Power-Aware Fault-Tolerance for Embedded Systems

Mohammad Salehi, Florian Kriebel, Semeen Rehman,
and Muhammad Shafique

1 Introduction

High core integration in multi-/many-core chips can facilitate reliability management through exploiting different hardening modes considering variants of redundant multithreading (RMT) [17]. However, in such large-scale chips the maximum number of cores that can simultaneously operate is constrained by the thermal design power (TDP, i.e., the maximum amount of power a chip is expected to dissipate and the nominal value for the cooling system to be designed around). Under a given TDP budget either less cores can be powered-on at the full performance level (the power-gated cores are referred to as dark silicon) or relatively more cores can be powered-on at a lower performance level (referred to as “dim” or “gray” silicon) [16]. In case TDP is exceeded, the elevated on-chip temperatures beyond the cooling capacity aggravate reliability threats like temperature-dependent transient faults and aging [1, 5, 6], unless the chip is throttled down which may lead to performance degradation. Reliability management under TDP constraints becomes even more challenging in the presence of manufacturing process variations that result in chip-to-chip or core-to-core variations in the maximum operating frequency and leakage power. This chapter presents a system-level power–reliability management technique for dark silicon multi-/many-core processors that jointly accounts for transient faults, process variations, and the TDP constraint.

M. Salehi (✉)
Guilan University, Rasht, Iran
e-mail: mohammad.salehi@guilan.ac.ir

F. Kriebel · S. Rehman · M. Shafique
Technische Universität Wien (TU Wien), Wien, Austria
e-mail: florian.kriebel@tuwien.ac.at; semeen.rehman@tuwien.ac.at;
muhammad.shafique@tuwien.ac.at

Fig. 1 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

In this chapter, at first, the system modeling including the power consumption and reliability models, as well as the reliability techniques are presented. Then, the power, reliability, and performance tradeoffs at the software and hardware levels as well as for different hardening modes are studied. After that, the power–reliability management technique is presented. It jointly considers multiple hardening modes at the software and hardware levels, each offering distinct power, reliability, and performance properties. At the software level, it leverages multiple reliable code versions that vary in terms of their reliability, performance, and power properties. At the hardware level it exploits different protection features and different RMT modes, subjected to the manufacturing process variations and different operating conditions (like changing the voltage–frequency levels). Finally, a framework for the system-level optimization is introduced. It considers different power–reliability–performance management problems for many-core processors depending upon the target system and user constraints (i.e., power, reliability, and performance constraints).

The main contributions of this chapter in the scope of this book lie on the application, SW/OS, and architectural layers as illustrated in Fig. 1.

2 System Models

2.1 Power Consumption Model

Power consumption in digital systems consists of static power (e.g., due to sub-threshold leakage) and dynamic power (mainly dissipated due to the circuit switching activities). The power consumption, when the system is operating under a supply voltage and a corresponding maximum allowable frequency (we call it a voltage and frequency (V-f) level), can be written as Eq. 1. For the systems that support the dynamic voltage and frequency scaling (DVFS), different V-f levels are specified at the design time, while at run time, a V-f level is selected considering the

system workload [10]. The static power P_{Static} increases exponentially when the threshold voltage (V_{th}) decreases and is proportional to the supply voltage (V). The dynamic power $P_{Dynamic}$ is proportional to the circuit switching activity (α), load capacitance (C_L), operating frequency (f), and the square of the supply voltage (V) [11, 12, 14, 15].

$$P(V, f) = P_{Static} + P_{Dynamic} = I_0 e^{\frac{-V_{th}}{\eta V_T}} V + \alpha C_L V^2 f \quad (1)$$

2.2 Fault and Reliability Models

In this chapter, transient faults which appear randomly in the underlying hardware and then disappear after certain time are considered. Examples of transient faults are single- and multiple-bit upsets due to energetic radiation particle strikes, circuit metastability, signals cross-talk, voltage noises due to electromagnetic interference (EMI), etc. [1, 5]. Transient faults in the hardware level may manifest themselves as bit-flips in the memory or combinational logic, i.e., the so-called soft errors. These errors may propagate to the software level (e.g., as silent data corruption, crash, halt, and wrong register values) and may finally result in a software failure [5].

These transient faults occur randomly and are typically modeled as a Poisson process with rate λ . The fault rate increases exponentially with a decrease in supply voltage V , as Eq. 2 [11, 12, 15].

$$\lambda(V) = \lambda_0 10^{\frac{V_{max}-V}{\Delta}} \quad (2)$$

In Eq. 2, λ_0 is the raw fault rate at the maximum voltage V_{max} (i.e., the minimum value for fault rate λ) and the parameter Δ determines the amount of increase in fault rate with one step decrease in voltage.

The *software's vulnerability* to soft errors due to hardware-level transient faults at the instruction-level can be quantified by the function vulnerability index (FVI) model [8, 9]. This model projects the error probability for an application software considering vulnerabilities of different instructions (modeled using instruction vulnerability index (IVI)) when executing through different hardware units (e.g., different pipeline stages) in a core. The IVI refers to the probability of an instruction's result being erroneous. It accounts for *temporal vulnerabilities* of different instructions (i.e., different instructions have different execution latency, instruction dependency, and intervals of the operand values) as well as *spatial vulnerabilities* (i.e., different hardware components occupy different chip area and perform different operations) [8, 9]. Knowing the hardware-level fault rate (λ) and the software vulnerability to soft errors (FVI), the software failure rate can be projected as $\lambda(V) \cdot FVI$. Accordingly, the functional reliability FR of an application execution that is defined as the probability of failure-free execution of the application can be written as [11, 12, 15]:

$$FR(FVI, c, V, f) = e^{-\lambda(V) \cdot FVI \cdot \frac{c}{f}} \quad (3)$$

In Eq. 3, $\frac{c}{f}$ is the application execution time under the operating frequency f and c is the number of clock cycles that are required by the core to finish the application execution.

Besides the *functional reliability* of an application, in many systems (e.g., real-time embedded systems), it is also required that the application execution has to finish before a deadline, referred to as *timing reliability* (i.e., probability of meeting deadlines). To jointly consider the functional reliability FR and timing reliability TR , the functional–timing reliability model of Eq. 4 can be employed. In this model, the parameter $0 \leq \beta \leq 1$ specifies the priority for functional and timing reliability. For example, for the systems with tight timing constraints, lower values for β are considered, and for the systems with severe constraints of timing and reliability (e.g., hard real-time systems), $\beta = 0.5$ can be considered to represent the same priority for functional and timing reliability.

$$R = \beta FR + (1 - \beta)TR \quad (4)$$

The reliability for a single application execution given by Eq. 4 may not satisfy the reliability constraint of the target system. In the following section, we study reliability techniques and hardening modes that can be used for soft error mitigation and reliability improvement in many-core processors.

2.3 Reliability Techniques

One prominent technique for tolerating transient faults in many-core processors is *process level redundancy (PLR)*, where multiple identical copies of an application task are executed on different cores and the application finishes successfully if at least one of the task executions finishes successfully (i.e., the application execution fails only if all the task executions fail). Therefore, the *total application reliability* is defined as the probability of at least one application task being executed successfully. Suppose that n identical copies of an application task are executed on n different cores and possible faults can be detected with the probability of μ_{FD} . The total reliability of the application can be calculated as:

$$R_{total}(R_1, R_2, \dots, R_n) = \mu_{FD} \left(1 - \prod_{i=1}^n (1 - R_i) \right) \quad (5)$$

In Eq. 5, R_i is the reliability of the i -th task copy execution (given by Eq. 4). Here, it is assumed that (1) there is no spatial correlation between fault occurrences in different cores and (2) parallel task executions on different cores are not dependent from the viewpoint of fault propagation, i.e., a fault occurrence in a core does not

affect the operation of the other cores. This assumption is also considered for the other reliability techniques in this chapter in which we have parallel task executions on different cores.

2.3.1 Software Error Detection with Re-execution (SEDR)

In this technique, each application task is executed on a single core and a software error detection mechanism (e.g., software-based control flow checking and acceptance tests) is used for error detection. Here, if an error occurs during the task execution, the task is re-executed once again on the same core for error recovery.

Therefore, the reliability of this technique can be calculated by Eq. 6, where μ_{SFD} is the error detection coverage of the software error detection mechanism (i.e., the probability of detecting existing errors).

$$R_{SEDR}(R) = \mu_{SFD}(R + (1 - R)R) = \mu_{SFD}(2R - R^2) \quad (6)$$

Here, it is assumed that there is no temporal correlation between the fault occurrences in consecutive executions of the same task, i.e., a fault occurrence during an execution of a task does not affect the next execution of the same task. This assumption is also considered for the other reliability techniques in this chapter in which we have consecutive task executions on the same core.

2.3.2 Dual Modular Redundancy (DMR) with Re-execution (DMRR)

Software-based error detection in the SEDR technique may not provide a high error detection coverage and also it may not be useful for some applications, e.g., it may entail incurring extra delay that may not be acceptable for hard real-time systems. One practical and powerful error detection mechanism is the comparison of the output result. In this mechanism, two identical copies of each application task are executed on different cores in parallel and the output results of the task are compared for error detection (i.e., DMR is applied at the individual core level). If the comparison task finds that the results are in agreement, the result is assumed to be correct. The implicit assumption here is that it is highly unlikely that both task executions experience the identical errors and they produce identical erroneous results. If the results are different, an error has occurred during the task execution, and the task is re-executed on another core for error recovery. Let R_{cmp} be the reliability of the result comparison process. Assuming that the two cores are identical, each with a reliability R , the reliability of the DMRR technique can be calculated by Eq. 7.

$$R_{DMRR}(R, R_{cmp}) = R_{cmp} \left(R^2 + 2(1 - R)R^2 \right) = R_{cmp}(3R^2 - 2R^3) \quad (7)$$

2.3.3 Triple Modular Redundancy (TMR)

N-Modular redundancy (NMR) that is an $M - of - N$ system with N (an odd number) and $M = (N + 1)/2$ can be applied at the individual core level, where N copies of each task are executed on N different cores in parallel and the results of at least M of them are required to be identical for proper operation. Thus, the task execution fails when the majority voting task finds that fewer than M results are identical [13]. This is similar to the redundant multithreading (RMT) approach if considering architecture-level redundancy management or process level redundancy (PLR) approach if considering operating system-level redundancy management. Here, it is considered that TMR ($N = 3$) is applied at the individual core level, i.e., three copies of each task are executed in parallel on three different cores, and majority voting is performed on the results for error masking. Let R_{vot} be the reliability of the majority voting task. The reliability of TMR can be calculated by Eq. 8.

$$R_{TMR}(R, R_{vot}) = R_{vot} \left(R^3 + 3R^2(1 - R) \right) = R_{vot} \left(3R^2 - 2R^3 \right) \quad (8)$$

3 Power–Reliability–Performance Tradeoffs

3.1 Tradeoffs at the Hardware Level

Due to technology process variations, the maximum operating frequency and the leakage power consumption vary for different cores in a single chip [3]. Figure 2a illustrates that the core-to-core frequency and leakage power variations in an Intel's 80-core test chip are up to 38 and 47%, respectively [7]. Therefore, regardless of which application is executed, different processing cores present different performance and power consumption.

Fig. 2 (a) Core-to-core variations in maximum operating frequency and leakage power and (b) hardware-level fault rate and power variations at different V-f levels. Adapted from [15]

Fig. 3 Core-to-core variations in power, execution time, and reliability when executing the same application. Adapted from [15]

One effective way to reduce power consumption is to decrease the operating V-f level through the DVFS technique. However, based on Eq. 2, decreasing the V-f level increases the hardware-level fault rate. Figure 2b shows that how the total power consumption (Eq. 1) and the hardware-level fault rate (given by Eq. 2) for a given core vary at different V-f levels. For the processor cores in our experiments, it is considered that the V-f level can have five different values as shown in Fig. 2b, i.e., the minimum V-f level is [0.72 V, 490 MHz] and the maximum V-f level is [1.23 V, 970 GHz], see details in Sect. 5.

Due to the core-to-core variations in the frequency and leakage power, when a given application is executed on different cores but under the same supply voltage, it presents different power consumption, performance (execution time), and reliability. Figure 3 shows core power consumption, application reliability, and execution time for the discrete cosine transform (DCT) application when it is executed on different cores but under the same supply voltage. Figure 3a illustrates that due to core-to-core variations in operating frequency, executing a given application on different cores presents different power consumption and performance properties.

According to Eq. 3, the application reliability depends upon the hardware-level fault rate, software vulnerability, and application execution time. Figure 3b illustrates that when a given application is executed on different cores, it provides different reliability levels. This is because, in this case, software vulnerability (FVI) and hardware-level fault rate (λ) remain the same in Eq. 3 (the same application is executed under the same voltage level) but due to core-to-core variations in operating frequency, the application execution time ($\frac{C}{f}$) varies when executed on different cores.

The analyses in Figs. 2 and 3 illustrate that the diversities in power and performance of different cores in a chip when executing the same application along with exploiting different V-f levels can be utilized for efficient reliability management at hardware level.

3.2 Tradeoffs at the Software Level

Since different applications execute different instructions on different operand values, they present different power, performance, and reliability properties even when executed on the same core and under the same V-f level. Figure 4 shows the power consumption, execution time, software vulnerability, and reliability for different applications when executed on the same core and under the same V-f level. Different applications exhibit different circuit switching activity and also require different clock cycles to complete, and hence, they exhibit distinct power consumption and execution time properties even when executed on the same core; see Fig. 4a.

Also, since different instructions present different vulnerabilities to soft errors (e.g., single event upsets), as shown in Fig. 4b, different applications exhibit distinct software vulnerabilities. Figure 4b shows that different applications, even when executed under the same V-f level (i.e., under the same hardware-level fault rate) and on the same core, exhibit different system-wide reliability. This is because, based on Eq. 3, the application reliability also depends upon its software vulnerability and execution time.

The above analysis shows that different applications exhibit different power, performance, and reliability levels when executed on different cores, thus enabling power–reliability–performance tradeoffs at software level.

3.3 Tradeoffs for Hardening Modes

3.3.1 Tradeoffs for Reliability Techniques

Reliability techniques usually employ *different types of redundancy* (e.g., hardware, software, and time redundancy) and *different redundancy levels* (e.g., dual or triple modular redundancy). Therefore, they offer different reliability, performance, and power properties. Also, two different reliability techniques may provide the same error tolerance capability but at different performance and power cost. For example,

Fig. 4 Application-to-application variations in power, execution time, software vulnerability, and application reliability when executed by the same core. Adapted from [15]

Fig. 5 System-wide reliability and power for different reliability techniques when performed at the minimum and maximum V-f levels ($V\text{-}f_{\min}$ and $V\text{-}f_{\max}$, respectively). Adapted from [12]

both the DMRR and TMR techniques can tolerate one single task failure; however, when an error occurs, DMRR requires more time to re-execute the task for error recovery, which incurs a performance overhead. Nevertheless, DMRR may consume less power and energy when compared to TMR. This is because when no error occurs, which could be a case for most of the time, DMRR does not require re-executing the task, while TMR always executes the third copy.

Figure 5 shows system reliability and power consumption when the reliability techniques in Sect. 2.3 are employed. To illustrate the effects of scaling the operating V-f level on the system reliability and power consumption, the reliability techniques are executed under the minimum and maximum V-f levels (i.e., $V\text{-}f_{\min}$ and $V\text{-}f_{\max}$). Also, in this figure, reliability techniques with different redundancy levels are considered (i.e., SEDR with a low redundancy level and TMR with a high redundancy level). Figure 5 illustrates that increasing the redundancy level (from SEDR to TMR) and V-f level (from $V\text{-}f_{\min}$ to $V\text{-}f_{\max}$) improves reliability but at the cost of increased power consumption.

The experiment in Fig. 5 shows that different reliability techniques when operating in different V-f levels exhibit distinct power, performance, and reliability properties, enabling power–reliability–performance tradeoffs that can be employed for power–reliability management.

3.3.2 Tradeoffs for Software Hardening

To further expand the power–reliability–performance optimization space, a reliability-aware compiler can be used to generate multiple reliable compiled code versions for a given application task through reliability-driven software

Fig. 6 Different compiled code versions of each application have different: (a) Power and performance (execution time); and (b) Software vulnerability (in log scale) and application reliability. Adapted from [15]

code transformations (see more details in [8, 9]). Different code versions of the same task present dissimilar power, reliability, and performance properties while implementing the same functionality. For instance, Fig. 6 shows power, execution time (in terms of clock cycles), software vulnerability, and overall reliability (given by Eq. 3) of different compiled code versions for five applications.

The reliability and execution time of an application task also vary with the operating V-f level of the underlying core. Figure 7 shows the reliability and execution time of three code versions for the *ADPCM* application under different V-f levels. Figure 7a illustrates that how different code versions of the *ADPCM* application when executed under different V-f levels can be used to achieve a given reliability requirement for the application (R_{req} in this figure). For instance, to meet the reliability requirement $R_{req} \geq 0.999$, shown by the dotted horizontal line in Fig. 7a, the operating V-f level for the code versions *cv1* and *cv2* should be at least [0.97 V, 730 MHz], whereas the V-f level for the code version *cv3* can be [0.85 V, 650 MHz]. Assume that the application execution has a deadline constraint to finish within 5ms, as shown by the dotted horizontal line in Fig. 7b. In this case, the operating V-f level for the code version *cv2* should be at least of [0.85 V, 650 MHz], for *cv1* should be at least [0.97 V, 730 MHz], and for *cv3* should be at least [1.1 V,

Fig. 7 Reliability and execution time of three compiled code versions for the ADPCM application under different voltage–frequency levels. Adapted from [11]

850 MHz]. Now assume that the underlying core has a TDP constraint that requires its operating V-f level should be at most [0.85 V, 650 MHz] (the TDP1 constraint in Fig. 7). Under the TDP1 constraint, to meet the reliability constraint we should select the code version *cv3*; see Fig. 7a. However, under the given TDP1 constraint, if the deadline constraint has to be met, we would select the code version *cv2*; see Fig. 7b. As another example assume that the core has the TDP2 constraint (i.e., the operating V-f level for the core should be at most [0.97 V, 730 MHz]). In this case, the code version *cv1* is the best choice since it can satisfy both the reliability constraints of $R_{req} \geq 0.999$ and the deadline constraints within 5 ms while meeting the power constraint TDP2.

4 Power–Reliability–Performance Management

From Sects. 2 and 3, the following key observations can be derived that lay the foundation of designing an efficient system for power–reliability–performance Management.

1. Executing tasks at a higher V-f level provides lower execution time and fault rate, resulting in higher system-wide reliability. However, the task power consumption at a high V-f level may be beyond the chip power constraint.
2. An effective way to decrease the power consumption is to lower the operating V-f level, e.g., through DVFS. However, lowering the V-f level leads to an increased execution time of the task that may result in a performance degradation and a missed deadline.
3. Different compiled code versions for an application task exhibit different vulnerability and execution time properties when executed on the same core.
4. Different compiled code versions for each application task when executed by different reliability techniques on different cores with frequency variations and supporting different V-f levels present distinct power, reliability, and performance properties.

In short, the variations in vulnerability and execution time of different compiled versions for each task along with the variations in reliability, power, and performance when using different reliability techniques and V-f levels can be exploited for power–reliability–performance optimization.

The previous works, dynamic redundancy and voltage scaling (DRVS) [12] and dark silicon reliability management (dsReliM) [11], consider the above-mentioned variations at hardware and software levels for run-time power–reliability management. DRVS exploits run-time reliability technique (task-level redundancy) with V-f selection for each application task to minimize system power consumption under reliability and timing (deadline) constraints. dsReliM leverages multiple pre-compiled code versions for each application task with V-f selection at run time to maximize reliability under timing and power constraints. However, such techniques that solely use task-level redundancy or pre-compiled code versions may impose the following restrictions on power–reliability management. Although task-level redundancy can substantially increase reliability, it may increase chip power consumption beyond its power constraint. Also, this technique can only be used if sufficient cores are available for task-level redundancy. In this case, it may be useful to leverage reliable compiled codes to improve reliability, since no extra cores are required to execute different code versions for each application task. Although compile-time software hardening can decrease power consumption, it may increase the execution time of the tasks beyond their timing constraint. Therefore, power–reliability management requires joint considerations of reliability, performance, and power properties of hardening techniques at both software and hardware levels, which is the primary consideration of this chapter.

4.1 Problem Definition

System reliability and total power consumption, V-f level and code version assignments, and task-to-core mapping are represented by different matrices with $n \times m \times c \times v$ elements. Here, n is the number of ready tasks, m is the number of available code versions for each task, c is the number of free cores, and v is the number of available V-f levels for each core. The matrices are:

- $R \in \mathbb{R}^{n \times m \times c \times v}$: A matrix to represent the system reliability. In this matrix, each element $R_{i,j,k,l}$ represents the reliability of the task i when the code version j of the task is executed by the core k under the V-f level l .
- $P \in \mathbb{R}^{n \times m \times c \times v}$: A matrix to represent the system total power consumption. In this matrix, each element $P_{i,j,k,l}$ represents the power consumption for the task i when the code version j of the task is executed by the core k under the V-f level l .
- $X \in \{0, 1\}^{n \times m \times c \times v}$: A matrix to represent the code version and V-f level assignments and task-to-core mapping. Code version j for the task i is mapped to the core k and is executed under the V-f level l if and only if $X_{i,j,k,l} = 1$.

Considering power, reliability, and performance as a design object or a design constraint, the potential goals of a power-aware reliable system design can be:

1. Maximize system reliability while keeping total power consumption under a given power constraint (e.g., TDP) and meeting tasks timing requirements (e.g., tasks deadlines) OR
2. Minimize power consumption while satisfying the system reliability and timing requirements.

The power–reliability–performance management problems can be formulated as a constrained 0-1 integer linear program (ILP). In the following, the problem is formulated where reliability is the design objective, while power and performance are the design constraints. That is,

- **Optimization Goal:** Maximizing the system reliability that is defined by the correct execution of all the application tasks.

$$\text{maximize } \prod_{i,j,k,l} X_{i,j,k,l} R_{i,j,k,l} \quad (9)$$

This is a 0–1 assignment problem, and hence, we have

$$X_{i,j,k,l} \in \{0, 1\} \quad (10)$$

- **Chip Power Constraint:** Total power consumption of the chip, i.e., the sum of power consumption of all cores should be less than the chip power constraint (i.e., chip-level TDP).

$$\sum_{i,j,k,l} X_{i,j,k,l} P_{i,j,k,l} \leq P_{TDP,chip} \quad (11)$$

- **Cores Power Constraint:** Power consumption of each core should be less than the core power constraint (i.e., core-level TDP).

$$X_{i,j,k,l} P_{i,j,k,l} \leq P_{TDP,k} \quad (12)$$

- **Tasks Timing Constraint:** The execution time $\frac{w_{i,j}}{f_{k,l}}$ of a task i when the code version j of the task (with $w_{i,j}$ clock cycles) is executed on the core k at the V-f level l should satisfy the task timing constraint (defined by the deadline d_i).

$$X_{i,j,k,l} \frac{w_{i,j}}{f_{k,l}} \leq d_i \quad (13)$$

- **Code Version Constraint:** The code version does not change during a task execution, i.e., for each execution of a task only one code version can be used.

$$\forall i, k, l \sum_j X_{i,j,k,l} = 1 \quad (14)$$

- **V-f Levels Assignment Constraint:** The V-f level does not change during a task execution, i.e., during a task execution the underlying core can only perform under a single V-f level.

$$\forall i, j, k \sum_l X_{i,j,k,l} = 1 \quad (15)$$

4.2 Proposed Solution

The *power-aware fault-tolerance (PAFT)* technique in this chapter jointly accounts for soft errors, process variations, user defined reliability constraint, and processor power constraint (i.e., TDP). At design time, considering the inherent software-level variations in the execution time of the applications, power, and vulnerability, the PAFT technique selects suitable code versions from multiple compiled codes for each application task (Sect. 4.2.1). At run time, considering the hardware-level variations in performance, fault rate, and power, the PAFT approach selects the hardware/software hardening mode (i.e., reliability technique and code version for each task) and performs task mapping and V-f level allocation (Sect. 4.2.2).

4.2.1 Design-Time Code Selection

As discussed in Sect. 3.3, different compiled code versions for an application task and also different reliability techniques (with different redundancy levels) exhibit different reliability, performance, and power properties. Therefore, for each application task, a tradeoff can be made between two cases:

1. Exploiting a code version with higher reliability and a reliability technique with lower redundancy level (e.g., SEDR) to achieve both high reliability and low power consumption.
2. Exploiting a code version with higher performance and a reliability technique with a higher redundancy level (e.g., TMR) to achieve both high performance and high reliability.

To enable the above tradeoff at run time, we leverage the design-time generated multiple code versions for each task using a reliability-aware compiler (see details in [8, 9]). Then, two types of code versions are chosen as follows (as shown in Fig. 8):

1. **Reliability-Driven Code Selection:** At run time, the reliability of executing a code version of a task on a single core (in the SEDR mode) may be high enough to satisfy the system reliability requirement. In this case, for the task, there is no need to employ a reliability technique with a higher redundancy level. However, the execution time of all the code versions with high reliability, even when executed on a high-performance core and at the maximum V-f level may not be low enough to meet the task deadline constraint. Also, the power consumption of a code version with high reliability may be higher than the processor power budget. Therefore, at design time, for each application task, a set of code versions with high reliability, low execution time, and low power consumption is selected. To do this, first we find the reliability-wise best code versions. Then, from the highly reliable code versions, the performance-wise best code versions (i.e., the code versions with the lowest execution time) are selected. Finally, from the selected code versions, the code versions that provide the lowest power consumption are chosen.
2. **Performance-Driven Code Selection:** The reliability of executing a single task on a single core (in SEDR mode) may not be high enough to satisfy the task reliability requirement. In this case, the task can be executed under a reliability technique with a higher redundancy level (e.g., in the DMRR or TMR mode) to improve its reliability. Here, a code version with a high performance is executed under a high redundancy level to make a balance between timing and functional reliability. Therefore, the performance-wise best code versions with high reliability and low power consumption are chosen.

Fig. 8 Overview of the design-time part of the power-aware fault-tolerance (PAFT) technique

4.2.2 Run-Time Hardening Mode and V-f Level Selection and Task-to-Core Mapping

The run-time part of the PAFT technique is shown in Fig. 9. It chooses the hardening mode (reliable code version and reliability technique), operating V-f levels and set of cores to implement the reliability techniques, such that the design objectives are achieved while satisfying the design constraints (e.g., maximized system reliability under timing and power constraints). The problem can be effectively solved by the use of existing ILP solvers (formulated in Sect. 4.1). Since 0-1 ILP problems belong to the class of NP-complete problems, ILP solvers generally exploit branch-and-bound mechanisms to find the optimal solution which leads to an exponential increase in their run-time complexity. Therefore, ILP solvers cannot be used in online scenarios where the parameters that are required for decision-making are

Fig. 9 Overview of the run-time part of the power-aware fault-tolerance (PAFT) technique

determined at run time (e.g., ready tasks and free cores). In the case of this problem, the complexity of ILP solvers increases at run time with the number of ready tasks, code versions for each task, reliability techniques, free cores, and V-f levels. Therefore, for this problem, a heuristic is developed, which at first aggressively chooses the hardening mode, operating V-f levels and set of cores in such a way that the highest possible reliability and performance are obtained. Afterwards, it iterates and updates the hardening mode, operating V-f levels and task-to-core mapping until the design constraints (e.g., reliability, deadline, and power constraints) are satisfied. To do this, the run-time part of the PAFT technique gives the chip processor variation map, chip-level redundancy map, design constraints, and library of selected code versions for each application task as input and performs the following four key steps to each ready application:

- 1. Initial Hardening Mode Assignment:** First, a reliability-wise best code version is assigned to each task to achieve the highest possible functional reliability for each task. Then, beginning from the task with the lowest reliability, the reliability-wise best technique is assigned to the reliability-wise worst tasks. Therefore, the reliability of the tasks with the lowest reliability is improved, resulting in an improvement in the overall system reliability (the overall system reliability is less than or equal to the reliability of the task with the lowest reliability).
- 2. Initial Mapping:** In this step, starting from the task with the highest execution time (lowest performance), the performance-wise worst tasks are mapped on the performance-wise best cores (cores with the highest operating frequency).

3. **Finding the Base Solution:** Until now, the highest possible reliability and performance have been achieved for the tasks which may be higher than the system performance and reliability requirements. Also, this may lead to an increased chip power consumption beyond its power constraint. In this step, starting from the task with the highest power consumption, the redundancy and V-f level assigned to the task are increased to reduce power consumption until the point that the chip power constraint is satisfied. Here, reducing the operating V-f level may lead to a task deadline miss. In this case, the task code is replaced with a code version with less execution time.
4. **Updating the Base Solution:** At run time, missed deadlines can be considered for performance monitoring and the number of encountered errors can be considered for making the reliability decision. Also, the power information can be acquired from a proxy power monitor. By the use of the run-time reliability, performance, and power monitoring information the system considers the following cases for power–reliability–performance management:
 - (a) When the execution of a task finishes, the free cores are employed to improve the system reliability and performance through updating reliability techniques and task-to-core mapping.
 - (b) When an error occurs, after tolerating the error by the use of a recovery mechanism, the redundancy and V-f levels assigned to the tasks are increased to compensate the reliability degradation and also to provide high reliability against possible consequent errors.
 - (c) When chip power consumption approaches its power constraint, redundancy and V-f levels are decreased to reduce power consumption.

Since estimating the tasks reliability, power, and performance properties is time-consuming, the decisions in steps (a)–(c) are made at run time based on the reliability, power, and performance values that are obtained through the design-time measurements and simulations.

Figure 10 shows how the run-time part of the system works on a ready task. In this figure, for simplicity of the explanations, it is assumed that the reliability technique (e.g., SEDR, DMR, and TMR) and the underlying cores for the task execution are already determined and now the code version and V-f level should be chosen under timing (deadline) and power (TDP) constraints. Suppose that, for the task, the design-time code selection part has selected different pre-compiled code versions ($cv1$, $cv2$, $cv3$, ...) with different reliability and execution time properties. At run time, to achieve maximum reliability without considering the deadline and power constraints, the code version with the highest reliability (i.e., $cv1$ in Fig. 10) is selected to be executed at the maximum V-f level ($V\text{-}f_{\max}$). In Fig. 10, without loss of generality, it is assumed that the execution time of $cv1$ at $V\text{-}f_{\max}$ is less than its deadline but its power consumption at $V\text{-}f_{\max}$ exceeds the chip power (TDP) constraint. In this case, the V-f level is scaled down until the power consumption decreases below the TDP constraint. Suppose the case where reducing the V-f level to a lower level increases the task execution time beyond the task deadline (the task timing constraint is missed). In this case, the code is changed to a version with less

Fig. 10 Code version and V-f level assignment for reliability management under timing (deadline) and power (TDP) constraints

execution time (the code version that can meet the deadline). Here, among the code versions that can meet the deadline, the one is selected that provides the maximum execution time reduction and the minimum reliability loss (i.e., the code version with the maximum $\Delta time/\Delta reliability$). However, if there is no code version that can meet the deadline, to achieve the minimum performance degradation, the one with the minimum execution time is selected. After selecting the suitable code version, the V-f level is scaled down and if needed, the code version is updated until the TDP constraint is met (as shown in Fig. 10).

5 Experimental Setup and Results

Figure 11 shows the experimental setup and evaluation framework. Experiments were conducted by the use of a system-level many-core simulator developed in the C/C++ language. Accurate power and performance (execution time) parameters of applications and underlying hardware were provided for the simulator through processor synthesis, logic simulation, and power estimation. To do this, the Synopsys Design Compiler and a TSMC 45 nm technology file were used to synthesize a VHDL implementation of a LEON3 processor core [2]. Different benchmark applications of *MiBench* [4] (listed in Fig. 4) were used, and multiple compiled code versions for each application task were generated by the use of a reliability-aware compiler of [8, 9]. ModelSim was used for logic simulation to acquire execution time (clock cycles) and activity factors for each compiled code versions of each application. Power estimation was done using the Synopsys Power Compiler with the process synthesis and logic simulation outputs.

As another input for the simulator, the process variation maps were generated through SPICE simulations. The frequency and leakage power variations were modeled through simulating a 13-stage ring oscillator containing *FO4* inverters based on two-input NAND gates (like in [11, 12, 15]). To implement DVFS for

Fig. 11 The experimental setup and simulation flow

the processor cores, it is considered that the voltage level can change from 0.72 to 1.23 V, with 0.13 V steps, and the corresponding frequency to the minimum and maximum voltage levels is 490 MHz and 970 GHz, respectively.

For reliability evaluations, multiple fault vectors were generated by a Poisson process where the transient fault rate at different V-f levels was modeled based on Eq. 2 with $\lambda_0 = 10^{-4}$ and $\Delta = 1V$. Also, for the system, two types of reliability requirements were considered: (1) *functional reliability (FR)* where only correct output of the tasks is required and the tasks have no deadline and (2) *functional-timing reliability (FTR)* where both correct output of the tasks and meeting tasks deadlines are required. For FTR, for each task, we considered a deadline between its execution time and $1.5 \times$ its execution time. Considering the stochastic behavior of transient faults, multiple combinations of benchmark applications were executed for 100,000 times (as a Monte Carlo simulation) and reported the average results.

To model chip power budget, different TDP constraints were considered for each chip between 40 and 100% of its maximum power consumption when all cores perform at their maximum V-f level. This determines a wide range of TDP from a high TDP constraint where up to 40% of the cores can perform at their highest V-f level (i.e., at least 60% dark silicon) to no TDP constraint where all of the cores can perform at their highest V-f level (i.e., 0% dark silicon). Also, two types of system workload were considered in the experiments: (1) *high workload*, when the number of ready tasks is more than 50% of available cores and (2) *low workload*, when the number of ready tasks is less than 50% of the number of available cores.

To evaluate the accuracy and run-time efficiency of the *power-aware fault-tolerance (PAFT)* technique in finding solutions at run time, it was compared with the following techniques:

- dsReliM [11]: which uses compile-time software hardening (different reliable code versions) with run-time code version and V-f level selection.
- DRVS [12]: which exploits run-time task-level redundancy through the SEDR, DMRR, and TMR modes (explained in Sect. 3.3) with V-f level selection.
- ILP Solver: which exploits both compile-time software hardening and run-time task-level redundancy with code version and V-f selection. It searches for the

Fig. 12 Reliability and execution time for different power–reliability management techniques. (a) Reliability under low workload. (b) Reliability under high workload. (c) Execution time. Adapted from [15]

optimum solution through ILP solving. For this system, Gurobi¹ was used as a well-known ILP solving tool.

The results of the reliability and execution time efficiency evaluations are shown in Fig. 12. From this figure, the following observations can be made:

- All four techniques achieve higher reliability when there is no timing constraint for the tasks (denoted by FR in Fig. 12) compared to the case when the tasks have a timing constraint (denoted by FTR in Fig. 12). This is because when tasks have no deadline, a higher task-level redundancy (more task copies) can be considered for the execution of the tasks, resulting in a higher reliability. In addition, highly reliable code versions even with a high execution time can be executed. However, when there are timing constraints only the code versions that can satisfy the timing constraints can be selected. For the same reasons, similar results are obtained for higher system workloads (see Fig. 12b).
- All four techniques provide higher reliability when the chip power budget increases from 40 to 100% of the maximum chip power consumption. This is because more cores can be powered-on and higher redundancy levels can be leveraged for more task executions. In addition, highly reliable code versions even with higher power consumption can be executed.
- From the viewpoint of accuracy, reliability levels provided by PAFT deviate far less than one order of magnitude from the optimum reliability provided by ILP Solver, while the execution time of PAFT is up to 1680× less than the execution

¹<http://www.gurobi.com/>.

time of ILP Solver for an 8×8 cores chip (Fig. 12c shows the average execution time for chips with 4×4 , 6×6 , and 8×8 cores).

- As Fig. 12c shows, the execution time for PAFT is up to 3% higher than the execution time of DRVS and dsReliM, while it achieves at least one order of magnitude more reliability. This is because PAFT leverages both task-level redundancy and code version and V-f selection to discover better tradeoffs between reliability, power, and performance.

6 Conclusion

This chapter presents a power-aware fault-tolerance technique (PAFT) that jointly accounts for transient faults, process variations, and the TDP constraint in multi-/many-core chips. It synergistically exploits different reliability techniques, software hardening modes, and V-f levels at run time for power–reliability management. The problem was modeled as a constrained 0–1 integer linear program (ILP), and a computationally lightweight yet efficient heuristic-based technique for solving the problem was proposed. Results have shown that compared to an ILP solver tool, PAFT deviates far less than one order of magnitude in terms of reliability efficiency while seeding up the reliability management decision time by a factor of up to 1680. PAFT also provides at least one order of magnitude reliability improvement under different TDP constraints when compared to the systems that use either hardware reliability techniques or software hardening modes while increasing the execution time less than 3%.

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. Brooks, D.M., Dick, R.P., Joseph, R., Shang, L.: Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro* **27**(3), 49–62 (2007). <https://doi.org/10.1109/MM.2007.58>
2. Gaisler, C.: LEON3 processor. <http://www.gaisler.com/index.php/products/processors/leon3>. Accessed 07 June 2019
3. 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>
4. 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, IEEE International Workshop, WWC’01*, pp. 3–14. IEEE Computer Society, Washington (2001). <https://doi.org/10.1109/WWC.2001.15>

5. 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>
6. Henkel, J., Bauer, L., Zhang, H., Rehman, S., Shafique, M.: Multi-layer dependability: from microarchitecture to application level. In: The 51st Annual Design Automation Conference 2014, DAC'14, San Francisco, June 1–5, 2014, pp. 47:1–47:6 (2014). <https://doi.org/10.1145/2593069.2596683>
7. Rangan, K.K., Powell, M.D., Wei, G., Brooks, D.M.: Achieving uniform performance and maximizing throughput in the presence of heterogeneity. In: 17th International Conference on High-Performance Computer Architecture (HPCA-17 2011), February 12–16 2011, San Antonio, pp. 3–14 (2011). <https://doi.org/10.1109/HPCA.2011.5749712>
8. 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>
9. 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>
10. Salehi, M., Ejlali, A.: A hardware platform for evaluating low-energy multiprocessor embedded systems based on COTS devices. IEEE Trans. Ind. Electron. **62**(2), 1262–1269 (2015). <https://doi.org/10.1109/TIE.2014.2352215>
11. Salehi, M., Shafique, M., Kriebel, F., Rehman, S., Tavana, M.K., Ejlali, A., Henkel, J.: dsRelim: power-constrained reliability management in dark-silicon many-core chips under process variations. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015, Amsterdam, October 4–9, 2015, pp. 75–82 (2015). <https://doi.org/10.1109/CODESISSS.2015.7331370>
12. Salehi, M., Tavana, M.K., Rehman, S., Kriebel, F., Shafique, M., Ejlali, A., Henkel, J.: DRVS: power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In: IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2015, Rome, July 22–24, 2015, pp. 225–230 (2015). <https://doi.org/10.1109/ISLPED.2015.7273518>
13. Salehi, M., Ejlali, A., Al-Hashimi, B.M.: Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. IEEE Trans. Parallel Distrib. Syst. **27**(5), 1497–1510 (2016). <https://doi.org/10.1109/TPDS.2015.2444402>
14. Salehi, M., Tavana, M.K., Rehman, S., Shafique, M., Ejlali, A., Henkel, J.: Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems. IEEE Trans. VLSI Syst. **24**(7), 2426–2437 (2016). <https://doi.org/10.1109/TVLSI.2015.2512839>
15. Salehi, M., Ejlali, A., Shafique, M.: Run-time adaptive power-aware reliability management for manycores. IEEE Des. Test **35**(5), 36–44 (2018). <https://doi.org/10.1109/MDAT.2017.2775738>
16. Shafique, M., Garg, S., Henkel, J., Marculescu, D.: The EDA challenges in the dark silicon era: temperature, reliability, and variability perspectives. In: The 51st Annual Design Automation Conference 2014, DAC'14, San Francisco, June 1–5, 2014, pp. 185:1–185:6 (2014). <https://doi.org/10.1145/2593069.2593229>
17. Shye, A., Moseley, T., Reddi, V.J., Blomstedt, J., Connors, D.A.: Using process-level redundancy to exploit multiple cores for transient fault tolerance. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25–28 June 2007, Edinburgh, pp. 297–306 (2007). <https://doi.org/10.1109/DSN.2007.98>

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.