Reliable On-Chip Systems in the Nano-Era: Lessons Learnt and Future Trends
Jörg Henkel†*, Lars Bauer†, Nikil Dutt‡, Puneet Gupta†, Sani Nassif§, Muhammad Shaﬁque∗, Mehdi Tahoori∗, Norbert Wehn††

*KIT Karlsruhe, †UCI Irvine, ‡UCLA Los Angeles, §IBM Austin, ††Uni Kaiserslautern

ABSTRACT
Reliability concerns due to technology scaling have been a major focus of researchers and designers for several technology nodes. Therefore, many new techniques for enhancing and optimizing reliability have emerged particularly within the last five to ten years. This perspective paper introduces the most prominent reliability concerns from today’s points of view and roughly recapitulates the progress in the community so far. The focus of this paper is on perspective trends from the industrial as well as academic points of view that suggest a way for coping with reliability challenges in upcoming technology nodes.

1. INTRODUCTION
Around a decade ago research increased focus on reliability for on-chip design with the move to nanoscale. At that time, technology roadmaps provided evidence that upcoming technology nodes would heavily suffer from reliability problems since feature sizes would reach limits at which certain effects would seriously jeopardize the correct functionality of circuits. An example of such an effect is Random Dopant Fluctuations, whereby the number of dopant atoms in the channel of a MOSFET becomes so small (e.g., 40) that the addition or subtraction of a single atom can make a significant difference to the behavior of the device. Even though many of these atomic-level effects had been known and measurable for a long time, their impact had not reached the point where they would impact circuit functionality. An inflection point occurred around a decade ago and started then to change the mindset of researchers and designers to plan for dependability as a major design challenge in upcoming technology nodes (also referred to as the nano-CMOS era). We have seen heavy research and development efforts towards dependability since then. The contribution of this perspective paper is as follows:

• Introducing the most prominent effects (from today’s view) that seriously jeopardize reliability. This helps novices to understand the basic problems. In particular, we will introduce variability, aging and temperature effects, and soft errors (Section 1).

• Summarizing and categorizing state-of-the-art techniques at the hardware-level, software-level and application-level (without claiming comprehensiveness since that is not the main focus of this perspective paper) (Section 2).

• Providing perspectives of future trends and approaches of how to combat the inherent reliability problems. This is the main contribution of this paper as we introduce new ways to cope with these problems that go far beyond the current state-of-the-art. Some catchwords are cross-layer approaches, self-organization, run-time adaptation and more. These perspectives are a result of research in that field conducted within approximately the last ten years and they reflect views from industry as well as from academia. These perspectives may help to guide the way to cope with reliability challenges in upcoming technology nodes (Section 3).

The discussions within this paper are focused towards higher levels of abstraction to increase and optimize for reliability. Physical and device level techniques have been the primary focus a decade ago and it has been recognized that higher abstraction levels like, for example, the software stack might contribute its share to increase reliability even if the source of unreliability is the device physics due to ever shrinking feature sizes. That said, we believe that all abstraction levels should contribute its share to achieve a high degree of reliability in future technology nodes.

In summary, this paper recapitulates a decade of research in reliability and it focuses on drawing a way to future research and development in dependability from lessons learnt in the past.

1.1 Manufacturing Variability: Sources and Magnitude
Dimensional scaling has exceeded the tolerances of equipment used to manufacture semiconductor circuits, resulting in ever increasing variability in power and performance.

Figure 1: Circuit variability as predicted by ITRS [1]

Figure 2: Sleep power variability across temperature for ARM Cortex M3 processor [2]

There are two sources of hardware variation:

• Semiconductor Manufacturing. Random as well as systematic variation at all length scales (die-to-die and within-die) has become a scaling bottleneck. In addition to increasing magnitudes (see Figure 1), the nature of variability is also changing (e.g., bimodality due to double-patterning [3]). For instance, measurements on 10 off-the-shelf embedded processors (results for 5 of them are shown in Figure 2) show over 9× variation in leakage power at room temperature.

• Vendor. Parts with almost identical specifications can have substantially different power, performance or reliability characteristics, as shown in Figure 3. This variability is a concern as single vendor sourcing is difficult for large-volume systems.

Summary: Variation among parts with identical specifications is large and is expected to grow, which is a significant
NBTI is the most prominent aging effect. This is primarily due to the slow scaling of power supply voltages, which when coupled with shrinking device dimensions causes ever increasing vertical and horizontal electric fields. These high fields lead to phenomena like Hot Carriers and Negative Bias Temperature Instability both of which cause permanent degradation of devices. Of these phenomena, NBTI has become most prominent and has received widespread attention (see for example [5–7]). While the exact mechanism involved is still a topic of active research, it is believed that high fields in the gate region cause the activation of traps in the gate material which when filled create a fixed charge that changes the surface potential and in turn causes the threshold voltage to shift.

NBTI exhibits itself on two time scales. The first is a short time constant (ns regime) phenomena whereby a device under high gate voltage stress will exhibit a threshold voltage higher than normal for a short period of time, with subsequent return to normal after the stress is removed. The second is a slow and steady change in the threshold voltage over time as traps get permanently filled. The phenomena is more pronounced for P-Channel devices, and Figure 4 shows a plot of threshold voltage increase after seven years of operation for a range of technologies.

Summary: Modern devices degrade, referred to as aging. NBTI is the most prominent aging effect.

1.3 Impact of Temperature

All major aging effects exhibit a temperature dependency i.e., they stimulate and/or accelerate aging. The dependency is expressed through the Arrhenius’ Law [8]. An aging effect $\lambda_{EFF}$ has the property:

$$\lambda_{EFF} \propto e^{\frac{-E_a}{RT}}$$

where $T$ is the temperature and $k$ is Boltzmann’s constant. The activation energy $E_a$ is specific for a certain aging process. Currently (i.e. 2012 where 22 nm has been commercialized) the most critical aging effect is NBTI (Figure 5 shows the influence of temperature on NBTI) as stated before in Section 1.2. Others are: electromigration (EM), Time-Dependent Dielectric Breakdown (TDDP), and Hot Carrier Injection (HCI). EM [9] is caused by the erosion of metal interconnects through ion movement. TDDP [10] results in conductive paths due to the breakdown of the dielectric through the formation of traps caused by high electric fields, HCI [10] is caused when hot carriers in a source-drain current attain sufficient energy to be injected into to gate oxide to form traps. Apart from the property expressed in Equation (1), $\lambda_{EM}$ is also affected by the thermal gradients\(^1\) on a chip. In this case the time dependency of electromigration becomes:

$$\lambda_{EM} \propto e^{\frac{-E_a}{q \Delta T}}$$

where $\Delta T_{joule}$ is the difference in heat energy resulting from local power consumption and not from heat conducted from elsewhere in the chip.

An additional effect which is responsible for shortening chip lifetime is thermal cycling which induces stress through periodic heating and cooling, modeled through the Coffin-Manson equation [11]:

$$N = C\left(\frac{1}{\Delta T}\right)^\beta$$

where $\Delta T$ is the change in temperature, $C$ is a material constant, and $N$ represents the expected number of cycles until a failure occurs. The exponent $q$ is the experimentally determined Coffin-Manson exponent with $q \in \{1, 3\}$.

Summary: Temperature stimulates and accelerates aging. Thermal gradients (spatial and temporal) play a key role.

1.4 Soft Errors

It is a common belief that with continuous downscaling of CMOS technologies the susceptibility of memories and logic to radiation coming from atmospheric neutrons or on-chip radioactive impurities increases. This is due to the fact that the soft error rate has an exponencial relationship with the critical charge $Q_{crit}$ which is the minimum amount of charge that can flip a data value in a memory cell or in logic [12]. But the soft error also depends on the sensitive depletior area which decreases with scaling. In the past, the reduction in the critical charge caused by lower supply voltage has been more than offset by the reduction of the sensitive area and by technology improvement. Recent investigations have shown that the trend does not longer appear to be true, at least for SRAMs below 40 nm [13]. In general, SRAMs are more vulnerable to soft errors than logic, since memory cells lack transient masking mechanisms and they are much more dense. This density makes SRAM cells much more susceptible to process induced transistor variability which strongly impacts $Q_{crit}$. Moreover, recent measurements have shown the importance of Multiple bit/multiple Cell Upsets (MCU) which results in an increased MCU occurrence to the total number of upsets [14].

But despite the fact that the soft error rate for a single memory cell or latch decreased, the capacity on a chip increased faster than the soft error rate change. For example, the transition from the 130 nm to the 65 nm technology node reduced the soft error rate by about a factor of 2 in SRAMs [14]. But at the same time memory capacity increased faster resulting in an increase of the system soft error rate. This is depicted in Table 1 taken from [13] which shows the single event upset (SEU) rate per microprocessor in various technologies. Beyond that, power efficiency forces designers to reduce the voltage via sophisticated techniques like dynamic voltage scaling or near subthreshold voltage which decreases $Q_{crit}$ and consequently increases the soft error rate.

Summary: Soft errors reverse a long term trend and show an increase. Furthermore, multi-cell upsets become much more frequent.

2. RESEARCH IN RELIABILITY

Since reliability has been recognized to become a major on-chip design challenge around a decade ago, heavy research efforts started at various levels of abstraction. In this section we are providing a glimpse of reliability-enhancing
2.1 Hardware-Level Mitigation

The existing hardware-level techniques range from hardware-redundancy (like TMR/DMR [15–18]) to pipeline protection with shadow latches [19] and designs with reduced architectural vulnerability factor [20]. Logic duplication and TMR have been used in commercial systems such as Tandem’s NonStop system [21]. Various sequential elements with circuit-level hardening, e.g., DICE [22], RCC [23], BISE [24], BCDMR [25], and selective protection [26] provide different degrees of error resilience with varying costs.

Some architecture level techniques such as self-checking designs [27] are cost-effective solutions, in which the protected circuit generates outputs encoded in an error detecting code (parity, Berger, or arithmetic code) and the circuit outputs are controlled by a checker circuit to detect and maybe correct possible errors [28]. Residue codes have been used for arithmetic circuits such as multipliers [29].

Another group of techniques used to protect memories, buses, or other microprocessor array structures (e.g., register file) are parity and error correction codes [30]. Several techniques to reduce their performance penalty are presented in [31]. Built-in current sensors are an alternative approach to detect soft errors [32] that can be applied to all memories. Examples of microarchitecture and architecture level techniques include DIVA [33] and Argus [34].

Various canaries and monitoring circuits [35, 36], circuit-level guardbanding techniques [37, 38], and on-line self-test and diagnostics [39, 40] are examples of other classes of resiliency techniques targeting specific failure mechanisms, such as early life failures, voltage droop and aging.

Summary: Existing hardware solutions focus on error sensing, detection, and masking. Error recovery is typically performed at higher (e.g., software) layers. However, the impacts of reliability failure mechanisms at hardware must be better evaluated at multiple levels for more cost-effective solutions. Therefore, there is a need for cross-layer resiliency to protect only what actually matters.

2.2 OS-Level – Thermal Management

As discussed in Section 1.3, many reliability concerns like aging are stimulated and/or accelerated by temperature, making thermal management a key technique. While Dynamic Voltage and Frequency Scaling (DVFS) and gating have been widely used to reduce the power consumption, they are only capable to effectively reduce the peak temperature but they unfortunately are not suitable to combat the important spatial/temporal thermal gradients [41]. For example, in [42] the hottest core is switched off and the coldest is activated to control the chip temperature. However, this method leads to thermal cycling and thus shortens the chip lifetime. A more effective method with respect to aging is proposed in [43]. It employs a closed-loop PI controller to increase the performance and throughput of the many-core systems under thermal constraints. A further controller-based approach proposed in [44] implements extremum-seeking control to optimize temperature distribution. The obtained reduction in peak temperature is 9°C and the reduction in thermal spatial variation is from 6°C to 1°C and therefore it effectively decelerates aging.

Distributing the computational workload (i.e., tasks) through different cores can largely mitigate the consumed power density resulting in balancing both the peak temperature and the aging-related spatial/temporal thermal gradients. The approach presented in [41] uses offline Integer Linear Programming (ILP) in order to optimally scheduling the tasks to the available cores in a Multi-Processor Systems. While this approach achieves optimal thermal results, it requires the task workload to be known a priori.

Summary: To prevent and/or decelerate various aging effects, thermal management techniques tailored to reducing spatial/temporal thermal gradients are necessary. Traditional DVFS techniques are not suited for that purpose.

2.3 Software-Level Mitigation

Prominent software-level reliability techniques primarily rely redundant code execution such as in SWIFT/-CRAFT [45–47] by embedding redundant instructions, comparison instructions, and control flow checking instructions using EDDI [47–49], check pointing and replication [16]. SWIFT-R employs majority voting or AN-code for recovery. These software-based schemes duplicate all the instructions and incur a performance and/or memory overhead (more than 2×), etc. Besides these, other prominent compile time approaches are: control flow checking [47, 50, 51], register vulnerability reduction [52], wear-out and aging-aware schemes to improve the reliability [53, 54], etc.

Several compiler-level techniques have been proposed as well. State-of-the-art reliability-aware instruction scheduling approaches [52, 55–57] reorganize the instruction profile of a program at the cost of some performance degradation and some memory overhead compared to instruction redundantly
dancy techniques. The technique ISSE [55] reschedules a program’s assembly code to minimize the operators’ vulnerable periods via exploiting the slack time. Since the available slack after the performance-optimized scheduling is limited, other instruction scheduling techniques [52, 55] provide limited reliability improvements. In order to protect the control flow, basic block signatures have been introduced [58].

**Summary:** Software level techniques alleviate or complement hardware level techniques or are deployed when hardware redundancy is prohibitive. However, software-level reliability techniques do not ensure completely error-free. A trade-off toward various kinds of errors can be attributed to one of the following two categories:

- **Algorithmic resilience** is given when a certain amount of errors can be tolerated by the algorithm itself without rendering the outcome of the calculations invalid [59, 60]. This is typical of probabilistic and iterative algorithms, which can be found in a class of algorithms often subsumed under the term Recognition, Mining and Synthesis (RMS) [61] and of wireless communication systems [62, 63], one of the fundamental technologies of today’s information society. Also all algorithms that can tolerate statistical behavior, such as fixed-point DSP algorithms and numerical calculations exhibit algorithmic resilience.

- **Cognitive resilience** stems from an application’s interaction with a human observer, e.g., video and audio processing. Errors are tolerable, as long as the user cannot discern quality difference, or accepts them as a trade-off for, say increased battery life time [63].

Taking into account the application-inherent resilience at the architectural and software level clears the way for a further reduction of the overhead for resilience by providing only a sufficient hardware reliability for a given application, while still achieving the desired application reliability. Compared to lower level resilience techniques like error-detection schemes, partial hardware redundancy, or Razor [64, 65] or tunable replica circuits, techniques on higher levels are very application dependent and require a real cross-layer approach. A typical example for a static application dependent protection mechanism is selective protection. It is well known that in many signal processing algorithms the higher-order bits have a much larger influence on the overall system than the low-order bits. So, it is very often sufficient to protect only the higher-order bits [62]. Many applications exhibit a large dynamic at run-time. This dynamic is well explored to optimize power and energy by, e.g., dynamic voltage and frequency scaling. Here, application specific quality-of-service (QoS) is traded-off against power efficiency at run-time. E.g., in video processing, the compression rate can be traded-off against power saving. But dynamic application behavior can be exploited for application-specific reliability enhancements, as well. This yields adaptive reliability tuned by the application at runtime according to the required application performance, monitored disturbances and operating conditions of the underlying hardware. For this purpose, so called resilience actuators can be defined (on hardware and/or software level) which are dynamic protection mechanisms to increase the error resilience on system level [66, 67]. They exist on various abstraction levels:

- **Do nothing:** this is the case for very low error rates where no impact of the errors on the system level is observed.

- **Change hardware operating point:** e.g., changing the voltage and/or frequency operating point which makes the system more robust at the cost of increased power/energy.

- **Change algorithmic parameters:** many algorithms have parameters which can be changed at run-time. E.g., most of the algorithms in wireless communication are iterative like Turbo-Code decoding in LTE. If the communication channel has a high signal-to-noise-ratio (SNR), less iterations are required compared to a low SNR channel. So, if the channel shows a high SNR, the decoding iterations can be reduced without changing the system performance. The timing margin obtained by the reduction of the iterations can be used to increase the reliability.

- **Change algorithms:** Besides changing only algorithmic parameters we can also change the complete algorithm itself. Often, there is a choice of different algorithms, starting from optimal with high complexity down to suboptimal algorithms with very low complexity. This offers a trade-off between QoS and resilience robustness.

In general, different actuators or combinations of actuators are used to increase the resilience robustness. It is preferable to use actuators which do not require changes in the system components themselves and can be performed on software level. So changes in the algorithm or algorithmic parameters are excellent candidates for software, resulting in variability-resistant software concepts [67]. Obviously each actuator offers a different trade-off between hardware reliability, QoS and implementation performance, e.g., energy and throughput. These trade-offs have then to be characterized.

**Summary of Research in Reliability:** We have focused on some representative abstraction layers to introduce to state-of-the-art in reliability. In a more comprehensive overview, the whole stack of layers from **Device** through **Hardware Architecture, System Software, Compiler and Application** should be covered since all layers can potentially contribute to increase the reliability (for example through cross-layer approaches like in Section 3.1.)

### 3. RELIABILITY PERSPECTIVES

In this section we present our vision of the road to reliability within the next decade formulated as a set of perspectives. These perspectives are not intended to be orthogonal meaning that, for example, a method introduced in one perspective might be a subset of a more general concept presented within another perspective. For this sense, the following perspectives also do not represent any kind of categorization. The perspectives, however, do represent statements that contain promising methods and concepts from different angles: from hardware and software, and from industry and academia. The perspectives have their origin in, in parts, larger-scale ongoing projects on reliability where at least some initial lessons have been learned that are believed to guide the way to the future reliability research. It should also be mentioned that these perspectives primarily focus on methods and concepts at higher abstraction levels since this is a recent trend as opposed to a decade ago when physical and device level techniques were the exclusive focus.

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Cross-Layer Leverage</th>
<th>Run-Time Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: Cross-Layer Leverage</td>
<td>“Cross-layer approaches can leverage reliability through techniques that are pro-actively designed with respect to techniques at other layers. This opens a so far neglected but efficient means to increase reliability”</td>
<td>“To address ever-increasing spatial and temporal variations, future systems will inherently sense and adapt”</td>
</tr>
</tbody>
</table>

3This by no means implies that lower techniques, like device, in future will have lower relevance.
P3: Run-Time Adaptation and Self-Organization:
“In order to achieve error resiliency in complex future many core systems, run-time adaptation and concepts/means of self-organization are needed”.

P4: Exploiting Application Resilience:
“Applications’ inherent resilience paves the path for cost-efficient error resilience; however, it requires fine-grained control on reliability methods”.

P5: CAD for Reliability:
“Reliability needs to be considered as the major constraint in an integrated CAD flow”.

P6: Scaling and Cost for Resilience:
“The cost for resiliency might become non-affordable in future technology nodes”.

P7: Reliability for CPSoCs:
“Future of complex chips will evolve into Cyber-physical Systems-on-Chip (CPSoCs) - this requires seamless integration of optimizations for reliability, safety, and security related issues”.

In the following we elaborate on these perspectives in more depth and provide –in parts– some early evidence supporting our perspectives.

3.1 P1 – Cross-Layer Leverage
A robust and dependable on-chip design needs to consider reliability at all levels of abstraction in order to leverage on the opportunity that techniques designed for one level have knowledge of other levels’ techniques. As opposed to early work a decade ago, it is not sufficient anymore to confine all reliability concerns at the hardware layers, as faults ultimately propagate to the software layer, and in between there are several sources of fault masking that may contribute towards cost-efficient cross-layer reliability solutions. Since each system layer comes with its own reliability tradeoffs, it is crucial to leverage multiple system layers in an integrated and collaborative fashion for joint optimization.

Key prerequisites in enabling cross-layer resiliency are cross-layer modeling of various reliability effects and cross-layer optimizations in order to be able to consider failure generation and failure propagation through multiple layers from Device all the way up to the Application layer.

3.1.1 Cross-Layer Reliability Modeling & Analysis
In the following, we discuss cross-layer software error analysis which is used to develop accurate software program-level reliability and resilience models.

CLASS: Cross-layer Soft Error Rate Analysis: Existing soft error rate (SER) analysis techniques mostly target one abstraction level such as hardware (e.g., 68-74). Cross-level techniques consider logic, electrical and timing window masking, and therefore suitable for irregular (random logic such as functional units and control logic) structures. Architecture and micro-architecture level methods consider only Architectural Vulnerability Factor (AVF) for regular (e.g., register files, caches, memories) structures. Hence existing methods fail to provide an accurate system-level analysis considering circuit to architecture level masking at various layers, and lead to an unacceptable inaccuracy.

Figure 6: Conceptional Block Diagram of CLASS

Combined Logic and Architectural Soft error Sensitivity analysis (CLASS) is a cross-layer approach for computing the soft error vulnerability of the entire (microprocessor) system. This hybrid approach uses a circuit-level SER analysis to model the propagation of errors in combinational and sequential logic blocks, and a Memory Architecturally Correct Execution (MACE) analysis to compute the vulnerability of memories, including the scenarios in which an erroneous value is written in the memory hierarchy and several cycles later either masked or used by the program, i.e. Memory Vulnerability Factor (MVF). The main idea is perform profiling of application at the (micro-)architecture levels and extracting all memory accesses and architectural masking (MACE analysis), and then performing circuit- and logic-level analysis with those annotations. The conceptional block diagram of this approach is shown in Figure 6.

Evaluation of the OR1200 processor using the CLASS approach is five orders of magnitude faster than statistical fault injection while its inaccuracy is less than 7%. This is more than five times more accurate than single-layer analysis either at logic or architecture levels. This cross-layer soft-error modeling framework can be combined with the cross-layer soft error mitigation framework, presented next.

To employ efficient software-level reliability methods, the challenge is to bridge the gap between the hardware and software by quantifying the effects of hardware-level faults at the instruction level, while considering the knowledge of low-level fault models (as discussed above) and processor layout. Moreover, it is important to understand which instructions –when fail– lead to which type of errors in the software program. Researchers like in [75] have developed various software program reliability models to quantify three reliability-related properties namely vulnerability, masking, and resilience of application software at different granularities (like instruction, basic block, and function-level):

- Instruction Vulnerability Index (IVI) quantifies the effects of faults in different processor components (spatial) during the execution of different instructions over time (temporal), types of errors, critical and non-critical instructions, and vulnerable bit analysis.
- Instruction-Level Error Masking Index quantifies the error masking probabilities at instruction level depending upon the data- and control flow properties of a program.
- Function Resilience that provides the probabilistic measure of functional correctness (output quality) of a program in the presence of failures.

3.1.2 Cross-Layer Reliability: Code Generation and Execution
Code generation by means of reliability-driven compilation should entail:

- Reliability-Aware Software Transformations like: Various software transformations i.e. reliability-guided loop unrolling, data type optimization, on-fly table computations, for (a) reducing the IVI; and (b) lowering the probabilities of program failures.
- Reliability-Aware Instruction Scheduling should target at improving the reliability of a software program given a user-provided tolerable performance overhead.

To increase reliability through code generation, the flexibility of compilers to generate multiple versions of functions should be provided such that the one with the lowest error probability, given a certain fault model, can finally be selected for execution at run-time.

Results in Figure 7 illustrate that a reliability-driven compiler is capable to provide up to 60% lower software program failures running on an unreliable hardware.

3.1.3 Underdesigned and Opportunistic Computing
Underdesigned and Opportunistic (UnO) computing machines (see Figure 8) provide a unified way of addressing variations due to manufacturing, operating conditions, and even reliability failure mechanisms in the field: difficult-to-predict spatiotemporal variations in the underlying hard-
There are several challenges in realizing the UnO vision: (1) efficient ways to generate hardware signatures and expose variations, are fully exposed to and mitigated by multiple layers of software [78].

Underdesigned machines could use parametrically underprovisioned circuits (e.g., voltage over-scaling as in [79]) or be implemented with explicitly altered functional description (e.g., [80]). They may allow erroneous operation and rely upon applications level of tolerance to limited errors (as in [81]); correct all errors (e.g., [65]); or operate hardware within distinct but correct operation limits (e.g., [2, 82]).

Consider an UnO machine example ViPZone [83] a system-level solution (see Figure 9) that opportunistically exploits DRAM power variation through physical address zoning in the operating system (OS). ViPZone is composed of a variability-aware software stack that allows developers to indicate to the OS the expected dominant usage patterns (write or read) as well as level of utilization (high, medium, or low) through high-level APIs. ViPZone’s variability-aware page allocator, implemented in the Linux kernel, is responsible for interpreting high-level requests for memory and transparently mapping them to physical address zones with different power consumptions. Experimental results across various configurations running PARSEC workloads show an average of 13.1% memory power savings.

There are several challenges in realizing the UnO vision: (1) efficient ways to generate hardware signatures and exposing them to software; (2) identifying effective, non-intrusive methods across the software stack to leverage monitored variations, and (3) taking advantage of software adaptability in hardware design flows.

Summary: Cross-layer approaches to leverage on reliability techniques applied throughout various layers from Device all the way up to the Application layer are key as they are indispensable for efficiently handling reliability in future on-chip systems.

3.2 P2 – Run-Time Sense

Adaptive mechanisms in hardware and/or software can optimize the trade off between errors, energy and performance based on the feedback from fabrication-time or run-time circuit performance/power/error monitors. Hardware monitoring can be done in one of the following ways.

- Production Test. Signatures can be measured and stored (to make them software accessible) using traditional test methods.
- Replica Monitors. Structures that are generally “decoupled” from the functional design (but can be aware of it) are inserted to capture hardware characteristics of interest (e.g., achievable frequency [84], leakage power [85] or aging [86]).
- In-situ Monitors. Replica monitors though cheap and non-intrusive cannot predict local variations. Structures that are embedded in the circuit can get better accuracy (e.g., delay fault detection flip-flop designs [64] or delay detection probes [87]).
- Online Self-Test. On-line self-test and diagnostics techniques allow a system to test itself concurrently during normal operation with little downtime visible to the end user (e.g., [88]).
- Software-Based Inference. A compiler can use the functional specification to automate the insertion (in each component) software-implemented test operations to dynamically probe and determine which parts are working (e.g., [89]).

As an example, consider two alternative approaches to performance monitoring. [84] proposes a methodology to automatically synthesize multiple design-dependent ring oscillators as smart canary structures which can reliably predict achievable chip frequency but with margins for local variations. The key idea is to identify clusters of critical paths which behave similarly with variation and construct a canary structure per cluster. Simulation results for 45 nm benchmarks show required margins decrease as number of monitors (e.g., number of clusters) increase from 1 to 12. Early silicon measurements indicate that multiple design-dependent ring oscillators can reduce needed delay margin beyond the predicted performance significantly (~25%) compared to generic inverter based ROs. To further improve the prediction (albeit at a higher overhead), [87] proposes a novel in situ timing slack monitoring methodology, namely SlackProbe, which monitors in situ timing slacks of carefully selected circuit nets (i.e., including path endpoints as well as internal nodes). By shifting the monitoring points to the internal nodes, the slack monitors can cover more critical paths with the cost of some additional timing margin, thereby reducing monitoring overheads by over 10X compared to monitoring all timing endpoints.

Summary: In future we will see an increased number of on-chip sensors. Designing and managing a hierarchy of heterogeneous sensors to get accuracy with minimum overheads is important. Systems will integrate a variety of fabrication-time test and runtime monitors to enable adaptation. Challenges lie in designing low-overhead monitoring schemes and exploring their tradeoffs with design margining approaches.
3.3 P3 – Run-Time Adaptation and Self-Organization

Run-time adaption will be another key component to increase the reliability of future on-chip systems. If only design-time techniques would be employed the designers would have to be more pessimistic as they would have to reserve more and more resources in order to provide some infrastructure to make up for any kind of failure of components that exhibit signs of unreliability only after the system has started operating (due to, for example, aging and temperature variations, see also Section 1). But this, in turn, would render future on-chip systems infeasible from a cost point of view (see also Section 3.6) since the inherent unreliability will further increase with future technology nodes.

A good example for why run-time adaption is necessary is the thermal problem that we will use for the rest of this section. Temperature increase when computational activity is increased. This is due to the distribution of computational workloads that can hardly be predicted at design-time: as a consequence, power densities (power per area) go up and in turn increase temperature and temperature gradients. Run-time adaption techniques can then be employed to balance the workload throughout the many-core system and thus decreasing peak temperatures and gradients.

Since future on-chip system not only exhibit a higher and unpredictable behavior in terms of reliability but also become more complex – many cores system with hundreds or even thousands of cores – the question arises of how to facilitate efficient and low-overhead run-time variations due to workload-dependent voltage and temperature thresholds. Our perspective is that concepts of self-organization will be needed for that purpose. A method for providing a scalable light-weight infrastructure for making distributed decisions is to employ agent-based systems. These allow a self-adaptive and decentralized approach that can handle global complexity by pairing local decisions with the cooperation among fellow agents thereby achieving scalability.

A fully-distributed agent-based approach is presented in [90] where each core has exactly one local agent. Each agent negotiates with its neighboring agent, i.e., the agents of adjacent cores, in order to distribute a global power budget based on performance constraints and measured temperatures. Through multiple iterations the budget can be propagated over larger regions of the many-core system with the goal of balancing temperature and thereby reducing peak temperatures and temperature gradients. While scalability is key tocope with complexity, a centralized approach with unlimited resources will still result in a higher quality thermal management solution due to its global view of the system. This is illustrated in Figure 10 where both the central reactive approach – i.e., one where thermal management is performed when a specified temperature threshold is reached – and the central proactive approach – i.e., one where future temperatures are predicted and thermal management is performed before a threshold is reached – obtain lower peak temperatures than the fully-distributed approach.

Summary: Run-time adaption is indispensable for managing reliability and self-organization is a promising means to get there since the complexity of future systems grows rapidly with hundreds or even thousand cores on a chip.

3.4 P4 – Exploiting Application Resilience

The importance of application-specific reliability was already identified in Section 2.4. There are many applications, i.e., all applications dealing with imprecise or incomplete information have an inherent error resilience, which can pave the path for cost-efficient error resilience. However, fine-grained reliability methods are inevitable. The inherent error-resilience of these applications is usually limited to the data processing. Therefore, the control flow is not error resilient. Furthermore, often the application resilience is dynamic and depends, e.g., on statistical properties of data content or environmental conditions. So, fine-grained reliability mechanisms are mandatory that can be activated and deactivated and application information has to be forwarded to different layers to guide the reliability.

Wireless application is a good example to illustrate applications’ inherent resilience. Communication systems are designed to recover the originally transmitted data sequence in spite of errors that are induced during transmission over a noisy wireless channel. These channel errors are corrected by advanced forward error corrections techniques in today’s communication systems. Errors induced by hardware can now be considered as yet another error source in communication systems and can be treated similarly to channel errors, i.e., the algorithms used for correcting channel errors can partially be re-used to recover hardware errors. Under bad channel conditions, i.e., large channel noise, hardware errors with low error rates are not at all visible at the system since the overall system behavior is dominated by the noisy channel errors. Figure 11 taken from [62] shows the system behavior of a WiMAX decoder: Frame error rate (FER) is traced over channel noise (\(E_b/N_0\)). The dashed blue graph shows the correct behavior of the decoder without hardware errors. Such behavior is specified in the WiMAX standard. The red graph shows the decoder’s behavior when hardware errors are injected in the memories (RAMs), the communication network (NW) and the functional units (CFUs). Obviously the decoder does not work at all. Fine-grained reliability mechanisms were used for the various building blocks exploiting the probabilistic behavior of the channel decoding algorithms to make the decoder error-resilient. Techniques like triplication with majority voting for the control parts, sign bit protection only for calculation etc. were applied. The green graph shows the behavior of the error-resilient decoder. The small deviation from the correct behavior is within quantization noise and the estimation error of the channel condition. The area overhead for the error resilient decoder is only about 20%. If the MTBF is quite high (purple graph), protection of the controller only is sufficient yielding an overhead of about 5%.

Summary: Application-specific reliability should be exploited whenever possible. However it requires profound application knowledge, fine-grained reliability methods and resilience propagation throughout different layers.

3.5 P5 – CAD for Reliability

Reliability needs to be considered as a major constraint in the CAD flow. As an example, here we present a timing analysis flow for consideration of various runtime variation effects. In the nanometer era, short-term and long-term runtime variations due to workload-dependent voltage and temperature variations as well as transistor aging introduce remarkable uncertainty and unpredictability to nanoscale VLSI designs. BTTI gradually shifts the threshold voltage of the transistors [92], thereby increases the circuit delay over the time (long-term effects). BTTI-induced timing degradation highly depends on operating context parameters including supply
3.6 P6 – Scaling and Cost for Resilience

We have observed that scaling to future technologies introduces a variety of reliability and resilience issues that have to be dealt with (see Section 1); and we have proposed that these issues must be dealt with in a cross-layer fashion (see Section 3.1) where other levels of design, e.g., the circuit, operating system and application levels, become involved in correcting resilience faults. It is clear that moving to more advanced nodes brings both opportunity as well as cost, and one must ensure that the net result is positive in order to make such a move viable. In this section, we propose a simple qualitative model for the impact and cost of resilience to put this issue into perspective.

In creating this simple cost model, we will make the following assumptions:

- The size and cost per wafer are constant. This is not a very accurate assumption since advanced technologies generally require additional manufacturing steps (e.g., double patterning), but it is optimistic in the sense that it makes the best possible case for scaling. We will denote the cost per wafer by $C_W$ and the size (area) of the wafer by $A_W$.
- The price of our product chip is constant. Meaning we are only making the choice as to which technology to make the chip in. We will denote the price per chip by $P_C$, and the area of the chip by $A_C$.
- The cost and area of design are proportional to the number of transistor. More transistors require more area and more design resources, and we assume that –for a given basic architecture– the two are linearly related.

The number of chips per wafer is $\frac{A_C}{A_W}$. The number of working chips per wafer incorporates the yield $Y$ which is assumed to be an increasing function of volume. The cost of manufacturing a working chip can thus be expressed as:

$$C_{\text{chip}} = \frac{C_W A_C}{A_W Y}.$$  \hspace{1cm} (4)

We denote the cost of designing the chip by $C_D$. If we make $N$ chips, the profit per chip at a given technology node will be:

$$P_C - C_C - \frac{C_D}{N}.$$  \hspace{1cm} (5)

Let us consider how the profit per chip will change if we were to use a more advanced technology node. A more advanced node would scale the area by a factor of $S$. But a more advanced node may require the addition of on-chip resources to handle resilience (an example might be the addition of parity protection or on-chip monitors). Assume the more advanced node requires an increase in device count by a factor $R$.

The area of the chip in this new technology node will be $A_C RS$, thus the new profit per chip would be:

$$\text{profit} = P_C - C_C RS - R \frac{C_D}{N}.$$  \hspace{1cm} (6)

Summary: From this result we can observe that:

1. Large volume chips benefit from scaling only if the cost of resilience is smaller than the shrink factor.
2. Small volume chips cannot afford as much resilience as high volume chips.
3. Reducing the cost of designing-in resilience (e.g., by creating automatic flows that can do this with little designer interaction) can make a significant difference for low volume chips.

3.7 P7 – Reliability for CPSoCs

The future of complex chips will evolve into so-called Cyber-physical Systems-on-Chip (CPSoCs). These CPSoCs will be sensor/actuator-rich, with multiple layers of hardware and software self-adaptive loops, comprising sense-de-
cide-actuate intelligence embedded into the fabric to manage a wide range of application-level, environmental, and platform (manufacturing) variability. The role of hardware-software virtualization and fusion becomes very important in reasoning about, managing, and effectively achieving specific design goals, be they individual or combinations of metrics such as energy, reliability, security, thermal, etc.

Summary: In future we will see entire Cyber-Physical Systems integrated on a chip. This will increase complexity and pose additional constraints to reliability.

4. CONCLUSIONS

Reliability turns into the major design constraint for on-chip systems as scaling moves on. After almost a decade of research, the problems are far from being solved. Whereas early reliability-enabling/increasing techniques almost exclusively focused on physical and device level, we have observed that higher abstraction levels (e.g., all the way up to the application software) have and increasingly will contribute its share to enhance reliability.

The first part of the paper gave an overview of the currently most prominent reliability concerns like variability, aging, temperature effects and soft errors. The second part provided (a certainly not comprehensive) state-of-the-art of techniques at hardware-level, software-level and application-level. The third part provided perspectives obtained through larger-scale ongoing projects on reliability where the authors from industry and academia are involved.

Our perspectives reach from already practiced techniques like exploiting cross-layer techniques to visionaries ones like future on-chip systems will evolve into complex Cyber-Physical Systems-on-Chip.

Though our seven perspectives are neither orthogonal nor comprehensive, we believe that they will cover a large part of the challenges that we are going to face within the next decade of technology scaling.

5. ACKNOWLEDGMENTS

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500 – http://spp1500.iti.tu-chemnitz.de) and the NSF Variability Expedition (Grants CCF-1029030, CCF-1029783).

References
