. 10b shows an overview of the dependable CAVLC with contributions on the architecture and algorithm/software layer, which are based on exploiting the video content properties and performing a statistical analysis of CAVLC.

- **Application-Specific Knowledge** is considered by (1) an analysis of error probabilities, (2) distribution of different syntax elements, (3) algorithmic properties, and (4) specifications defined by the standard. It includes an analysis of different macroblock categories (homogeneous/textured, fast/slow motion). The most important observations are that the *total non-zero coefficients have a significant influence on the error probabilities of different syntax elements*. They can be used to detect potential errors at the algorithm level if the macroblock properties are known.
- **Selective Data Redundancy:** Based on the application-specific knowledge obtained by the analysis, selected CAVLC data (e.g., quantized coefficients, coefficient statistics, etc.) can be protected by storing redundant copies and parity data in unused data structures. This is possible, e.g., for the quantized coefficients as the quantization often leads to unused ("0") entries, where redundant data can be stored in a reflected fashion. *Only the low-frequency coefficients are protected in case the space is insufficient.*
- **Dependable CAVLC Hardware Architecture:** The original and redundant values are loaded by a hardware module, which performs error detection and error recovery. In case of a mismatch, the parity is calculated and compared to the stored one, so that the correct entry can be found. A recovery is even possible if both entries are corrupted by reloading the original block and performing the quantization step again. Additionally, the coding tables used by CAVLC for obtaining the *codeword* and *codelength* need to be protected. For that, the individual tables are split into different sub-tables, where the partitioning decision is based on the distribution of the syntax elements. *Sub-tables not being accessed frequently can then be power-gated for leakage energy savings.* For each sub-table, a block parity-based protection approach is used for error detection, trading-off the additional memory required and the protection offered. Furthermore, entries not being accessed due to the algorithm properties and zero-entries are not stored. Similarly, *the data in tables containing mirrored entries also has to be stored only once*, thereby further reducing the memory requirements and leakage energy.
- **Run-Time Manager:** The dependable CAVLC architecture is controlled by a run-time manager which activates/deactivates the power-gating of the memory parts storing the sub-tables, loads the requested data from the tables, and controls error detection and reloading of data.
- **Dependable CAVLC Processing Flow:** The overall flow starts with a macroblock characterization, which determines the power-gating decision. Then, highly probable values for the syntax elements are predicted, which are used later for the algorithm-guided error detection. Afterwards, the header elements

are loaded by the hardware module performing error detection and error recovery. Finally, the quantized coefficients are coded by CAVLC for each 4×4 block.

This example architecture illustrates how application-specific knowledge can be leveraged to improve the design decisions for enhancing the dependability of the system and its power consumption. It achieves significant improvements in terms of the resulting video quality compared to an unprotected scheme. Moreover, *leakage energy savings of* 58% *can be achieved by the application-guided fault-tolerance and table partitioning.*

7 Conclusion

Dependability has emerged as an important design constraint in modern computing systems. For a cost-effective implementation, a cross-layer approach is required, which enables each layer to contribute its advantages for dependability enhancement. This chapter presented contributions focusing on the architecture, SW/OS, and application layers. Those include modeling and estimation techniques considering functional correctness and timeliness of applications as well as approaches for generating dependable software (e.g., by dependability-aware software transformations or selective instruction redundancy). Additionally, the run-time system is employed for selecting appropriate dependable application versions and adapting to different workloads and run-time conditions, enabling a tradeoff between performance and dependability. It has furthermore been shown how application-specific characteristics can be used to enhance the dependability of a system, taking the example of a multimedia application.

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500—spp1500.itec.kit.edu).

References

1. Alam, M.A., Kufluoglu, H., Varghese, D., Mahapatra, S.: A comprehensive model for PMOS NBTI degradation: recent progress. Microelectron. Reliab. **47**(6), 853–862 (2007). https://doi.org/10.1016/j.microrel.2006.10.012
2. Burns, A., Davis, R.: Mixed criticality systems-a review, 7th edn. Tech. rep., University of York (2016)
3. Chen, K., Chen, J.: Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In: 12th IEEE International Symposium on Industrial Embedded Systems, SIES 2017, Toulouse, 14–16 June 2017, pp. 1–8 (2017). https://doi.org/10.1109/SIES.2017.7993392

4. Chen, K., Bönninghoff, B., Chen, J., Marwedel, P.: Compensate or ignore? Meeting control robustness requirements through adaptive soft-error handling. In: Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems, LCTES 2016, Santa Barbara, 13–14 June 2016, pp. 82–91 (2016). https://doi.org/10.1145/2907950.2907952
5. Chen, K., Chen, J., Kriebel, F., Rehman, S., Shafique, M., Henkel, J.: Task mapping for redundant multithreading in multi-cores with reliability and performance heterogeneity. IEEE Trans. Comput. **65**(11), 3441–3455 (2016). https://doi.org/10.1109/TC.2016.2532862
6. Chen, K.H., Von Der Brüggen, G., Chen, J.J.: Overrun handling for mixed-criticality support in RTEMS. In: WMC 2016, Proceedings of WMC 2016, Porto (2016). https://hal.archives-ouvertes.fr/hal-01438843
7. Chen, K., von der Brüggen, G., Chen, J.: Analysis of deadline miss rates for uniprocessor fixed-priority scheduling. In: 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, 28–31 August 2018, pp. 168–178 (2018). https://doi.org/10.1109/RTCSA.2018.00028
8. Chen, K., von der Bruggen, G., Chen, J.: Reliability optimization on multi-core systems with multi-tasking and redundant multi-threading. IEEE Trans. Comput. **67**(4), 484–497 (2018). https://doi.org/10.1109/TC.2017.2769044
9. Chen, K.H., Ueter, N., von der Brüggen, G., Chen, J.: Efficient computation of deadline-miss probability and potential pitfalls. In: Design, Automation and Test in Europe, DATE 19, Florence, 25–29 March 2019
10. Ernst, R., Di Natale, M.: Mixed criticality systems - a history of misconceptions? IEEE Des. Test **33**(5), 65–74 (2016). https://doi.org/10.1109/MDAT.2016.2594790
11. Esper, A., Nelissen, G., Nélis, V., Tovar, E.: How realistic is the mixed-criticality real-time system model? In: RTNS, pp. 139–148 (2015)
12. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. CAD Integr. Circuits Syst. **32**(1), 8–23 (2013). https://doi.org/10.1109/TCAD.2012.2223467
13. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: Mibench: a free, commercially representative embedded benchmark suite. In: Proceedings of the Workload Characterization, 2001, WWC-4. 2001 IEEE International Workshop, WWC '01, pp. 3–14. IEEE Computer Society, Washington (2001). https://doi.org/10.1109/WWC.2001.15
14. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October 2011, pp. 69–78 (2011). https://doi.org/10.1145/2039370.2039384
15. Henkel, J., Bauer, L., Dutt, N.D., Gupta, P., Nassif, S.R., Shafique, M., Tahoori, M.B., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: The 50th Annual Design Automation Conference 2013, DAC '13, Austin, 29 May–07 June 2013, pp. 99:1–99:10 (2013). https://doi.org/10.1145/2463209.2488857
16. Kuhn, H.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart. **2**, 83–98 (1955). https://doi.org/10.1002/nav.20053
17. Oh, N., Shirvani, P.P., McCluskey, E.J.: Error detection by duplicated instructions in super-scalar processors. IEEE Trans. Reliab. **51**(1), 63–75 (2002). https://doi.org/10.1109/24.994913
18. Raghunathan, B., Turakhia, Y., Garg, S., Marculescu, D.: Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: Design, Automation and Test in Europe, DATE 13, Grenoble, 18–22 March 2013, pp. 39–44 (2013). https://doi.org/10.7873/DATE.2013.023
19. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Reliable software for unreliable hardware: embedded code generation aiming at reliability. In: Proceedings of the 9th International

Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October 2011, pp. 237–246 (2011). https://doi.org/10.1145/2039370.2039408
20. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: Revc: computationally reliable video coding on unreliable hardware platforms: a case study on error-tolerant H.264/AVC CAVLC entropy coding. In: 18th IEEE International Conference on Image Processing, ICIP 2011, Brussels, 11–14 September 2011, pp. 397–400 (2011). https://doi.org/10.1109/ICIP.2011.6116533
21. Rehman, S., Shafique, M., Kriebel, F., Henkel, J.: RAISE: reliability-aware instruction scheduling for unreliable hardware. In: Proceedings of the 17th Asia and South Pacific Design Automation Conference, ASP-DAC 2012, Sydney, 30 January–2 February 2012, pp. 671–676 (2012). https://doi.org/10.1109/ASPDAC.2012.6165040
22. Rehman, S., Shafique, M., Henkel, J.: Instruction scheduling for reliability-aware compilation. In: The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, 3–7 June 2012, pp. 1292–1300 (2012). https://doi.org/10.1145/2228360.2228601
23. Rehman, S., Toma, A., Kriebel, F., Shafique, M., Chen, J., Henkel, J.: Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations. In: 19th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2013, Philadelphia, 9–11 April 2013, pp. 273–282 (2013). https://doi.org/10.1109/RTAS.2013.6531099
24. Rehman, S., Shafique, M., Aceituno, P.V., Kriebel, F., Chen, J., Henkel, J.: Leveraging variable function resilience for selective software reliability on unreliable hardware. In: Design, Automation and Test in Europe, DATE 13, Grenoble, 18–22 March 2013, pp. 1759–1764 (2013). https://doi.org/10.7873/DATE.2013.354
25. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. IEEE Trans. CAD Integr. Circuits Syst. **33**(11), 1597–1610 (2014). https://doi.org/10.1109/TCAD.2014.2341894
26. Rehman, S., Kriebel, F., Sun, D., Shafique, M., Henkel, J.: dtune: leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In: The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, 1–5 June 2014, pp. 84:1–84:6 (2014). https://doi.org/10.1145/2593069.2593127
27. Rehman, S., Chen, K., Kriebel, F., Toma, A., Shafique, M., Chen, J., Henkel, J.: Cross-layer software dependability on unreliable hardware. IEEE Trans. Comput. **65**(1), 80–94 (2016). https://doi.org/10.1109/TC.2015.2417554
28. Rehman, S., Shafique, M., Henkel, J.: Reliable Software for Unreliable Hardware - A Cross Layer Perspective. Springer (2016). https://doi.org/10.1007/978-3-319-25772-3
29. Reis, G.A., Chang, J., Vachharajani, N., Rangan, R., August, D.I., Mukherjee, S.S.: Software-controlled fault tolerance. Trans. Archit. Code Optim. **2**(4), 366–396 (2005). https://doi.org/10.1145/1113841.1113843
30. Shafique, M., Zatt, B., Rehman, S., Kriebel, F., Henkel, J.: Power-efficient error-resiliency for H.264/AVC context-adaptive variable length coding. In: 2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, 12–16 March 2012, pp. 697–702 (2012). https://doi.org/10.1109/DATE.2012.6176560
31. Shafique, M., Rehman, S., Aceituno, P.V., Henkel, J.: Exploiting program-level masking and error propagation for constrained reliability optimization. In: The 50th Annual Design Automation Conference 2013, DAC '13, Austin, 29 May–07 June 2013, pp. 17:1–17:9 (2013). https://doi.org/10.1145/2463209.2488755
32. Shafique, M., Rehman, S., Kriebel, F., Khan, M.U.K., Zatt, B., Subramaniyan, A., Vizzotto, B.B., Henkel, J.: Application-guided power-efficient fault tolerance for H.264 context adaptive variable length coding. IEEE Trans. Comput. **66**(4), 560–574 (2017). https://doi.org/10.1109/TC.2016.2616313
33. von der Bruggen, G., Chen, K., Huang, W., Chen, J.: Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In: 2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, 29 November–2 December 2016, pp. 303–314 (2016). https://doi.org/10.1109/RTSS.2016.037

34. von der Brüggen, G., Piatkowski, N., Chen, K., Chen, J., Morik, K.: Efficiently approximating the probability of deadline misses in real-time systems. In: 30th Euromicro Conference on Real-Time Systems, ECRTS 2018, Barcelona, 3–6 July 2018, pp. 6:1–6:22 (2018). https://doi.org/10.4230/LIPIcs.ECRTS.2018.6
35. Yayla, M., Chen, K., Chen, J.: Fault tolerance on control applications: empirical investigations of impacts from incorrect calculations. In: 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems, EITEC@CPSWeek 2018, 10 April 2018, Porto, pp. 17–24 (2018). https://doi.org/10.1109/EITEC.2018.00008

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.