FP7-ICT-2013-10
HARPA
Harnessing Performance Variability

Deliverable	D3.1 Technological Context and State-of-the-Art
Work Package #	WP3
Lead Partner	University of Cyprus (UCY)
Author(s)	Giorgos Klokkaris, Panagiota Nikolaou, Kypros Chrysanthou, Zacharias Hadjilambrou, Panagiotis Englezakis, Marios Kleanthous, Chrysostomos Nicopoulos, Yiannakis Sazeides
Nature	Report
Dissemination Level	PU: Public
Delivery Date	30/04/2014

Abstract

In this deliverable, we identify the challenges in providing performance dependability guarantees in future computing systems, due to technological trends, physical phenomena, micro architectural features, and operating system traits. These include: (i) a slowdown of voltage scaling and frequency, (ii) fixed power envelope, (iii) soft-errors, (iv) a shift to probabilistic design and less reliable silicon primitives due to static and dynamic variations, (v) sharing of resources among concurrently executing threads, and (vi) performance-dependability-agnostic system management.

We elucidate the manifestations of these trends and phenomena into specific challenges for scalable performance dependability, such as (i) various types of safety margins and overheads during the design and operation, which limit performance benefits from area scaling, (ii) performance non-determinism (in terms of both timing and energy), (iii) accelerated wear-out and shorter lifetime, (iv) vulnerability to transient-errors, and (v) compromise of functional correctness, but also parametric reliability (such as exceeding the leakage energy limits).

We aggregate technology forecasts (e.g., from ITRS) for various trends, rates and distributions, such as the scaling of various safety margins with smaller feature size, models proposed in the literature to capture the behavior of various physical phenomena (e.g., rate of device degradation as a function of time and temperature), and run-time micro-architectural and operating system behavior. These predictions and models will serve as the foundations for deciding and designing the type of monitors and knobs that are essential to the HARPA engine to guarantee performance dependability.

We perform a comprehensive investigation of the state-of-the-art in monitors and knobs found in real products and in the research literature. Monitors enable the observation of physical, micro-architectural, and operating system phenomena that can hinder performance dependability. Examples of various types of monitors proposed or current in production are: performance counters, timing-violation detectors, wear-out detectors, temperature sensors, identification of the memory-address range of specific application data, and error-detection circuits. On the other hand, knobs enable the direct or indirect control of various phenomena that influence the delivery of performance dependability guarantees. Memory error-correction codes and controlling the voltage and frequency are classic examples of knobs.

A vast amount of monitors and knobs already exist, or have been proposed in the literature. In order to provide meaningful information, we categorize the various techniques into three fundamental dimensions. Thus, we assign each monitor and knob into one of these three categories: (i) error recovery techniques (which are further broken down into transient faults, permanent faults, and time-dependent variations), (ii) power management, and (iii) performance management. We also try – wherever possible – to label each of these techniques based on (a) the market segment being targeted (e.g., HPC/embedded), (b) the layer in the compute stack they are applied to (e.g., circuit, micro-architecture, software/OS, etc), (c) whether it is an academic/research concept, or whether it is already implemented in real products, (d) whether the mechanism is pro-active or re-active, and (e) whether the target is parametric, or time-dependent variations. In addition to our detailed exploration, we also summarize all investigated techniques (based on the above categorization) in the Appendix of this report.

Finally, we conclude this task by describing the motivation and the main goal of the HARPA project and we propose the monitors and knobs that best suit the context of HARPA. Important characteristics of the monitors and knobs we are seeking are (where applicable) interface, precision, granularity (in terms of area covered and how many are needed/used), minimum sampling interval, response latency, lifetime, design and run-time overheads, computing layer used, and dependence on environmental conditions. It is important to note that these monitors and knobs may be applicable to the HPC domain, the embedded-system domain, or both.

Contents

Part I
Technological Trends that Influence Delivery of Performance Dependability

1 Introduction

Modern systems have grown to the scale where thousands of processors are employed and coordinated in order to accomplish complex tasks for a large amount of users. At the same time, application-specific embedded systems have become ubiquitous in every-day life, and they constitute key components in devices ranging from electrical appliances, to automotive/aerospace/industrial systems, to biomedical components. Unfortunately, application requirements, as well as power constraints and performance guarantees, have become the primary limiting factors in the development of both large-scale, high-performance systems and resource-constrained embedded devices. Interestingly, however, the two seemingly divergent domains of High-Performance Computing (HPC) and embedded systems have both tackled these limiting factors within their own context and environment. One of the objectives of the HARPA project is to try to cross-fertilize these two domains and investigate solutions that can benefit both sectors. One of the main challenges towards such a development is the growing susceptibility to time-dependent variations in silicon devices and wires. There is a consensus that increasing guard-bands to battle variations is not scalable, due to the large worst-case cost impact for technology nodes around 10 nm. The role of HARPA is to bring together solutions that enable next-generation of embedded and high-performance heterogeneous many-core processors to cost-effectively confront variations and yet provide Dependable-Performance: correct functionality and timing guarantees throughout the nominal lifetime of a system under thermal, power, and energy constraints. In order for this to be achieved, the behavior of the system has to be monitored. Moreover, the system has to be equipped with some knobs for it to be adjustable.

The goal of this part is to first identify the trends that motivate the need for providing performance dependability guarantees in future computing systems, due to technological trends, physical phenomena, micro-architectural features, operating system and application traits. Then, we explore the manifestation of these trends and phenomena to specific challenges for scalable performance dependability by aggregating technology forecasts for various trends, rates and distributions.

2 Terminology and metrics

This section defines commonly used terms and metrics that are used throughout this report. These terms and metrics refer to all three aspects (reliability, performance and power) found in this deliverable. The terms are also used in the categorization adopted to organize the research domain of monitors and knobs. This categorization includes the layer in the computational stack, the market segment (HPC, embedded), whether a technique is proactive or reactive and whether a technique is targeting parametric or time variations.

2.1 Faults, errors, and failures

Fault is an incorrect state of hardware or software resulting from physical defect, design flaw, or operator error. Faults can be introduced during system design, during manufacturing, or they may occur during operation due to physical phenomena or operator error. An **error** is the manifestation of a fault. **Failure** is the system-level effect of an error that is visible to the user. Giving an example, a fault can be a physical phenomenon that affects a transistor in the memory, an error can be a bit-flip in a memory value resulting from the transistor fault, and failure can be the computation that uses this incorrect memory value and provides an incorrect result to the user.

Near-Term 2013-2020	Summary of issues
Reliability due to material, process, and structural changes, and novel applications.	TDDB, NBTI, PBTI, HCI, RTN in scaled and non-planar devices. Gate-to-contact breakdown. Increasing statistical variation of intrinsic failure mechanisms in scaled and non-planar devices. 3D interconnect reliability challenges. Reduced reliability margins drive need for improved understanding of reliability at circuit-level. Reliability of embedded electronics in extreme or critical environments (medical, automotive, grid, ...)
Long-Term 2021-2028	**Summary of issues**
Reliability of novel devices, structures, and materials.	Understand and control the failure mechanisms associated with new materials and structures for both transistors and interconnect. Shift to system-level reliability perspective with unreliable devices. Muon-induced soft error rate.

Table 1: Short-term and long-term reliability challenges [8].

Technology	Inverter ζ	Latch ζ	SRAM ζ
45nm	≈ 0	≈ 0	6.1e-13
32nm	≈ 0	1.8e-44	7.3e-09
22nm	≈ 0	5.5e-18	1.3e-06
16nm	2.4e-58	5.4e-10	5.5e-05
12nm	1.2e-39	3.6e-07	2.6e-04

Table 2: Predicted failure probability (pfail) for different types of circuits at different technology nodes [154].

2.2 Fault duration

Depending on their duration, faults can be categorized into **transient(soft)**, **intermittent**, and **permanent(hard)** faults. **Transient faults** occur randomly and are not an indication that the device is permanently damaged. They can cause, for example, alteration in the values stored in memory causing the system to read corrupted data. **Intermittent faults** have duration but are not permanent. They can occur and recur. Intermittent faults occur only under specific conditions (e.g., elevated temperature). Finally, **permanent faults**, or hard faults, occur and remain in the system for ever. They can only be repaired by replacing the faulty device.

2.3 Sources of faults

To better understand faults, we will discuss their main sources, based on the categories described in Section 2.2. **Transient faults** [252, 151] can occur from various sources. The first source is cosmic radiation [251]. More specifically, transient faults can occur from the high-energy particles that are produced when cosmic rays interact with the earth's atmosphere. The second source of transient faults are alpha particles [144]. Alpha particles are produced by the natural decay of radioactive isotopes. Another source of transient faults is electromagnetic interference and voltage droops. Transient faults are mainly a concern in memory cells and latches, and less of a concern in combinational logic.

Permanent-fault sources can be divided into two categories. The first category is the physical wear-out. Wear-out can occur from different sources [212], like electromigration [49, 135], stress migration, time-dependent dielectric breakdown (TDDB) [148], thermal cycles, hot-carrier injection (HCI), and negative/positive bias temperature instability (NBTI/PBTI) [189]. There are several techniques that contribute to wear-out avoidance, thus extending the lifetime of a system. The second category of the sources of permanent errors are the fabrication defects and limitations in testing. Defects on the chips might appear during the manufacturing process. These defects may be detected during the testing phase. In that case, the defected chip is discarded, or configured, in order for it to remain functional (lower frequency). However, defects are not always detected. This is due to the complexity of the chips and the extremely large test space that cannot be explored in full. Consequently, some fabrication faults may only be discovered when chips are deployed in the field.

The sources of faults can also be divided in **static** and **dynamic** variations. **Static variations** [32, 37, 34, 22] appear due to parametric variations caused by imperfections during the manufacturing process. This leads to devices with different physical and operational characteristics than their design specifications. The

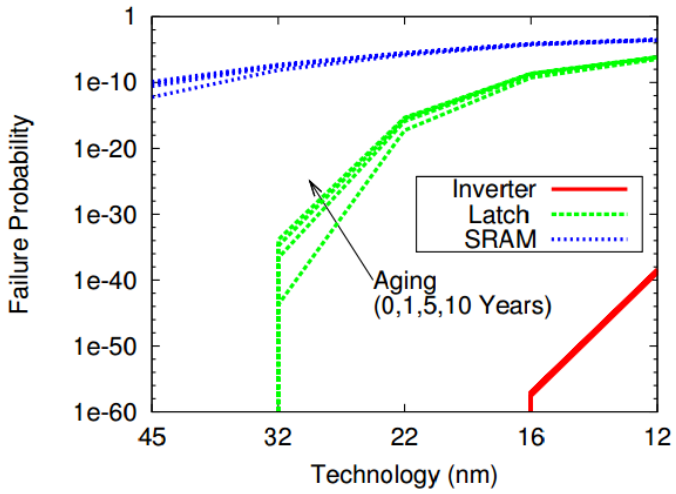

Figure 1: Impact of NBTI on failure probability trends [154].

differences can be observed in the length, width, oxide thickness, and doping of a transistor. Parametric variations can also make devices unreliable at low voltage, thus increasing the minimum voltage required for reliable operation (Vccmin) [239]. This results in a limited amount of voltage scaling in the system. Thus, reliability mechanisms have been proposed to allow voltage scaling to extend below Vccmin. On the other hand though, operating below Vccmin increases failure rates. Another category of faults are distributed workload-dependent parametric transient errors due to degradation (aging) and noise (supply noise e.g.) [82, 180, 111, 183].

Dynamic variations [225, 36], or time-dependent variations, occur during operation and include voltage droops, temperature variations, cross-coupling capacitance, multiple input switches, and path delay degradation due to aging.

2.4 Fault/error masking, fault avoidance, fault tolerance, error detection, error correction, and failure rates

As described above, a fault can manifest as an error and an error can manifest as a failure. It is well known though that not all faults manifest as errors and similarly not all errors manifest as failures. This is referred to as **fault and error masking**. Error masking can, for example, happen in memory. A corrupted value is overwritten so the error does not exist any more. **Fault tolerance** is the process of maintaining functionality in the presence of faults. The most important aspects of fault tolerance are **error detection** and **error correction**. Error detection and correction are two intertwined terms. When an error is detected, the processor must take action to mask its effects from the software. This action can be dynamic correction or self-repair (use spare components to replace the faulty one). Error detection and correction can be performed at different granularities (memories, cores, circuit). On the other hand, **fault avoidance** strives to maintain functionality by preventing the occurrence of faults.

2.5 Reactive and Proactive

To offer performance dependability guarantees in the presence of time-dependent variations and aging throughout the lifetime of the system, we need to utilize both **proactive** and **reactive** techniques. Proactive techniques are used in the absence of hard failures, while reactive techniques are used in the presence of hard failures.

2.6 Performance and fault tolerance metrics

A wide variety of metrics is used to express performance and fault tolerance aspects. These metrics are very helpful for measuring, comparing and presenting the results and can be use as program inputs. One popular

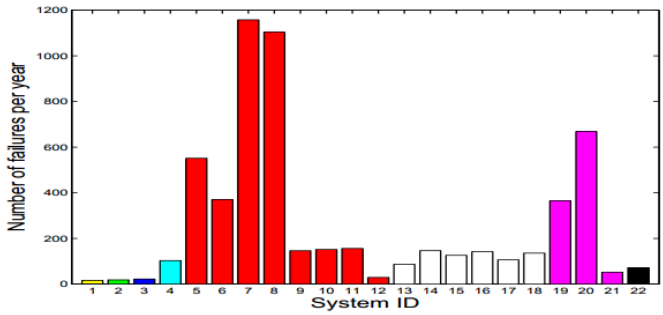

Figure 2: Average number of failures for 22 systems per year [190].

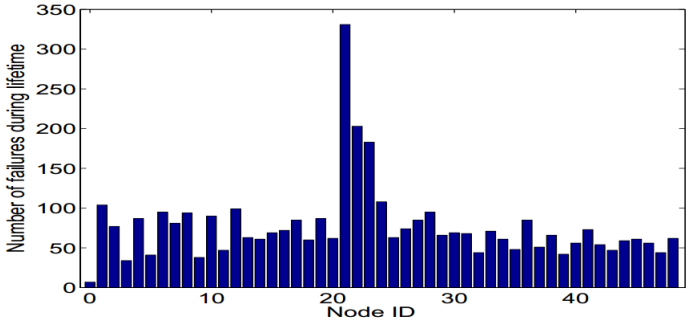

Figure 3: Number of failures per node for a system as a function of node ID [190].

fault tolerance metric is the **Failures in time (FIT) rate**, which is defined as the number of failures in 10^9 hours. **Mean time to failure (MTTF)** is often a useful metric. It represents the mean time until a failure occurs in a system. **Mean time between failures (MTBF)** is similar to MTTF, but takes into account the **Mean time to repair (MTTR)**, which refers to the time that the system is down for recovery reasons. The relationship between mean time to failure and mean time between failures is:

$$MTBF = MTTF + MTTR.$$

The **Availability** of the system is the probability of a system operating correctly at a given time. This metric is appropriate for many computing and cloud applications. The system's availability can be measured in terms of "nines" (i.e., number of nines to the right of the decimal point) with this equation:

$$Availability = \frac{MTBF}{MTBF+MTTR}$$

Another important metric is the **Architectural vulnerability factor (AVF)** [152], which defines the probability that a fault will result in a failure to a programs output. The idea behind AVF is to classify microprocessor state as either required for architecturally correct execution, or not. One other useful metric is the probability of failure (**pfail**). This metric refers to the probability of devices in a chip to fail. For instance, what is the pfail of cells in a cache array as a function of manufacturing and operating conditions.

Another metric used is the **field return** metric. This metric describes the number of units that are sold in the market and are then returned to the manufacturer. This situation can happen either because the unit was shipped defective or it broke down earlier than expected due to manufacturing reasons that were not detected at the testing phase.

There is also a variety of metrics to measure performance. The performance metrics used to determine how well a processor performs are: **instructions per cycle (IPC)**, number of **cache misses/number of instructions**, **program runtime**, and **transactions per minute**. Additionally, **quality of service (QoS)** is another metric that is used to show the degree to which an activity satisfies the customer, in terms of response time. For example, QoS can be 90% of response time must be below 200 ms for a web-search application. Also, **Performance vulnerability factor (PVF)** [94] is used to measure the performance degradation caused by permanent faults.

3 Challenges for performance dependability

Specific challenges for scalable performance dependability are investigated in this report, such as (i) various types of safety margins and overheads during the design and operation, which limit performance benefits from area scaling, (ii) performance non-determinism in terms of both timing and energy, (iii) accelerated wear-out and shorter lifetime, (iv) vulnerability to transient-errors, and (v) compromise of functional correctness (also parametric reliability, such as exceeding the leakage energy limits).

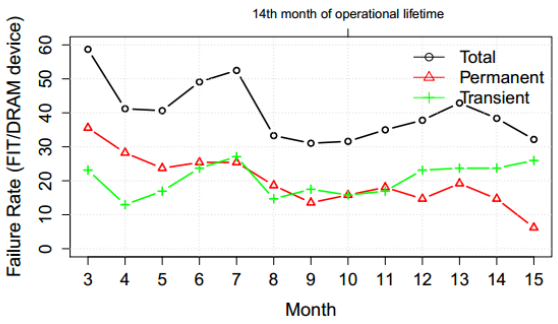

Figure 4: DDR3 DRAM device fault rates per month [210, 209].

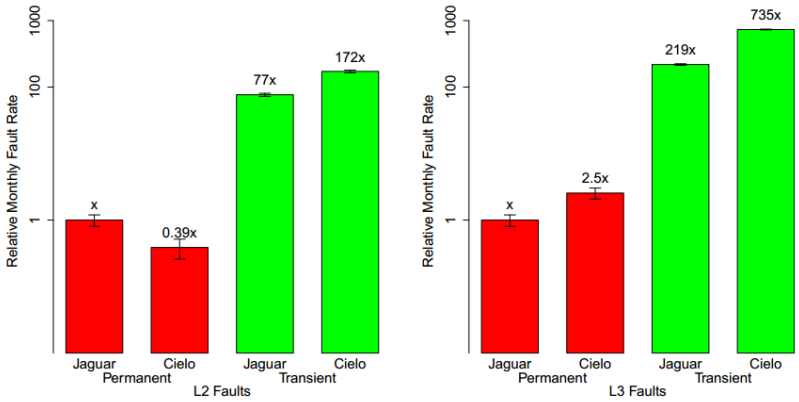

Figure 5: SRAM transient and permanent faults in the a) L2 and b) L3 caches of two currently deployed supercomputers (Cielo and Jaguar) [210].

4 Technology Forecast

Continuous device miniaturization in silicon chips elevates power, performance, and reliability into prime design constraints across all computing market segments. Due to this miniaturization, there is an increase in both static and dynamic variations. This bleak technology forecast is reported in almost every roadmap. The most recent ITRS report (2013) [8] indicates the short- and long-term reliability challenges in Table 1.

The ITRS report also suggests the use of monitors and knobs as a possible solution to the challenges that occur from static and dynamic variations. A possible monitor would be to check for circuit parts that suffer from performance degradation. Then, a possible knob could be the ability to change the biasing of the circuit at runtime.

Furthermore, a recent report [154] shows projected "pfail" values for inverters, latches, and SRAM cells due to random dopant fluctuations as a function of technology node. Table 2 shows that the "pfail" values of all types of devices are increasing dramatically with scaling. It also indicates that not all types of devices are equally vulnerable to soft errors. The trends for negative-bias temperature-instability (NBTI) are similar, as shown in Figure 1.

The work in [190] shows that the average yearly failure rates vary widely across systems they studied, ranging from 17 failures to an average of 1159 failures per year, as depicted in Figure 2. The average yearly failure rate across the nodes of the system range from 5 to 325 failures, as shown in Figure 3. The same work reports that memory was the single most common "low-level" root cause of failures for all systems. Similarly, a study of DRAM [210, 209] and SRAM [210] faults in large high-performance computing systems (that also examined the impact of aging on transient and permanent errors) has shown that there is a marked shift from permanent to transient faults in the first two years of DRAM lifetime, as illustrated in Figure 4. Figure 5 shows that more than 98% of SRAM faults in two currently deployed supercomputers (the Cielo and Jaguar supercomputers) are transient, in both the L2 and L3 caches.

We conclude this section by presenting the challenges in different market segments (datacenter, HPC, embedded), as seen in the most recent HiPEAC roadmaps [71, 72]. One of the main challenges is the power efficiency. All computing systems (datacenters, HPC, and embedded systems) are power-constrained either by battery capacity, energy cost, or cooling. Hence, it is very important for these systems to find more power-efficient solutions. One of the proposed solutions to maintain reasonable power limits is to be able to deactivate a fraction of the chip. Furthermore, addressing energy efficiency is essential for providing increases in performance. Another important challenge highlighted in the HiPEAC roadmap is reliability. It is vital to move away from the illusion of fault-free devices. Designers should come to the realization that systems are built out of

unreliable components. Consequently, reliability has to be addressed at all levels of the system, from devices to architecture to operating system and software. For datacenters and HPC systems, redundancy and statistical failure predictions are required. On the other hand, for embedded systems, new methodologies for proofing, validating, and testing functionality are needed. HiPEAC roadmap also articulates the need of dependable timing and predictable performance for all the market segments [71, 72]. The dependable performance becomes more relevant because of the increasing complexity and dynamism that is present in the application workloads.

5 Manifestation of faults in different market segments

Technology scaling trends lead toward smaller feature sizes and higher transistor densities. As a result of this miniaturization, faults are becoming more frequent and they present the processor designer with both challenges and opportunities. One of these challenges is to provide reliable operation with little or no performance degradation in the presence of faults.

Even though faults appear in all market segments, their manifestation differs. Likewise, the requirements for each system may also vary. Due to this diversity, it is worth differentiating monitors and knobs based on their market segment.

In this section, the implications of various faults to different market segments (datacenter, HPC, embedded) are described.

A very important aspect in datacenters is the availability of the services that users rely on. It is crucial for these services to be perpetually available. A failure to these systems can cause unavailability of a service at any given time. A cause of faults to these systems is the loss, or corruption, of critical data. In many cases, data corruption can pass unnoticed, due to fault- or error-masking. The faults can also be masked by a well-designed fault-tolerant infrastructure. The infrastructure will effectively make the faults invisible outside of the service provider. Failure to completely mask the fault can lead to QoS degradation.

In a similar vein, the HPC community also takes measures to cope with increasing failure rates. Nowadays, a supercomputer is constructed with the goal of delivering greater performance with low power overhead. Failures are typically expected in scenarios where two or more components work concurrently to achieve higher performance. A failure at that point can cause performance degradation, because it will cause all the components working in parallel to stop their job and restart from earlier safe states.

Finally, in the embedded systems domain, the key characteristic is that the overall setup operates under real-time computing constrains. A failure in those systems can cause a violation to the real-time deadlines. Also, a failure in embedded systems can cause energy/performance overhead and, more importantly, cause undesired safety issues.

The goal of this deliverable is to present a survey of monitors and knobs that target the implications of faults in all pertinent market segments (datacenters, HPC, and embedded), and identify solutions that can be applicable to both the general-purpose high-performance domain and the application-specific embedded world. The overall objective is to enable the cross-fertilization of approaches from both domains into a holistic and all-encompassing attack on the variability issues affecting modern and future multi-/many-core systems.

The following Table can help the reader to locate monitors and knobs for various attributes considered in this deliverable.

Part II
State-of-the-Art Monitors and Knobs

1 Introduction

Failures in computing systems have been the subject of many research studies in the past decade. The problem is becoming more important as the reduction in feature sizes increases the probability of a fault, both due to the manufacturing process, but also due to aging and soft errors. Furthermore, the issue of reliability is closely intertwined with the issue of power. The slowdown in voltage scaling has marked the end of Dennard scaling and has imposed even stricter power constraints in modern multi-/many-core systems. Elevated power consumption puts further reliability strain on the various components. In general, it is important for a system designer to consider the various factors that affect the reliability of an architecture and invent techniques to mitigate the effects of failures and maintain performance guarantees.

In this chapter, we will visit a number of techniques targeting this problem and we will attempt to locate the different monitors and knobs that are used by these techniques to detect the causes and alleviate the effects. The rest of this chapter is organized in the following sections: Section 2 addresses reliability issues and is divided into three subsections. The first Subsection, 2.1, discusses previous work on transient errors, the second Subsection, 2.2, describes prior work on permanent faults and the third Subsection, 2.3, explains the causes of faults caused due to wear-out and how they are handled. Section 3 visits work that deals with problems caused by power (and, therefore, temperature issues), and finally Section 4 discusses work that considers performance issues due to failures and how to regain the performance loss.

Each of the following Sections and Subsections in Section 2, is separated in two main parts. First, we refer to the various techniques that attempt to monitor the systems and detect or predict failures. Secondly, we present previous work that attempts to solve the problems caused due to failures, either for correctness or for performance. The main goal of this chapter is to present to the reader the state-of-the-art in monitors and knobs for reliability, by providing a summary of the recent work and attempting to label – wherever is possible – each of these techniques based on: (a) the layer in the computational stack that they are applied, (b) if they are targeting parametric or time variations (or both), (c) if they are reactive or proactive techniques, and (d) if they are applicable to high-performance systems, embedded systems, or both. A table in the Appendix summarizes all the techniques and categorizes them based on the above criteria.

2 Reliability

2.1 Transient errors

The lower voltage margins and higher process variation resulting from shrinking transistors increase the probability of transient errors. This makes transistors more vulnerable to alpha particles and electrical noises, and the errors can occur within any component of the system (Cache, Interconnect, DRAM, etc.). This section will visit various software and hardware techniques for detecting and managing transient errors. The techniques target various components and operate within different layers within the system stack.

2.1.1 Monitors

Transient errors manifest themselves in different ways depending on the afflicted unit. A transient error can simply lead to system downtime or result in incorrect output, thus making the monitoring for transient errors a critical feature in modern systems. While monitoring, the system can perform both error detection and error identification to correct them, if possible.

One of the most common techniques, especially in memory arrays, for transient error monitoring is data redundancy. Data redundancy requires extra bits that encode values stored in the array. Some of the most

popular coding schemes are parity bit, error detection (EDC), and error correction codes (ECC) [93, 102]. An odd or even parity bit is the simplest form of error detecting codes. EDC and ECC codes are more complicated and are used both for detecting and correcting errors (their correction functionality will be discussed later).

These extra bits are usually checked on a read access of a memory array entry, where the detection logic will indicate if the data in the entry is valid or if it contains any errors. Depending on the code strength and the type of error, this can be either correctable or uncorrectable. Another approach [156] suggests to check the memory array entries for errors on write accesses (Check on Write) just before the value is overwritten, instead on read (Check on Read). This provides a significant energy reduction, because of much fewer writes than reads, but also decreases the fault-coverage resulting from memory entries that are not overwritten and, thus, could not be checked for an error but requires failure atomicity support.

To reduce the probability of errors propagating through the memory hierarchy, and causing an exception, scrubbing can be used to detect additional errors. Scrubbing can be applied periodically (patrol scan), on demand (on error detection) [200], or during idle time [142], and it scans all the entries to prevent the occurrence of uncorrectable errors. This technique is used for DRAM and Last Level Caches (LLC) but also for L1 data caches, that detect but do not correct errors on the fly [119]. For non-critical applications, poison bits [237] can be used to track unrecoverable errors causing an exception only on a user visible failure.

In a different approach, researchers attempt to detect transient errors by exploiting spatial redundancy instead of data redundancy. The idea relies on the comparison of the results of multiple, functionally identical components executing the same task simultaneously. In the case of a non-unanimous result, the task can be re-executed, or the most popular output can be considered as the correct one by a majority vote. This approach is called N-Modular Redundancy [231], where N represents the number of components used. NMR can be applied at different granularities of the system (a complete system, a memory, a core, or a functional unit) and the most common forms of NMR are Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) [243, 42, 74]. The DMR technique can be used for single-error detection, while the TMR form can also be used for single error correction. It is obvious that the more instances we have, the more reliable the results are, but also the cost increases both for the extra hardware and for the voting logic to decide the result. TMR is usually used for critical applications and to assure the correctness of a primary fault-detection mechanism in a system [114].

Another way to monitor errors in the circuit is by invariance checking. Most, if not all, of the components of a computing system are defined by rules. For example, adding two positive numbers cannot lead to a negative number. Such an invariance can be used to check the output of a component for illegal results and indicate a violation in the functional correctness. Observing these cases throughout the system [171, 247] can be a cost efficient solution, both in terms of hardware and latency. More specifically, interconnects can be characterized by fixed-size packets and bound time-windows, which can be considered as user-imposed rules. These characteristics enable the use of packet/flit counting and time-out counters for detecting dropped and spurious packets/flits [87]. Flit counting could be used at the input ports, counting the number of flits in a packet and expecting them to match a pre-defined packet size. Furthermore, packet counting could be used in routers as a whole, counting packets inside a router, where there can never be a negative number of packets and all packets should leave the router in a pre-defined time-window.

A network-level solution for interconnects could be to send a notification before sending the actual packet to the destination, enabling the destination node to keep track of expected incoming traffic. This could be done by utilizing a secondary checker network for sending a signature of a packet [12], or even a simple notification [159] to the destination node where the actual packet is sent over the primary network. This way, the destination nodes can verify packet integrity or keep count of pending packets.

Besides data and spatial redundancy, there is also a third category of schemes that exploits time redundancy. The main idea of these techniques is to run two copies of the program and compare the results to ensure higher fault coverage. For example in AR-SMT [182], there are two threads, the active and the redundant threads, which execute the same instructions and compare their results. When the results are identical, instructions are committed, else the redundant thread will provide the information for recovery. For detecting permanent errors, the two threads are required to use different functional units [191]. Another related concept is that of

idempotent tasks [147]. Idempotence ensures that a task which can be executed multiple times should always have the same result as the first execution, and a fault is detected when the idempotent property is violated.

In the same vein, instruction re-execution can be used to detect errors. This technique can be implemented either in the processor level, such as DIVA [24], whereby a checker is placed at the commit stage, or at the compile time, such as SWIFT [178], where instructions are duplicated during compilation. In both cases, the main idea is to check the instructions results by executing them twice and in case of a difference a possible error is detected.

Finally, at higher levels of the compute stack, transient error detection can be achieved by monitoring the symptoms [234], which are usually memory access violations, alignment exceptions, and branch miss-speculations. Fault screening is an alternative way for symptom based detection and this can be achieved by monitoring the unprotected structures (which do not contain any ECC or parity protection) for program's state internal inconsistencies compared with past behavior [174].

2.1.2 Knobs

Although some errors only require detection and re-execution, there are certain cases where recovering from an error is crucial to continue normal operation. Some techniques exploit the same mechanisms both for detection and for recovering from transient errors. For example, in memory arrays the data redundancy is used both for monitoring but also for correcting faults. More specifically, in ECC codes, a 64/8 SECDED (single error Correction Double Error Detection) Hamming [93] or Hsiao [102] code can be used detect double bit errors and to correct single bit errors in a word. ECC and EDC codes are commonly used to correct errors in memory arrays such as caches and DRAM.

Another technique for correcting cache errors is by extending caches with two-dimensional parity bits [122]. This approach contains an EDC along a cache row that detects the error and an ECC along the cache column that corrects the error enabling more error correction coverage against higher cost. In addition, a new cache [141] employs a parity bit to detect an error and two XOR registers to correct the error. The first register stores the XOR of all the data written to the cache and the second register stores the XOR of all dirty data that are removed from the cache, providing a fault recovery mechanism at a low cost.

Transient error correction in DRAMs is implemented slightly differently in memory DIMMs, to include one or more extra devices to store the ECC code. The codes can be extended to double bit error correction and triple bit error detection (DEC-TED) [137], for multiple error detection and correction [177], or even for Single-Chip error correction and Double-Chip error detection (SCCDCD). SCCDCD codes, also called "chip-kill," are commonly used for DRAM protection on large-scale systems and the mechanism is mainly implemented with three different schemes. The first scheme uses interleaving SECDED code words across 4 ranks, such that no more than 1 bit comes from a single DRAM chip [65], the second scheme employs stronger error detection and correction codes to construct a 144-bit code across two ranks [1], and the third scheme provides 128 data bits to an ECC word over two bursts across one rank [10].

Some other techniques have been proposed to provide chip-kill-level reliability with lower overheads. Virtualized ECC (VECC) [245] separates error detection and correction into two tiers. The first tier is responsible of error detection and it consists of a simple error code for detection or light-weight correction. The second tier –which is mapped to a different physical address and shares the same name space and storage devices as the data – is responsible for data correction when an error is detected. The other approach that ensures also chip-kill-level reliability is the LOT-ECC [224], which uses multiple-tiered checksum and parity fields.

Another direction of work targets the protection of addresses from transient errors. Schemes were proposed to embed the address into the data by XORing them together [145], or by encoding it in the ECC codes [91]. The last technique is taking advantage of the fact that SEC-DED codes are shortened codes, which means that the number of data bits to be protected is smaller than the maximum bits that can be protected, and thus can be used to encode additional information. Although this might result in code weakening, the redundant-encoding-of-attributes technique can be used to encode the same information in multiple code words in a cache or memory [187] to recover part of the code strength.

Even-though most of the techniques used for transient error correction operate before the manifestation of an error, due to the criticality of the errors, techniques that react to this manifestation can also be applied for certain applications. Check-pointing [158] is a technique used to recover from transient errors. During normal operation, intermediate fault-free system states are stored, and once an error is detected, the system is notified to roll-back to a safe checkpoint and restart its operation from there to prevent the usage of potentially corrupt data [207]. In an interconnect, the check pointing could be done in each router separately [50] and a recovery pointer could be kept at each input buffer pointing to where the operating header could roll-back to and reroute lost packets. These pointers have to be kept until the corresponding packets have reached the next router.

In large-scale supercomputers, check-pointing can be both within a local scope, as described before, but also within a global scope. For example, on a parallel execution on a set of n nodes, any component can fail and therefore the program will be forced to recover from a global check-point. A major challenge to global check-pointing is the scaling of the systems, due to the fact that a single failure requires the entire system to roll back to a safe state. A solution to this problem could be to divide execution in intervals [165] and create system checkpoints when an interval is verified. More specifically, each time a core reaches the end of an interval, it creates a checkpoint and continues execution. This checkpoint is later committed upon receiving a healthy status from all other cores. Otherwise, it stops operation and restarts from the last safe checkpoint. Another problem faced with global checkpoints is synchronization. System synchronization could be achieved by check-pointing at a finer-granularity (node check-pointing) in order to reduce the effects on the entire system [92].

A important factor of the effectiveness of check-pointing techniques is the amount of overhead required to keep the checkpoints and also the frequency and delay required to create them. There has been work [57, 246] and mathematical models proposed [70], which consider the checkpoint effects and attempt to optimize check-pointing by fine tuning the check-pointing frequency to reduce the overhead for both local and global check-points. Check-pointing or buffer states are also used in conventional processors at various points in the pipeline but processor that employ idempotency address the problem in a different way. Such processors, execute program binaries divided into idempotent regions and satisfy the idempotent property [61, 123]. When an error is detected, during the execution of a region, this processor recovers by simply jumping back to the beginning of the region. To guarantee successful recovery, the idempotent property ensures that the execution does not proceed beyond the end of the region until that region is verified to be free of errors. An important characteristic of check-pointing and idempotent recovery is that they provide failure-atomicity. Failure atomicity describes the property of having a sequence of operations to either all or none complete. When there is no completion the state of the system remains unaffected. Map-reduce programming [63] and Transactional-memory [99, 240] leverage failure atomicity.

2.2 Permanent errors

Reliable circuit manufacturing becomes harder as the feature size decreases and, although millions of dollars are spent to improve yield with technology scaling, transistors are still prone to failures. Permanent faults may appear during the manufacturing time, resulting in yield reduction, but also later in the circuit's life, resulting in performance degradation, or even complete failure of the component. In this section, we will discuss various techniques addressing the need for reliable components. The first part of the section discusses techniques that apply to the detection of permanent faults in both hardware and software. The second part of the section focuses on different knobs that can help restore a system to a fully functional state after a permanent fault.

2.2.1 Monitors

A frequent approach for permanent fault detection is to duplicate hardware. Hardware duplication is applicable at different granularities, based on the desired corresponding granularity of identification. For example, double modular redundancy (DMR) can be used with two cores to verify each other's execution [130, 15]. This is a coarser-grain approach, where a fault is identified at the core-level. At a finer granularity, the detection can be done within a single component; for example, a hardware monitor for array structures, like SRAS [35],

where errors are monitored with the use of check rows. This gives the ability to diagnose exactly which row of the array is faulty. Check rows identify the presence of a permanent fault upon a mismatch between the read values of both normal and check rows.

In addition, for monitoring wiring and logic, the Built-in Self-Test (BIST) mechanism can be used. During the test, the circuit is taken offline and a signature input, designed to exercise most of the circuit's features, is applied. Then, the output is compared with a signature output (the expected correct result) to verify the circuit. The circuit under test can have an arbitrary size. For example, in interconnects it could be as small as a single link [131, 230], or as big as a whole router [114]. Another important aspect of BIST is the interval at which it is employed. Usually, the verification process is employed at fixed intervals but, in an attempt to reduce power consumption and performance overheads, a trigger could be used to employ the checkers only on specific events, e.g. when a fault is detected globally [68].

A similar detection mechanism to BIST, where the circuit is at the granularity of a module (e.g., a functional unit), is the use of residue codes [157, 208]. Residue codes are used for detecting erroneous arithmetic operations. They are popular for their simplicity and their low implementation overhead. They can also provide separability between the result of residue codes and the result of regular arithmetic operations. Incorrect results from defective functional units can also be detected by re-executing the same arithmetic operation using shifted operands [163] on the same functional unit. An alternative way for detecting defective functional units is the use of sub-checkers [244], which verify the correctness of functional units' calculations.

In the interest of minimizing hardware complexity and overheads, monitors could also be implemented in software. Comparison between static and dynamic analysis is an alternative option for detecting faults by constructing flow graphs at compile time (statically) and during workload execution (dynamically) for comparison. A mismatch between the two flow graphs is an indication that an error affected the execution. Since control flow and data flow are not independent, by combining them the detection becomes stronger [145].

Machine Check Architecture (MCA) is an interface between the processor and the OS used in real products. This mechanism is used for detecting and reporting critical machine check exceptions and errors that are not correctable [11, 2, 109]. The OS, using the MCA reports, examines the error and determines if it is contained to an application, a thread, or an OS instance.

An additional monitoring tool at the OS level is the collection of ECC syndromes, given that ECC codes are used in the underlying hardware. Specifically, decisions and actions taken during an error correction could be used in extracting general conclusions, or detecting permanent-faults. One way this could be achieved is by keeping a scoreboard with the positions and the frequency that an error occurred and check if any frequency surpasses a set threshold. A possible implementation of the ECC syndromes could be to keep track of faults on link wires [87, 192, 161] to check if one of these wires is no longer functional.

Lastly, defective hardware can also be detected at higher levels of the compute stack. Instead of monitoring a system for faults, an alternative solution will be to watch for their implications, such as anomalous behavior at the software level. Such anomalies can be fault traps (that are not thrown during normal operation), hangs (by monitoring the frequency of branch instructions), high number of contiguous Operating System instruction [134] and panics (kernel detection of unrecoverable errors that occur during execution) [96]. These monitoring techniques imply negligible (or even no) changes in hardware, thus making them an attractive solution, but their fault coverage is imperfect and their detection latency may be long.

2.2.2 Knobs

In contrast to transient errors, the permanent errors require faults to be repaired to prevent complete system failure or performance degradation. The most natural approach to a permanent fault is to no longer use the faulty component by, somehow, reconfiguring the system to ignore it.

A way of reducing permanent faults is by handling static process variations. An approach to safely achieve this is by using cycle stealing techniques [220], where the time slack of faster stages is transferred to the slower ones by skewing the clock arrival times to the latching elements of the pipeline stages. Another approach is

to rely on a checker unit and trade-off error correction latencies with higher frequency (to gain performance), assuming the worst-case paths are not utilized often [185]. An alternative solution is to use variable/adjustable latency [136] for key microarchitecture units, while keeping a constant global frequency, or using different voltage supplies for speeding up slow paths and slowing down fast ones.

Since modern microprocessors can support more than one thread in the same core and more than one core on the same chip, researches have proposed to exploit intrinsic redundancy. Intrinsic redundancy can be achieved at the pipeline stage granularity, where different cores can use pipeline stages of others if a fault occurs [181]. All pipeline stages can be shared across all cores, and any core can be cannibalized, enabling other cores to occupy their fault-free pipeline stages. A similar approach can be applied at the thread level where threads can avoid faulty components [196] to minimize the yield impact and sustain the core's functionality. This technique gives the illusion to the operating system that the underlying hardware is fully functional, while the firmware manages the faulty components by using thread migration when necessary [170, 112].

Furthermore, faulty hardware avoidance can be applied to the interconnect, which is a critical component in the so called "uncore" portion of a multi-/many-core microprocessor. Permanent faults within the on-chip interconnect backbone are extremely critical, since they either lead to faulty links, or faulty nodes, followed by a communication interruption. A trial-and-error approach, where the system iterates through several spare components until no error is detected, can be used to locate alternative routes for rerouting [50].

Another, somewhat brute-force, approach would be to equip the interconnect with two-channel switches, i.e two physical links between any two routers [115]. In a fault-free system, this feature provides the benefit of a better QoS, but in the case of a failure, the secondary switch works as a spare and allows the node to stay connected. Similar techniques can be employed at a finer granularity to include a crossbar bypass bus, a flexible FIFO buffer design which can bypass faulty positions, and an input port-swapping technique that is used to maximize the functional bidirectional links by changing the mapping of input units to ports [165].

In an effort to minimize hardware overheads, a "partially faulty link" can still be used to transfer data in a truncated form in multiple cycles [3] [230]. Taking advantage of the fact that links, throughout a system, consist of many wires and it is highly unlikely for a fault to affect all of them at the same time, the "partially faulty link" can remain active and the remaining functional wires could be used for sending information. The network availability can be increased even further by providing spare wires in every link at design-time, which is cheaper than a secondary network, and swapping them with the faulty ones at runtime [89]. Spare cache-to-cache wires can also be used as an emergency measure [67] to transfer the architectural state and cached data of a disconnected node to a nearby connected node's cache.

Routing reconfiguration is another widely used approach for avoiding permanent faults in interconnects. The main concept is that when a link or a node becomes faulty, a new fault-free topology is calculated and the routing tables at each router are updated accordingly by a central manager. The manager is responsible for maintaining the network's state and informing the nodes when a change has to be applied. Reconfiguration information can be passed to the network as a distinctive message to each node, or there could also be a predefined control network [213]. The information kept varies depending on the technique. For example, it could depend on the routing algorithm [131], or on whether the links are considered as individual unidirectional links [161] or as a pair forming bidirectional links.

On the other hand, routing reconfiguration could be implemented in a distributed manner. A router's reconfiguration is performed in collaboration with its neighbors to update their routing tables, on an adjacent link failure [68]. A simple, distributed and global reconfiguration could be achieved by broadcasting a reconfiguration flag upon a fault detection which can setup a new topology with the root being the node that initiated the reconfiguration [16]. This could also be extended by applying a selection phase at the start of the reconfiguration process, where the node providing the highest network connectivity can be used as the root of the new topology [160].

The concept of reconfiguration flags is also used in Immunet [172], which leverages the fact that in any connected graph, there is at least one spanning tree. Combining this information with the fact that all healthy links are bidirectional, a safe ring traversing all connected nodes can always be found, at least once.

When a permanent fault is detected in an interconnect, network draining is usually supposed to happen, but

it might not always be possible. To this end, a secondary, verified and fault-free, network could be used to transfer trapped packets to their appointed destinations [12]. Coverage could be broadened to the recovery of even dropped packets, by retaining a copy of the inputs until it safely reaches the next node [159].

Moreover, interconnects are, at their core, a small-scale network and thus routing could be done dynamically, called "adaptive routing." For example, nodes could keep health information about their neighbors in a 2-hop range and can make local-view decisions that will bypass any faulty components achieving minimal and fault-free paths [73]. However, in cases where 3D NoCs are employed, packets can move in an additional direction. A heuristic could be to always try moving on one axis when possible and then on the plane created from the other two [175]. A more sophisticated solution could be to extend the interconnect with wireless links that would turn it into a small-world network. This would lead to reducing the hops of possible paths and routing could be calculated after an exhaustive search through every possible path [86].

Continuing our exploration of the criticality of permanent failures at the "uncore" granularity, we shift our attention to the memory sub-system. The criticality of the memory system is accentuated in systems with heavy reliance on vast amounts of physical memory. For example, most of the functionality of large-scale datacenters is based on memory transactions, thus making memory the most critical component. Consequently, several techniques have been proposed for the recovery of permanent failures in the memory of such large-scale systems. The first task, when an error appears in memory, is to initiate a "recovery action" depending on whether the error is correctable, or uncorrectable. The choice of "recovery action" depends on the recent repair history that can be found in an error log file [4]. For instance, if the error is correctable and appears few times in the same location, then the ECC [93, 102] codes are responsible for the repair action.

In the case of an uncorrectable error, offline or online maintenance actions are required to recover from that error. During offline maintenance, the faulty component, or the whole server (depending on the system maintenance model) needs to be replaced with a cold spare. In the case that the correctable errors exceed a predefined threshold, the management processor sends an alert to the server's administrator, and the administrator can view a log file of correctable and uncorrectable error events through a Management log file [5]. This log file can be used for better handling of correctable and uncorrectable memory errors.

While offline maintenance requires the physical replacement of the faulty component, the online techniques attempt to solve the problem without interrupting the execution. An example of an online maintenance technique is page retirement [217], where a faulty physical page is removed from the available memory space and its content is reallocated to another physical page. To compensate for the lost entries, active spares [35, 113] can be employed to remap the faulty component, or even the whole device. Sparing is a design feature found in certain datacenters. DIMM-sparing provides protection against persistent DRAM failures. An excessive number of correctable errors can also be considered as a permanent fault and, although they do not affect correctness, they degrade performance, because of the recovery overheads. In such cases, a copy of the contents of an unhealthy rank to an available spare portion can be made to resolve the problem [9, 7].

A more advanced feature in today's systems is known as Intel Cache Safe technology [4] and is used for multi-bit error protection. The system first determines if the failed cache bit has a permanent or transient error, and in case of a permanent fault, the defective cache line is disabled and the data moved to a spare location. Also, some systems can provide main memory mirroring [9, 7], which maintains two copies of all data. If an uncorrectable ECC memory error occurs, the system automatically retrieves the fault-free data from the mirrored (redundant) copy and the system continues to operate normally without any user intervention.

Mitigation of permanent faults can also be achieved at the software level. For example, if a functional unit is faulty, a calculation can be performed by emulating it in a different unit [112], or by using the functional unit(s) with narrower inputs. Moreover, static and dynamic compilation, as well as binary translation, could be used for modifying executed code [146]; for example, to avoid assigning values to faulty registers. Another approach, applied at the software level, is the use of MCA [4]. The MCA recovery function generates a machine exception when it detects an error, and the OS is responsible for choosing whether to shut down the thread, application, or the virtual machine to prevent a crash. For arrays, it has been proposed to avoid the faulty entries by not assigning values to faulty registers, or change code to avoid specific entries in the instruction cache by using branches [146].

Error recovery can also be performed at a very coarse granularity, e.g., at the infrastructure level. Again, this is typically applicable to very large-scale systems, such as datacenters. The infrastructure of such systems requires high availability and reliability for the offered services. This can be achieved by redundancy and replication. Many services necessitate machine replication, because of huge working sets and the need for short latency and high throughput. The same data is stored on multiple storage devices and when a machine fails, the job running on it can be migrated to another machine, together with the replicated data [26]. Replication can mask faults from the end-user, if the faults can be pro-actively prevented, or reactively recovered from, by a well-designed fault-tolerant infrastructure. Unmasked faults result in Quality-of-Service degradation, or Quality-of-Content degradation. Data partitioning [26, 41] can also be used to facilitate both reliability and availability, and at the same time to improve throughput. Partitioned data across servers increases the aggregated capacity, reduces response latency and it also implies less lost data in case of a server failure.

2.3 Time-Dependent Variations

While in the previous section we discussed faults resulting mainly from static variations, in this section we will focus on the fault causes due to time-dependent variation. A system's lifetime is highly dependent on the implications of time-dependent variations. The most dominant time-dependent artifacts that become more pronounced with transistor miniaturization are: (a) TDDB [127, 120] (Time-dependent gate-oxide breakdown), where the gate oxide breaks down from long exposure to relatively low electric fields; (b) NBTI [188, 103] (Negative Bias Temperature Instability), which shifts the threshold voltage and reduces drain current as a result of pMOS transistor stress; and (c) HCI [14, 48] (Hot Carrier Injection), where the transistor switching characteristics change because of large drain-to-source voltage, or gate-to-source voltage applied. Such phenomena are further accelerated when transistors are exposed to high temperatures or high switching activity. This section describes monitors for detecting circuit aging and knobs for delaying, hiding, or recovering from wear-out.

2.3.1 Monitors

The implications from time-dependent variations have the property to appear gradually through a change in the transistor characteristics. Such characteristics enable circuit wear-out detection before wear-out actually reaches the point that it will affect the system's operation.

Circuit aging can be characterized with models attempting to identify the different wear-out causes based on low-level monitor readings. Monitoring voltage and temperature, which are highly dependent on the dynamic behavior of the underlying hardware, can help estimate the MTTF of the observed component [211, 81]. Additionally, monitoring resource utilization can be used for a circuit's decay estimation. For example, interconnect aging can be predicted based on the activity and utilization that the routers and links have experienced [31, 121].

A straight-forward approach to identify circuit aging is to test the actual hardware under aggressive operating conditions, such as lower supply voltage or higher frequency, which will potentially expose the circuit's wear-out [149, 81, 206, 216]. The identification is achieved by running test programs on cores [206, 149] under testing conditions, by taking the system offline, or by taking offline only a portion of the system while the rest of it continues to operate normally. This method is also proposed for ranking cores based on their ability to operate under different conditions.

Instead of applying tests on the underlying hardware, a similar and more accurate solution is to estimate aging based on signal propagation. The effect of NBTI can be measured by comparing the drain current of the stressed circuit signal and an unstressed one [173], but also by measuring the signal propagation latency (available slack time) [33] in combination with past measurements. Any changes between recent and past measurements will be an indication, and probably a result, of circuit aging. An alternative way for characterizing the presence of NBTI phenomena is the sub-threshold leakage current. A reduction of the leakage current indicates a shift in the threshold voltage, which is a consequence of the NBTI effect [116, 84, 202].

Test structures are another option for low-level circuit aging monitoring. Test structures are in-situ monitors that can be placed in the integrated circuit and, based on the operating conditions of the circuit, can detect any decay that may be caused by different aging phenomena. To achieve this, two identical circuits are required, one that is stressed and another that is not. Based on the wear-out mechanism that the designer wishes to monitor, different approaches are adopted. In the case of NBTI and HCI, detection can be achieved with the help of ring oscillators and the use of comparator to characterize the effects of NBTI and HCI by comparing the beat frequency of the two circuits, while in the case of TDDB, gate leakage must be compared to detect aging signs [118, 117].

2.3.2 Knobs

Although today's microprocessors are tackling time-dependent variations by extending safety frequency guard-bands, this might not be feasible in the future, due to rapid circuit aging in short amount of time, which demands even larger guard-bands. To mitigate time-dependent emerging faults, researchers have proposed techniques that either attempt to slow down the aging, hide it, or recover from it when it is feasible. Usually, knobs targeting time-dependent faults are acting pro-actively in an effort to reduce, or balance, the circuit stress as much as possible.

The NBTI aging effect afflicting pMOS transistors will be more evident in the next-generation processors due to smaller feature sizes and power constrains. Much effort has been put on solving the problem of NBTI. It has been observed that a transistor can partially recover from NBTI if the stress on the transistor (when it operates with negative gate-to-source voltage) is alleviated. Infused by this NBTI recovery feature, many techniques have been proposed to exploit it.

A recent study shows that the effect of charged gate oxide defects on the characteristics of pMOS transistors is reduced both for time-zero and for time-dependent variability when a forward body bias is applied. This improvement is due to a reduction in threshold voltage variability (at time zero) and a reduction in single-defect-induced ΔVth [83]. Power-gating has been found to accelerate the pMOS transistors' recovery from NBTI. Power-gating can be used at a coarse granularity to power off a whole core or multiple cores [228], and at a finer granularity to power-gate idle VCs in an on-chip router [52, 199, 254].

Similarly, another approach could be clock-gating. Here, the power supply would still be available, but the clock signal would get cut off [253]. In combinational logic and functional units, the inputs remain the same during idle periods, and thus the same pMOS transistors can be affected. NBTI recovery can be achieved by applying specific patterns in which different inputs will be exercised, so that different pMOS transistors will be affected. This results in a more balanced activity of the pMOS transistors, leading to lower stress that will improve the components' lifetime [13, 84]. An alternative activity-balancing technique for functional units is micro-architecture scheduling [198], whereby instructions are scheduled for execution in such a way as to achieve balanced activity and reduce the NBTI effect.

The pMOS transistors in SRAM cells tend to be stressed more, because they hold the same value for longer periods. Larger 8T and 10T cell designs have been shown to be more resilient to NBTI, due to their ability to operate at lower supply voltages [128, 46]. A modified SRAM cell, which is able to switch to a recovery mode using a technique called Recovery Boosting has also been proposed [197].

Furthermore, micro-architectural redundancy can be exploited to pro-actively deactivate and rotate portions of the cache by migrating contents into spares [195]. Thus, the impact of NBTI is reduced, or the system can recover from it through reduced stress. Along the same lines, instead of migrating the cache blocks, one could simply invert their contents in SRAM cells (specifically for invalid data) to balance the stress in pMOS transistors [13]. More generally, aging can be slowed down by using Backward Body Biasing and adaptive supply voltage reduction [221], in order to minimize the stress applied to the circuits.

In the case of the on-chip interconnects, architectural reconfiguration could be applied by changing input-output mappings on crossbars [20] to vary the values. Smarter arbiters could be used to give priority to less stressed Virtual Channels [85] and balancing could even be applied by dividing information into chunks of bits and rotating these bits inside a channel. Further, the concept of adaptive routing could also be used to

minimize the aging effects on the various routers/nodes [31, 85].

At the software level, time-dependent-variation-aware scheduling is another possible solution that tries to schedule heavy workloads on cores that have been less affected from aging, as compared to others [221, 81], or to balance the stress across the processor.

3 Power Management

Power management and thermal balancing are two important challenges that we will face during the HARPA project. Violating the power and thermal constraints of a component may result in an unstable and unreliable behavior. It is clear that power and dynamic thermal management techniques can cover a wide range of applications, both in high-performance and embedded systems. Different monitors are required, based on the application and the system type, but also different knobs need to be used considering the physical constrains of the system. In this section, we describe some thermal monitors and knobs and we focus on some techniques that are well established and widely used to resolve the problem.

3.1 Monitors

The most common thermal monitor at the circuit level is a thermal sensor, which is placed strategically close to potential hot-spots, and it monitors the temperature at frequent intervals [203, 204]. A safety margin is always considered, since the thermal sensor cannot be placed directly on top of hot-spot prone logic, such as the register file.

Thermal monitoring becomes more complicated when it is applied to multicores. Thermal coupling is the problem when the temperature of a core does not depend only on its own activity, but also on that of neighboring cores [69]. Work has been proposed to observe both off-line characteristics, but also run-time activity, to predict thermal coupling effects in MPSoCs [51]. In addition, when processors are heterogeneous, detecting thermal coupling is even harder.

Recently, a technique has been proposed that considers the effect on performance of a heterogeneous multicore with a CPU and a GPU [164], when throttling is employed to reduce temperature. A dynamic power management technique has been proposed that considers both the thermal sensors, but also the performance reduction on each core. The frequency is then scaled to balance thermal coupling and optimize the performance under a given constraint. More advanced algorithms have modeled the thermal behavior problem as a control theory problem, aiming to design an optimum frequency controller by achieving thermal balancing in the multicore's thermal profile [248].

Thermal sensors are expensive though and contribute to the power envelope. An alternative way is to model power at run time in order to shave this overhead. These models collect various architectural events via performance counters to model the power consumption and locate hot spots [132, 201].

Such an approach can be even more appealing when it comes to embedded devices. The limited resources in such devices, like power and area, prevent manufacturers from using conventional thermal sensors, and more practical techniques are required. For example, memory accesses and footprint [23], or ISA characterization [38], can be an indication of high activity and, therefore, high temperatures in specific hot-spots. Additionally, embedded devices need to monitor the temperature and adjust power for reliability, but they also need to monitor power consumption and energy availability for sustainability purposes [29].

3.2 Knobs

Provided that the monitors manage to detect, or predict, any thermal issues – or other operational threatening events, like battery drain – certain knobs need to be applied to prevent the disaster. The most commonly known Dynamic Thermal Management (DTM) technique is Dynamic Voltage and Frequency Scaling (DVFS) [238, 43]. DVFS is widely used in modern processors – embedded, mobile, and general-purpose – and it can reduce power density and temperature by reducing the voltage up to the point where the devices will continue to operate without errors. When voltage is reduced, the frequency also needs to be reduced to avoid timing

errors in the logic. In a similar manner, the DVFS technique can be applied to memory controllers [66], by dynamically varying voltage and frequency to trade available memory bandwidth, and, therefore, conserve energy when memory activity is low. The DVFS margins can be extended if the technique is also combined with Adaptive Body Biasing (ABB) to allow further reduction in the voltage [143].

Most of the existing DVFS technique propose predefined states, where both power and frequency scale to avoid undesirable conditions. A control mechanism has been recently proposed to jointly control the voltage and frequency transient periods, where both actuators are developed independently [17] ensuring global system stability. The proposed controller can increase the system's performance during the transition, if frequency is allowed to scale less than the voltage.

Fetch gating, and frequency-only scaling [43] are two other DTM methods. Both techniques attempt to reduce the cores' activity, either by toggling on and off the fetch stage, or reducing the frequency to lower the power density and, thus, the temperature. A variation of fetch gating is Flicker [166], where the pipeline is sliced and certain slices are power-gated to reduce power consumption when possible. PowerDial [101] is another type of fetch gating, but at a higher level, where a system can be manipulated to dynamically trade off the accuracy of the computation in return of a reduction in the computational resources.

Power gating [169] is a more aggressive technique for reducing temperature by reducing power density. This technique completely switches off specific components in the core, caches, execution units etc., or an entire core. Power gating can be applied to processors through ACPI, an interface of power gating that defines power states and power-performance states for the processor and other hardware devices. For example, the CPU power state C0 implies that the CPU is fully operating, while the C3 states implies that the CPU is in a sleep state. Manufacturers can add additional power states if needed – such as a new family of barely alive power states – and enable fast state transitions between the different states [19]. There are other approaches, including techniques for conserving energy in services, which consolidate workloads into a subset of servers and turn others off [45, 47, 167].

Studies have shown that Dynamic Voltage Scaling with No Frequency Scaling (DVNFS) can be employed, and performance degradation due to errors can be mitigated by techniques that can detect them and recover. An example is the Razor technology [75] which enables a circuit to work below voltage margins to reduce its dynamic power consumption (up to 40%). It does so by monitoring the error rate and adapting the voltage. For recovery in a Razor pipeline, two schemes were proposed: global clock gating (with impact on the operating frequency) and counterflow pipelining (with higher penalty in terms of wasted cycles). The work in [186] eliminated the clock delays required by Razor, by using canary logic. For SRAM cells, the concept of canary bit-cells was proposed to act as feedback architecture for sensing the minimum supply voltage that an SRAM cell requires to be able to correctly hold a value [232, 233].

A similar approach is to directly reduce the voltage to the safest working margins [218, 133, 25]. The working voltage can either be known a priori, by testing each core, or it can be detected dynamically. The dynamic approach can be achieved either with special hardware to detect working-voltage margins for each logic component or pipeline stage, or by adjusting the voltage margins in very small voltage steps and monitoring the correctable error reports raised by the hardware. The solution can detect the safest voltage for different cores on the chip and adapt the supply voltage to each core, based on its variability. Such techniques are useful for both variation-aware tuning, or for detecting errors and adjusting the voltage resulting from aging problems.

Thread migration is another method used to reduce the chip temperature. Various allocation algorithms are used, such as round-robin migration, or hot-to-cold allocation [54, 69, 98, 204]. Assuming a heterogeneous multicore – or heterogeneous activity within the multicore – the algorithms attempt to migrate the threads around the chip, in order to produce a more uniform power density profile and, consequently, lower temperature, by migrating threads from active and hot cores to either active or idle colder cores. A technique called Adapt3D [53] for dynamic thermal-aware job scheduling was proposed for 3D systems. The thermal profile of other layers in the 3D stack is also considered when applying DVFS and migration to reduce the temperature.

A different approach is applied in on-chip interconnects, as an alternative to voltage and frequency scaling. In an attempt to reduce power when the network traffic is low, the network bandwidth can be divided to sub-networks [59]. Packets are injected in the same sub-net until it is congested, before moving to the next one.

This enables the network to power-gate unused sub-nets without affecting performance and, additionally, to provide a level or reliability through redundancy, by having multiple routers connecting a node to the network. Thus, in the case of a defective or damaged router, the network would remain connected.

In the embedded world, there are physical constraints that restrict conventional DTM techniques. For example, space and power are an issue, since embedded processors are usually fitted in space-limited locations and/or they have to be self-sustainable, so conventional cooling techniques, such as heat-sinks and fans, are not possible [242]. A major benefit of embedded systems is that the workload and applications running on the processor are (for the most part) known a priori and they are typically fairly constant and predictable. This gives the opportunity for adapting the DTM techniques to be more efficient. For example, thread migration techniques can be used more efficiently and planned ahead of time, since the workload is known [194], while the thermal profile can be predicted based on the application characteristics and activity monitoring [38]. Moreover, different DTM techniques can be used based on the location and application type of the embedded system [162], assuming different ambient temperatures must be tolerated.

Furthermore, aside from DTM techniques, power management techniques can be employed in embedded systems to control temperature and power consumption. The Linux Control Groups can be used to manage resources at the software level to address power, thermal, and reliability issues [30], or by switching microcontrollers at the architectural level [39] to achieve similar results.

4 Performance

Although fault tolerance is the major concern when designing for reliability, performance guarantees are equally important for real-time systems. Fault tolerance strives to keep the functionality of a system unchanged in the presence of faults, and techniques have been proposed that complement this effort by increasing system performance in the absence of faults. Such techniques can also be exploited in the presence of faults, in an attempt to maintain performance as much as possible. We have to note here that faults can also have an impact on performance without affecting correctness, such in the case of errors in predictors [129].

This section will first focus on techniques that enable performance monitoring and, secondly, it will visit techniques for controlling and improving performance. These techniques are examined at various levels of the computing stack and range from hardware to software solutions. Finally, performance modeling techniques will be discussed to provide the state-of-the-art in prediction of system performance.

4.1 Monitors

The most common performance monitoring tool – leveraged by most techniques – is the performance counters that are provided by the hardware vendors as registers. Performance counters enable applications to analyze performance and they can be exploited and configured by the user to observe behavior. By tuning correctly the counters, performance monitoring can be conducted [110, 6, 79] to provide critical information for reliability and performance, and it can possibly lead to optimizations, or fault-preventing actions.

Although simple register counters provide most of the information, further analysis needs to be provided when co-scheduled threads interfere with one another. This situation can affect both fairness and performance. This interference could severely damage the performance of a starving thread and the phenomenon can be observed in any shared component. For example, in the case of a last-level cache, an unusual miss rate can be observed, which is attributed to interference between threads on the same cache sets [250, 80].

Main memory is also an important shared resource, where the memory-request serving rate can be monitored as an indication of interference and possibly performance degradation [214]. In addition, isolated thread performance can be estimated at run-time by measuring the computation cycles, miss cycles, and wait cycles [77]. For a thread, if the wait time is significantly longer than the time spent on execution or the time waiting for data, then this is an indication of an interference from other threads. Another approach is to put the memory system under stress to measure any possible increase in latency [241]. This technique provides a characterization for any application and enables the detection of performance degradation at a coarser level.

Heterogeneous multi-core systems are increasingly being used, since they can enable higher performance given a fixed power envelop. This is the reason that modern microprocessors incorporate accelerators along traditional processors in the same chip, e.g., GPUs. In such systems, memory-intensive applications do not suffer from performance degradation when executed on a small (e.g., in-order) core. On the contrary, compute-intensive applications benefit from executing on a large (e.g., out-of-order) core. So, random resource allocation can damage system performance. To avoid unnecessary performance degradation, some statistics must be monitored. CPI stacks, memory level parallelism, and instruction-level parallelism are factors that are agnostic of the heterogeneity of the system. Thus, these factors can be used to monitor system performance degradation [56], or slowdown of threads [55].

Moving further up in the compute stack, large data-centers are required to monitor the performance of individual services in order to maintain performance guarantees and QoS. The main concern is a slow server, due to failure in the link or its hardware, and even worse the server being unresponsive to requests. To maintain the QoS, such events should be detected instantly and any pending or new requests should be steered to another server. To achieve this, timeout counters are used, also called "watchdog" timers, which are set empirically to abort unusual long-running requests [26, 145]. When a machine fails to meet the timing constrains, the watchdog timer will generate a timeout signal and actions should be taken accordingly. Watchdog timers are also commonly found in embedded systems, where real-time computing constraints must be retained and must be able to react to faults in a timely manner.

4.2 Knobs

Process variation forces the chip manufacturers to "frequency-bin" their processors based on their worst-case execution. This results in performance degradation (due to the lower frequency), even though the chip could probably run at higher frequencies for most of its lifetime. A way to relieve this issue is to set the processor's frequency based on the average execution instead of the worst-case execution, thus providing higher performance with the trade-off of possible timing errors. Aside from frequency tuning, the performance can also be managed at the thread level either in multicore systems or simultaneous multi-threading cores [223]. By exploiting the monitoring information of each thread's performance counters, the scheduler can strategically decide their mapping to cores to improve performance and eliminate resource sharing interference problems [250, 214, 241, 56, 55]. Furthermore, a non-uniform distribution of CPU time slices [80], or fetched instructions [215], can also be used when necessary to improve fairness and overall performance.

In the case of the on-chip interconnect, the abstractness and modularity of the interconnect allows the development of techniques for further increasing performance dependability. As earlier discussed in Section 2.2, a versatile and flexible aspect of an interconnect is the way packet routing is done. In packet routing, performance dependability could be emphasized even through the most basic characteristics. For example, resources could be reserved right from the start, at the source node, so that packets would traverse the network uninterrupted [88], following a form of virtual circuit switching. Non-interference could also be achieved by exclusively scheduling channels based on waves of message domains [236]. Also, packets may be forwarded over multiple routers in a single cycle when there is no interference [126]. Furthermore, by inspecting the routing of a group of packets (instead of a single packet), packets could be prioritized based on lifetime in the network and congestion at their destination nodes [193]. Priority could also be provided during the Switch Arbitration (SA) stage in interconnect routers. SA requests could be prioritized based on Virtual-Channel Allocation (VA) requests which are actually possible future SA requests [44].

Network traversal delay may be reduced by relieving the network from unnecessary traffic. This could be done in a couple of ways. One way is for applications to be clustered based on their network communication intensity [58], to keep applications from having prolonged execution resulting from network congestion. Moreover, routers could keep track of recent requests that were denied, by observing negative acknowledgments. Later on, these routers could respond to other requests to the same destination and reduce the useless traffic traversing the network [249]. Finally, in the case of multicast communication, acknowledgment responses to multicast requests could be reduced in a tree-like manner, instead of having multiple independent unicast responses [140].

At the level of data centers, as the latter scale up to support more powerful online Web services, they also need to deliver service-wide responsiveness with acceptable "tail latency" levels. Tail latency refers to the latency of the slowest requests, and it can be reduced by sending the same request to multiple replicas, so as to use the one that provides smaller delays. Also, by partitioning the data further (micro-partitioning) among the machines, the access time [62] can be reduced and the quality of service improved. Another course of action could be the over-provisioning of resources, where a job could be scheduled on another machine and be re-executed or continue operation from where it left off in the case of timeouts [63].

4.3 Performance and Fault Models

Fault and performance modeling can be considered as a special category of monitors and knobs, which helps designers to make quick design exploration, while taking into account trade offs between the implications of faults on yield, correctness, performance, system configuration, target application workloads, and desired performance goals.

Modeling of faults can be divided into three main categories. The first category explores the effects of faults on yield [229, 125, 150] and is trying to estimate any yield loss based on manufacturing process or field return expectation. The second category investigates the impact of faults on system's correctness. A representative example is the Architectural Vulnerability Factor (AVF) analysis where is trying to quantify the vulnerability of structures in soft errors. Recently, a model has been derived to provide insight and allow designers to quantitatively understand the factors that affect the AVF of a structure [153]. The third category explores the impact of faults on performance. In this case, faults are affecting system's performance, when a faulty structure is disabled, or its capacity is reduced, to maintain correctness. An attempt to calculate the miss ratio that a cache will experience in the presence of faults using fault maps [168] has been made and a model that is able to calculate expected miss ratio in the presence of faults while not requiring any fault maps was also proposed [184]. A first-order analytical model [95] has been proposed, which is able to derive the vulnerability of arrays (architectural and non-architectural) – in terms of performance – given a workload and a probability of failure as well as to derive expected performance degradation, and distributions. For critical real-time embedded systems, a model has been proposed, which is able to derive worst-case execution time in the presence of faults [205], with the aid of probabilistic timing analysis. These fault agnostic performance models can be extended to take in to account performance degradation caused by a fault.

There are performance models though that do not take into account the performance degradation caused by faults. Such models should be able to consider various design decisions that will possibly affect performance. Usually, cache parameters and application behavior [138] are important factors that affect performance and should be explored. For example, a common performance bottleneck in modern processors is the memory bandwidth when multiple processes compete for this resource. Models have been proposed to explore the relationship between bandwidth partitioning schemes and performance objectives, to address the problem. Through this correlation, the model can derive the optimal scheme for different system-level objectives [235]. Since different processes running on the same chip have different requirements, the scheme can be applied to provide performance guarantees for critical applications while the performance for the rest could be maximized in a best-effort manner.

In addition, the combination of memory bandwidth partitioning and scheduling memory accesses with hardware prefetching can improve system performance further, but the effects and interactions should be carefully considered. A proposed model [139] includes a composite prefetching metric that can help determine under which conditions prefetching can improve system performance, along with a bandwidth partitioning model that considers prefetching effects.

Thread performance isolation is also important. To model multi-threaded workloads and analyze their performance, the cycles spent on each event can be recorded into cycle stacks with a simulation-based methodology [97]. Also, based on the stack methodology, a technique has been proposed that is able to quantify the impact of various scaling bottlenecks for multi-threaded workloads running on multi-cores [76]. The technique is applicable both to multi-processors and various forms of multi-threaded architectures. Finally, a model has been proposed to estimate the performance degradation for multi-programmed workloads [226]. By

using workload profiling from single-core runs, the model can characterize performance degradation problems resulting from co-scheduling multi-programmed workloads sharing resources.

The heterogeneous multi-core trend forced the Amdal's law to be adapted accordingly to estimate the speedup of parallel programs running on such architectures [100]. It has been shown that executing a parallel program on an asymmetric processor yields lower performance benefits as predicted by Amdahl's law [78]. Performance models have been produced to explore the heterogeneous multi-core design space by estimating the heterogeneous multi-core performance [227]. Starting from single-core runs, the model enables the designer to quantify heterogeneous architecture performance for a large number (hundreds) of possible job mixes. A model has also been proposed to analyze the input data sensitivity of specialized heterogeneous processors, in terms of performance and energy [40], thus providing better understanding on how different data sets affect the system.

5 Real Commercial Products

This section will provide a summary of key monitors and knobs found in real products and in large-scale warehouse computing setups. The monitors and knobs mentioned here have already been mentioned in the previous sections. In this section, we break down the various techniques based on their fault tolerance, power, and performance management features.

Processor manufacturing companies have exclusive products for the server market segment, which provide a lot of fault tolerance features, known as RAS (Reliability Availability Serviceability). Intel has the Xeon and Itanium series, AMD offers the Opteron series, IBM has the Power Series, and Sun Microsystems offers the SPARC processors [3]. All of these server-processor series have similar RAS features. All models support ECC for DRAMs, for at least one of the cache levels in the memory hierarchy [3], and some models also for registers [109]. In the server DRAM market, we also have products that support multiple errors on two DRAM devices, memory scrubbing, memory DIMM sparing, migration, and memory mirroring [109].

In addition, modern processors are able to identify and disable degrading components – such as cache lines [4] – tag data on corrupted memory locations, use data poisoning, and report hardware errors to the OS (MCA) [109]. This way, the hardware gives the OS the opportunity to react to uncorrectable errors, which normally would lead to a system crash. Furthermore, server systems can even deallocate a whole degrading processor [3], if necessary. In the case of soft-errors, instruction retry and packet retry upon an interconnect error are very common features used for recovery [3].

Next, we will discuss the power management features on modern processors. All modern products are compatible with the ACPI standard, which defines an OS-to-device configuration and power management interface [222]. For example, the power gating and DVFS [105, 18] techniques found in modern devices are defined through this standard. Modern products also offer CPU temperature aware power management [107], as well as memory activity throttling in case of high temperatures [109].

Modern processors not only allow power tuning but they also offer monitoring capabilities both for voltage and power. Fine-grain monitoring of voltage in modern processors can be achieved by on-die voltage sensing pins. For example, the methodology of voltage monitoring for both the Intel Core 2 Duo [176] and AMD Bulldozer [124] is publicly available. Also, fine-grain voltage noise can be measured by on-die voltage droop hardware sensors [108]. The Intel Sandy Bridge architecture provides voltage, temperature, and power consumption readings through a package control unit [155], while development boards based on ARM chips provide voltage, current, and power readings from software-accessed meters and external sensing pins [21].

Regarding performance monitoring and throttling features, most modern processors offer performance counters that give access to events like cache misses, instructions committed, branch miss-predictions, etc. [104]. On-demand over-clocking or turbo mode[106] is a technology that allows a processor to dynamically increase performance by exploiting thermal and power headroom. For example, the AMD Turbo Core technology switches off some cores and increases the frequency of the remaining cores.

Finally, at the datacenter scale, there are techniques for providing and maintaining performance and availability of web services [27]. Server health checking and data replication are used by datacenter operators both

for performance and availability reasons. Sharding or partitioning is another way to increase performance by splitting a problem into many parts and using parallel execution to speed up the response times, e.g., Web Search [28]. Some of these sub-parts can be executed redundantly for tail latency tolerance. In HPC systems and datacenters, the Map-Reduce programming model [63] is used in order executing jobs across a large number of machines. Map-Reduce provides fault-tolerant execution and efficient use of resources.

It is also known that Google's datacenters have performance profiling infrastructure, called Google Wide Profiling (GWP) [179]. GWP continuously records application performance statistics, like data from hardware counters or memory used by applications. Datacenter owners are also concerned about the cost of operating a datacenter and their objective is to use it as efficiently as possible. Many datacenters use virtualization as a way to increase utilization and energy efficiency [90].

Part III
HARPA Monitors and Knobs

1 The HARPA Project

The focus and motivation of the HARPA project is to "bring together teams from embedded computing and high-performance computing to jointly address challenges that are common in these two areas and are magnified by the ubiquity of many-cores and heterogeneity across the whole computing spectrum."

The HARPA project's ultimate goal is to enable next-generation embedded and high-performance heterogeneous many-core processors to provide *dependable performance*, by cost effectively confronting variations. More specifically, *dependable performance* refers to providing correct functionality and timing guarantees throughout the nominal lifetime of a system under thermal, power and energy constrains, and this can be achieved pro-actively (in the absence of hard failures), or re-actively (in the presence of hard faults). The final HARPA product will rely on a cross-layer approach using the pertinent actors from the whole compute stack (i.e, from the circuit level to the software level). The project also aims to shave the conservative timing margins using micro-architectural and middleware techniques.

The heart of the project is the HARPA Engine. The HARPA engine consists of a feedback loop, in which different metrics – such as performance, timing, power, temperature, manifestations of time-dependent variations, etc. – are continuously monitored. Based on the various observations, the HARPA engine actuates the appropriate knobs to bias the execution flow as desired, according to the state of the system and the performance requirements of the application running at the time. The HARPA engine itself is comprised of two key, intertwined components: the Operating System (HARPA-OS) and the Run-Time system (HARPA-RT).

The run-time layer (HARPA-RT) resides at a low level of the compute stack and can directly contact the various monitors and knobs. It is an application-independent layer that has direct and responsive control on hardware resources. Its low-level nature allows the run-time engine to respond at a millisecond time granularity and this enables extremely fast system behavior adaptation, which is mandatory in order to provide guarantees for the hard real-time specifications of an application. Essentially, the HARPA-RT layer is a "firmware" used by the HARPA-OS for accessing the system's monitors and knobs. The OS is expected to maintain a piece of software (e.g., a library) linked with each active application, which provides a set of common services to support its run-time optimization. Note that this OS software is - by construction - semantically closer to the running applications, which allows it to obtain salient semantic information about the run-time applications (such as the phases of the application and its behavior). Being situated at a higher level in the compute stack, the HARPA-OS reacts/adapts to system changes at the second granularity.

The HARPA engine is built atop a heterogeneous multicore architecture, which fits within the context of the emerging multicore computational paradigm, where cores have different performance and power characteristics (low-power, slow cores, and high-performance, fast cores, as well as application-specific accelerators).

2 Monitors and Knobs Applicable to the HARPA Project

In the previous chapters, a multitude of monitors and knobs have been proposed, in an attempt to tackle transient and permanent errors resulting from both time-zero and time dependent variations. The HARPA engine aims to provide performance dependability by initially using proactive techniques in the absence of faults, and later on using reactive techniques when faults appear.

Proactive monitors usually monitor time-dependent variations, which indicate potential future permanent faults resulting from circuit wear-out. In the literature, test structures [118, 117] have been proposed for identifying wear-out effects, such as NBTI, TDDB, and HCI, which are attractive due to their small hardware and power overhead. They can be applicable to both embedded and HPC systems.

Test structures can expose a circuit's decay, based on the operating conditions experienced. This technique, though, does not provide sufficient coverage, since these testing structures do not involve the actual circuit.

Additionally, this technique fails to identify early lifetime failures.

Thus, the HARPA engine also needs the ability to test the underlying hardware. Test patterns should be executed on cores under aggressive operating conditions [149, 81, 206, 216] (i.e., higher frequency and/or lower voltage). This technique, which is applied in HPC systems, is very beneficial to multicore systems. In such systems, the test patterns can be applied to idle cores without affecting the system's performance. Since the HARPA project uses multicore systems, such a solution is attractive. Moreover, a somewhat more indirect solution could be to monitor the utilization of the various resources, e.g., the individual cores, or even the interconnect's routers, to estimate their wear-out [31, 121].

Time-dependent variations can either be delayed or addressed when they appear, by using circuit recovery techniques. At the circuit level, forward body biasing [83, 143] (which can positively affect the DVFS process) and the various design options for SRAM cells [128, 46] (8T and 10T), make the circuit less vulnerable to NBTI effects. Such design decisions are applicable to both worlds, embedded and HPC. Additionally, idle components, both in logic and in the interconnect, can be power gated [228, 52, 199, 254, 169], a technique proposed both for embedded and HPC. This should provide the opportunity to the system to recover from NBTI implications.

The appearance of reliability/aging artifacts can also be delayed by balancing the utilization of the hardware in a certain circuit. Special patterns, or inverted data, can be applied to idle functional units and/or invalid array blocks to reduce the stress from NBTI effects [13, 84]. Furthermore, if redundant hardware is available, it can be used to uniformly stress the system and to recover from NBTI by rotating the computation among the identical components [195]. These techniques are usually applied in the HPC world, due to the higher redundancy available, but they can also be applied to embedded systems, whenever possible. Finally, the notion of balancing the workload throughout the hardware could also be applied at higher levels of the compute stack. At the software level, wear-out aware scheduling techniques can be employed to balance the workload among the cores [216, 221, 81].

Power and thermal management of the system can directly slow down circuit aging, in addition to all the other benefits provided. Based on power and thermal sensors [203, 204, 29], DVFS techniques [238, 43, 17, 66] are frequently used in both HPC and embedded systems, and they aim to reduce thermal and power stress on cores based on their process variability. Cycle stealing [220] is another attractive technique, applied in HPC systems, allowing circuits to operate at a higher frequency than the one determined by the worst-case path. Similarly, in the higher levels of the compute stack, another option is thermal-aware scheduling [54, 69, 98, 204, 53], where threads are scheduled or migrated from hot cores to cold ones, or core-intensive threads are scheduled on cold cores, a technique inspired from heterogeneous and multicore systems. The ACPI interface [19, 45, 47, 167] may be also suitable for the HARPA engine, to help determine different power states of the system, based on application requirements.

Fault detection may be applied at different granularities and be orchestrated together with power management for optimizing DVFS. Functional units can be monitored using residue codes [157, 208], arrays can be monitored and protected using ECC codes [93, 102, 137, 177, 141], and interconnects may rely on invariance violation monitoring [171, 247]. In addition, to perform control and data flow fault detection, dynamic and static comparisons of flow graphs can be used [145, 178]. The increased overhead of this technique, however, renders it inapplicable to low-end systems. At higher levels of the compute stack, monitoring for anomalous behavior [134, 96, 234] is also an option for complementing lower-level monitoring. Next, upon a fault detection, with the help of architectural replay and check-pointing [158, 207, 50, 165, 92, 57, 246, 70], the system (embedded or HPC) can recover from the erroneous state. Faults can also be tolerated by using scrubbing [200, 142] in arrays to correct possible faults and better avoid uncorrectable errors.

Another technique, applied periodically, is BIST checking [131, 230, 114, 68], which can be used in order to identify any permanent faults. If faults are detected repeatedly at the same location in hardware, then this may be an indication of a permanent fault. When a permanent fault is detected, the most obvious action would be to disable the faulty component and replace it with a spare [4, 35]. In the absence or shortage of spares the HARPA engine must be able to minimize the impact of the fault, in terms of yield and performance, without compromising the system's correctness. A very attractive approach is to try to avoid the faulty component

within the core. Furthermore, if the cores are multi threaded, threads could be migrated to another core when the core they are assigned to becomes unable to execute them [170, 112]. Adaptive routing [73, 175, 86] or routing reconfiguration [213, 131, 161, 68, 16, 160, 172] is also an interesting solution for avoiding faulty hardware in the interconnect. Another solution is to disable the component that contains the fault at the finest granularity possible (e.g., disable only a block from a cache, retire a DRAM page, make a functional unit narrower, etc.) All of the above techniques come from the HPC domain, where multi-cores are widely used under more relaxed, as compared to embedded, constrains, and where some form of redundancy typically exists. However, the underlying concepts of these techniques can also be applied to embedded systems, albeit within a different context and using different assumptions.

Based on application requirements and criticality, HARPA cores can also be used in a Triple Modular Redundancy (TMR) fashion [231, 243, 42, 74, 114, 130, 15] in order to provide the desired performance. When no other way is available for circumventing a faulty component, emulation of instructions may be applied to remove the need of using a certain functionality [146, 112].

To provide performance dependability, all of the above solutions (proactive and reactive) must be evaluated and their efficiency must be estimated. A way of estimating the effectiveness of a solution is by using performance counters [110, 6, 79] to collect information at run-time. Similarly, watchdog counters [26, 145] are also a solution that provides timing guarantees. Moreover, to provide performance isolation, the scheduler must be able to monitor the behavior of each thread [214], and this can be achieved by statically or dynamically profiling threads and collecting statistics like memory-level parallelism and CPI.

Thread scheduling can also be used as a mechanism to provide performance guarantees. Scheduling, for example, can be used either for load balancing between the cores, or to prevent performance degradation in the cases where resource-hungry threads compete for the same resources [30]. Concerning the NoC, network-state observations and other pertinent information can be used to either prioritize traffic [193], or re-distribute traffic in a more optimized fashion [126].

Aside from the above proposed monitors and knobs, the HARPA project must also rely on models that will assist architects in design time decisions, and will be attached on the HARPA OS for run-time estimations. One of the most relevant models to the HARPA project is the PVF model [95] which is able to model the implications of permanent faults in terms of performance. This model must be extended in order to not only cover time zero permanent faults but also time dependent in order to analyze the performance variability that a processor will experience during its lifetime. Cache vulnerability to permanent faults in terms of expected miss rate [184] is a suitable metric in the scope of the HARPA project. Performance models that have been proposed in Section 4.3 can also be used given they are extended by taking into account the possible presence of faults.

Part IV
Appendix

This appendix presents – with the help of a table – a concise summary of all the investigated monitors and knobs. The table also highlights the different categories and features of the various monitors and knobs.

***Market Segment**: **D** (Datacenter), **H** (HPC), **E** (Embedded)*
***Applicability**: **T** (Transient), **P** (Permanent), **Pe** (Performance),**Po** (Power), **TD** (Time Dependent), **TZ** (Time Zero)*
***Layer**: **Fi** (Firmware), **Mi** (Micro-architecture), **OS** (Operating System), **Ma** (Manufacture), **In** (Infastructure), **Ci** (Circuit), **App** (Application)*

Technique	Monitor/ Knob	Layer	Real Product	Proactive/ Reactive	Market Segment	Applicability
Parity & ECC/EDC [93, 102, 137, 177]	M/K	Mi	Yes	R	D/H/E	T/P
Check On Writes[156]	M	Mi	No	R	D/	T/P
CPPC [141]	M/K	Mi	No	R	D/	T/P
Two dimensional parity [122]	M/K	Mi	No	R	D/	T/P
Redundant encoding [187]	K	Mi	No	R	D/H/E	T
Chipkill[65, 1, 10]	M/K	Mi	Yes	R	D/H/E	T
VECC[245]	M/K	Mi	No	R	D/H/E	T/P
LOT-ECC[224]	M/K	Mi	No	R	D/H/E	T/P
Map-Reduce[64, 63]	K	App	Yes	R	D/H	T/P/TD
Argus [145]	M	Mi	No	R	D/H/E	T
ECC encoding [91, 187]	M/K	Mi	No	R	D/H/E	T/P
Scrubbing [200, 142]	M	Mi	Yes	P	D/H/E	T/P
Poison bit [237]	M	Mi	Yes	R	D/H/E	T/P
NMR(DMR/TMR) [231, 243, 42, 74, 114, 130, 15]	M/K	Mi	Yes	R	D/H/E	T/P
SMT [182, 191]	M	OS/App	Yes	R	D/H/E	T
Idempotent tasks [147, 61, 123]	M	OS/App	No	R	D/H/E	T
DIVA [147]	M	Ci	No	R	D/H	T
Residue codes [157, 208]	M	Mi	No	R	D/H/E	T/P
SWIFT [178]	M	OS/App	No	R	D/H	T/P
ReStore [234]	M	Mi/OS	No	R	D/H	T
Functional unit checker [244]	M	Mi	No	R	D/H/E	T/P
Checkpoint/ Rollback [158, 207, 50, 165, 92, 57, 246, 70]	K	OS/App	Yes	R	D/H	T
RESO [163]	M	Mi	No	R	D/H	P
SRAS [35]	M/K	Mi	No	R	D/H	P
MCA [11, 2, 109]	M	OS	Yes	R	D/H/E	T/P
MCA recovery [4]	K	OS	Yes	R	D/H/E	T/P
Log files [4, 5]	M	OS	Yes	P	D/H/E	T/P/Pe/Po/TD
Page retirement [217]	K	Mi/OS	Yes	R	D/H	P
Memory spares [35, 9, 7]	K	Ma/Mi/In	Yes	R	D/H/E	P
Proactively use of spares [195]	K	Mi	No	P	D/H	TD
Intel Cache Safe technology [4]	K	OS/In	Yes	R	D/H	P
Memory mirroring [9, 7]	K	Mi/In	Yes	R	D/H	P
Replication [26]	K	Ma/Mi/In	Yes	R	D/H	P/Pe
Data partitioning [26]	K	OS	Yes	R	D/H	P/Pe
SWAT [134, 96]	M	App	No	R	D/H	P
Fault screening [174]	M	App	No	R	D/H	T/P
Core Cannibalization [181]	K	Mi	No	R	D/H	P
Microarchitectural redundancy [196]	K	Mi	No	R	D/H	P
Core Salvaging [170, 112]	K	Fi/Ci	No	R	D/H	P
Detouring [146]	K	OS	No	R	D/H	P
Memory DVFS [66]	K	Mi/Ci	No	R	D/H/E	Po
Power states [19, 45, 47, 167, 222]	K	OS	Yes	R	D/H	Pe/Po
Performance counters [110, 6, 79]	M	Mi	Yes	R	D/H/E	Pe
Watchdog timers [26, 145]	M	Mi	Yes	R	D/H/E	Pe
Tail tolerance [62]	K	App	Yes	R	D	Pe
Map-Reduce [63]	M/K	App	Yes	R	D/H	P/Pe
RAMP [211]	M	Ci/OS	No	P	D/H/E	TD
Test Circuit under aggressive conditions [149, 81, 206, 216]	M	Ci/OS	No	P	D/H	TD
Measure Signal Propagation [173, 33]	M	Ci	No	P	D/H	TD
IDDQ measurement [116, 84, 68]	M	Ci	No	P	D/H/E	TD
Test structures [118, 117]	M	Ci	No	P	D/H/E	TD

***Market Segment**: **D** (Datacenter), **H** (HPC), **E** (Embedded)*
***Applicability**: **T** (Transient), **P** (Permanent), **Pe** (Performance),**Po** (Power), **TD** (Time Dependent), **TZ** (Time Zero)*
***Layer**: **Fi** (Firmware), **Mi** (Micro-architecture), **OS** (Operating System), **Ma** (Manufacture), **In** (Infastructure), **Ci** (Circuit), **App** (Application)*

Technique	Monitor/ Knob	Layer	Real Product	Proactive/ Reactive	Market Segment	Applicability
Canary circuits [232, 233, 186]	M	Ci	No	P	D/H/E	Po/TD
Forward Body Biasing [83, 143]	K	Ma/Ci	No	P	D/H/E	Pe/Po/TD/TZ
Power-gating [228, 52, 199, 254, 169]	K	Ci	Yes	P	D/H/E	Po/TD
Apply Specific Patterns on Circuit [13, 84, 197]	K	Mi	No	P	D/H	TD
8T and 10T SRAM cell designs [128, 46]	K	Ma/Ci	No	P	D/H/E	Po/TD/TZ
Ageing Aware scheduling [221, 81]	K	OS	No	P	D/H	TD
RazorII [60]	M/K	Ci	No	R	D/H/E	T/Po/TD/TZ
ReCycle [219]	K	Ci	No	R	D/H/E	Pe/Po/TD/TZ
EVAL [185]	K	OS	No	R	D/H	Pe/Po/TD/TZ
ReVIVaL [136]	K	Mi	No	R	D/H	Pe/TD/TZ
Yield Analysis [229, 125, 150]	M	Ma	No	P	D/H/C	P
Memory Bandwidth Partitioning Model [235]	M/K	Ma/Mi	No	P	D/H	Pe
Hardware Prefetching and Bandwidth Partitioning Model [139]	M/K	Ma/Mi	No	P	D/H	Pe
Context Switch Misses Model [138]	M/K	Ma/Mi	No	P	D/H	Pe
Simulation-Based Parallel Performance Analysis [97]	M/K	Ma/Mi	No	P	D/H	Pe
SpeedUp Stacks [76]	M/K	Ma/Mi	No	P	D/H	Pe
Multi-Program Performance Model [226, 78]	M/K	Ma/OS	No	P	D/H	Pe
Single-ISA heterogeneous architectures Model [227]	M/K	Ma/Mi	No	P	D/H	Pe
Data Input Sensitivity of Specialized Processors Model [40]	M/K	Ma/Mi	No	P	E	Pe
AVF analytical model [153]	M/K	Ma/Mi/Ci	No	P	D/H	P/Pe
PVF Model [95]	M/K	Ma/Mi/Ci	No	P	D/H	P/Pe
Cache Miss Ratio Model [184]	M/K	Ma/Mi/Ci	No	P	D/H	P/Pe
Fault-Aware Probabilistic Timing Analysis [205]	M/K	Ma/Mi/Ci	No	P	E	P/Pe
Thermal Sensors [203, 204]	M	Ci	Yes	P/R	D/H	Pe/Po/TD
Thermal Management Techniques [69, 43, 162]	M/K	Ci	No	P	D/H/E	Pe/Po/TD
Thermal Coupling Techniques Prediction [51]	M	Ci	No	P	D/H/E	Po/TD
Power,Thermal and Performance management on Heterogeneous multi-cores [164]	M/K	Ci/Mi	No	P	D/H/E	Pe/Po/TD
Frequency Controller [248]	M/K	Ci	No	P	D/H	Pe/Po/TD
Memory Accesses and Footprint for power management [23]	M	OS	No	P	E	Po/TD
ISA characterization [38]	M/K	OS	No	P	E	Po/TD
Power Consumption and Energy Availability [29]	M	Ci	No	P	E	Po
DVFS [238, 43, 17]	K	Ci	Yes	P	D/H/E	Pe/Po/TD
Flicker [166]	K	Ci	No	P	D/H/E	Pe/Po/TD
PowerDial [101]	K	OS	No	P	D/H	Pe/Po/TD
Consolidate workloads on a subset of servers [45, 47, 167]	K	In	No	P	D/H	Po/TD
DVS [218, 133, 25]	K	Ci	Yes	P	D/H/E	Po/TD
Thermal Aware Thread Migration [54, 69, 98, 204, 53]	K	Mi/OS	No	P	D/H	Po/TD
Energy and thermal management in heterogeneous embedded multiprocessor SoCs [194]	K	Mi/OS	No	P	E	Pe/Po/TD
Effective Run-time Resource Management [30, 39]	K	Mi/OS	No	P	E	Pe/Po/TD
Invariance Checking [171, 247]	M	Mi	N	R	D/H/E	T/P
Built-in Self-Test [131, 230, 114, 68]	M	Mi	N	R	D/H	P
ECC Syndromes [87, 192, 161]	M	App	N	R	D/H	P
Architectural Reconfiguration [115, 165]	K	Mi	N	R	D/H	P
Routing Reconfiguration [213, 131, 161, 68, 16, 160, 172]	K	Mi	N	R	D/H/E	P
Adaptive Routing [73, 175, 86]	K	Mi	N	R	D/H	P/Pe/TD
Resource Utilization [31, 121]	M	Mi	N	P	D/H/E	Pe/TD
Secondary Control Network [12, 159]	M/K	Mi	N	R	D/H	T/P
Packet Prioritization [88, 193, 44]	K	App	Y	P	D/H/E	Pe
Application clustering [58, 140]	K	App	N	P	D/H	Pe
Catnap [59]	M/K	Mi	N	P	D/H	P/Po

References

[1] BIOS and Kernel Developers Guide (BKDG) for AMD Family 10h. April 2010.

[2] Debugging machine check exceptions on embedded ia platforms. July 2010.

[3] Technical white paper,mainframe-class availability at one-eighth the total cost of ownership (tco). HP, february 2010.

[4] Technical white paper,ras features of the mission-critical converged infrastructure. HP, June 2010.

[5] Hp advanced memory error detection technology. July 2011.

[6] Intel performance counter monitor - a better way to measure cpu utilization. August 2012.

[7] Technical white paper, avoiding server downtime from hardware errors in system memory with hp memory quarantine. HP, January 2012.

[8] The international technology roadmap for semiconductors, technical report, edition. 2013.

[9] Technical white paper, how memory ras technologies can enhance the uptime of hp proliant servers. April 2013.

[10] Bios and kernel developers guide (bkdg) for amd family 15h. February 2014.

[11] Intel 64 and ia-32 architectures software developers manual volume 3 (3a, 3b, 3c): System programming guide. February 2014.

[12] Rawan Abdel-Khalek, Ritesh Parikh, Andrew DeOrio, and Valeria Bertacco. Functional correctness for cmp interconnects. In *ICCD*, pages 352–359. IEEE, 2011.

[13] J. Abella, X. Vera, and A. Gonzalez. Penelope: The nbti-aware processor. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 85–96, Dec 2007.

[14] Alexander Acovic, Giuseppe La Rosa, and Yuan-Chen Sun. A review of hot-carrier degradation mechanisms in MOSFETs. *Microelectronics Reliability*, 36(78):845 – 869, 1996. Reliability Physics of Advanced Electron Devices.

[15] Nidhi Aggarwal, Parthasarathy Ranganathan, Norman P. Jouppi, and James E. Smith. Configurable isolation: Building high availability systems with commodity multi-core processors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 470–481, 2007.

[16] Konstantinos Aisopos, Andrew DeOrio, Li-Shiuan Peh, and Valeria Bertacco. Ariadne: Agnostic reconfiguration in a disconnected network environment. In Lawrence Rauchwerger and Vivek Sarkar, editors, *PACT*, pages 298–309. IEEE Computer Society, 2011.

[17] M Altieri, W Lombardi, D Puschini, and S Lesecq. Coupled voltage and frequency control for dvfs management. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013 23rd International Workshop on*, pages 207–214. IEEE, 2013.

[18] AMD. AMD PowerNow! Technology. *Techical White Paper*, 2000.

[19] Vlasia Anagnostopoulou, Susmit Biswas, Heba Saadeldeen, Alan Savage, Ricardo Bianchini, Tao Yang, Diana Franklin, and Frederic T. Chong. Barely alive memory servers: Keeping data active in a low-power state. *J. Emerg. Technol. Comput. Syst.*, 8(4):31:1–31:20, November 2012.

[20] Dean Michael Ancajas, James McCabe Nickerson, Koushik Chakraborty, and Sanghamitra Roy. Hci-tolerant noc router microarchitecture. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 40:1–40:10, New York, NY, USA, 2013. ACM.

[21] ARM. Coretile express a15x2 a7x3 cortex-a15-a7 mpcore (v2p-ca15-a7) technical refrence manual. 2012.

[22] A. Asenov, A.R. Brown, J.H. Davies, S. Kaya, and G. Slavcheva. Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale mosfets. *Electron Devices, IEEE Transactions on*, 50(9):1837 – 1852, sept. 2003.

[23] David Atienza, Stylianos Mamagkakis, Francesco Poletti, Jose M. Mendias, Francky Catthoor, Luca Benini, and Dimitrios Soudris. Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems. *Integration, the VLSI Journal*, 39(2):113–130, March 2006.

[24] Todd M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd International Symposium on Microarchitecture*, pages 196–207, November 1999.

[25] Anys Bacha and Radu Teodorescu. Dynamic reduction of voltage margins by leveraging on-chip ecc in itanium ii processors. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 297–307, New York, NY, USA, 2013. ACM.

[26] Luiz Andr Barroso, Jimmy Clidaras, and Urs Hlzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.

[27] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.

[28] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. Web search for a planet: The google cluster architecture. *Micro, Ieee*, 23(2):22–28, 2003.

[29] Dimitris Bekiaris, Ioannis Kosmadakis, George I. Stassinopoulos, Dimitrios Soudris, Theodore Laopou-

los, Gregory Doumenis, and Stylianos Siskos. Run-time measurement of harvested energy for autarkic sensor operation. In Jos L. Ayala, Delong Shang, and Alex Yakovlev, editors, *PATMOS*, volume 7606 of *Lecture Notes in Computer Science*, pages 185–193. Springer, 2012.

[30] P Bellasi, G Massari, and W Fornaciari. Exploiting linux control groups for effective run-time resource management. In *PARMA 2013 Workshop HiPEAC 2013, Jan 2013, Berlin, Germany*, 2013.

[31] Kshitij Bhardwaj, Koushik Chakraborty, and Sanghamitra Roy. Towards graceful aging degradation in nocs through an adaptive routing algorithm. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 382–391, New York, NY, USA, 2012. ACM.

[32] A.J. Bhavnagarwala, X. Tang, and J.D. Meindl. The impact of intrinsic device fluctuations on cmos sram cell stability. *Solid-State Circuits, IEEE Journal of*, 36(4):658–665, Apr 2001.

[33] J. Blome, Shuguang Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 109–122, Dec 2007.

[34] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference, 2003. Proceedings*, pages 338–342, June 2003.

[35] Fred A. Bower, Paul G. Shealy, Sule Ozev, and Daniel J. Sorin. Tolerating hard faults in microprocessor array structures. In *Proceedings of the 34th International Conference on Dependable Systems and Networks*, pages 51–60, June 2004.

[36] K. Bowman, J. Tschanz, C. Wilkerson, Shih-Lien Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 4–7, July 2009.

[37] K.A. Bowman, S.G. Duvall, and J.D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Solid-State Circuits, IEEE Journal of*, 37(2):183 –190, feb 2002.

[38] Carlo Brandolese and William Fornaciari. Software energy optimization through fine-grained function-level voltage and frequency scaling. In Ahmed Jerraya, Luca P. Carloni, Naehyuck Chang, and Franco Fummi, editors, *CODES+ISSS*, pages 539–546. ACM, 2012.

[39] Carlo Brandolese, William Fornaciari, Luigi Rucco, and Federico Terraneo. Enabling ultra-low power operation in high-end wireless sensor networks nodes. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 433–442. ACM, 2012.

[40] M. Breughe, Zheng Li, Yang Chen, S. Eyerman, O. Temam, C. Wu, and L. Eeckhout. How sensitive is processor customization to the workload's input datasets? In *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pages 1–7, June 2011.

[41] Eric A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, July 2001.

[42] D. Briere and P. Traverse. Airbus a320/a330/a340 electrical flight controls - a family of fault-tolerant systems. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 616–623, June 1993.

[43] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 171–182. IEEE, 2001.

[44] Yuan-Ying Chang, Yoshi Shih-Chieh Huang, Matthew Poremba, Vijaykrishnan Narayanan, Yuan Xie, and Chung-Ta King. Ts-router: On maximizing the quality-of-allocation in the on-chip network. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 390–399, Washington, DC, USA, 2013. IEEE Computer Society.

[45] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.

[46] G. Chen, D Sylvester, D. Blaauw, and T. Mudge. Yield-driven near-threshold sram design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(11):1590–1598, Nov 2010.

[47] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[48] K.-L. Chen, S.A. Saller, I.A. Groves, and D.B. Scott. Reliability effects on mos transistors due to hot-carrier injection. *Solid-State Circuits, IEEE Journal of*, 20(1):306–313, Feb 1985.

[49] J.J. Clement. Electromigration modeling for integrated circuit interconnect reliability analysis. *Device and Materials Reliability, IEEE Transactions on*, 1(1):33–42, Mar 2001.

[50] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. Bulletproof: A defect-tolerant cmp switch architecture. In *In Proceedings of the 12th International Symposium on High Performance Computer Architecture*, pages 3–14,

2006.
[51] Simone Corbetta and William Fornaciari. Exploiting thermal coupling information in mpsoc dynamic thermal management. In *Architecture of Computing Systems–ARCS 2013*, pages 50–61. Springer, 2013.
[52] Simone Corbetta and William Fornaciari. Performance/reliability trade-off in superscalar processors for aggressive nbti restoration of functional units. In *Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI*, GLSVLSI '13, pages 221–226, 2013.
[53] Ayse Kivilcim Coskun, José L. Ayala, David Atienza, Tajana Simunic Rosing, and Yusuf Leblebici. Dynamic thermal management in 3d multicore architectures. In *DATE*, pages 1410–1415, 2009.
[54] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Keith Whisnant. Temperature aware task scheduling in mpsocs. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1659–1664, San Jose, CA, USA, 2007. EDA Consortium.
[55] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout. Fairness-aware scheduling on single-isa heterogeneous multi-cores. In *PACT*, pages 177–187. IEEE, 2013.
[56] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvez, and Joel S. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ISCA*, pages 213–224. IEEE, 2012.
[57] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 2006.
[58] Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi. Application-to-core mapping policies to reduce memory interference in multi-core systems. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 455–456, New York, NY, USA, 2012. ACM.
[59] Reetuparna Das, Satish Narayanasamy, Sudhir Satpathy, and Ronald G. Dreslinski. Catnap: energy proportional multiple network-on-chip. In Avi Mendelson, editor, *ISCA*, pages 320–331. ACM, 2013.
[60] S. Das, C. Tokunaga, S. Pant, W. H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, January 2009.
[61] Marc de Kruijf and Karthikeyan Sankaralingam. Idempotent processor architecture. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 140–151, New York, NY, USA, 2011. ACM.
[62] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.
[63] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.
[64] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13, 2008.
[65] Timothy J Dell. A white paper on the benefits of chipkill-correct ecc for pc server main memory. *IBM Microelectronics Division*, pages 1–23, 1997.
[66] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. Memscale: Active low-power modes for main memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM.
[67] Andrew DeOrio, Kostantinos Aisopos, Valeria Bertacco, and Li-Shiuan Peh. Drain: Distributed recovery architecture for inaccessible nodes in multi-core chips. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 912–917, New York, NY, USA, 2011. ACM.
[68] Andrew DeOrio, David Fick, Valeria Bertacco, Dennis Sylvester, David Blaauw, Jin Hu, and Gregory K. Chen. A reliable routing architecture and algorithm for nocs. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(5):726–739, 2012.
[69] James Donald and Margaret Martonosi. Techniques for multicore thermal management: Classification and new exploration. *SIGARCH Comput. Archit. News*, 34(2):78–88, 2006.
[70] Xiangyu Dong, Naveen Muralimanohar, Norm Jouppi, Richard Kaufmann, and Yuan Xie. Leveraging 3d pcram technologies to reduce checkpoint overhead for future exascale systems. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 57:1–57:12, New York, NY, USA, 2009. ACM.
[71] M. Duranton, D. Black-Schaffer, K. De Bosschere, and J. Maebe. The HiPEAC vision for Advanced Computing in Horizon 2020. 2013.
[72] M. Duranton, D. Black-Schaffer, S. Yehia, and K. De Bosschere. Computing Systems: Research Challenges Ahead The HiPEAC Vision 2011/2012. 2011.
[73] M. Ebrahimi, M. Daneshtalab, J. Plosila, and F. Mehdipour. Md: Minimal path-based fault-tolerant routing in on-chip networks. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 35–40, Jan 2013.
[74] Christian Engelmann, Hong Ong, and Stephen L Scott. The case for modular redundancy in large-scale

high performance computing systems. In *Proceedings of the IASTED International Conference*, volume 641, page 046, 2009.
[75] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. pages 7–18, December 2003.
[76] Stijn Eyerman, Kristof Du Bois, and Lieven Eeckhout. Speedup stacks: Identifying scaling bottlenecks in multi-threaded applications. In *ISPASS*, pages 145–155, 2012.
[77] Stijn Eyerman and Lieven Eeckhout. Per-thread cycle accounting in smt processors. *SIGPLAN Not.*, 44(3):133–144, March 2009.
[78] Stijn Eyerman and Lieven Eeckhout. Modeling critical sections in amdahl's law and its implications for multicore design. *SIGARCH Comput. Archit. News*, 38(3):362–370, June 2010.
[79] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. A performance counter architecture for computing accurate cpi components. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 175–184, New York, NY, USA, 2006. ACM.
[80] Alexandra Fedorova, Margo Seltzer, and Michael D. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. *Parallel Architectures and Compilation Techniques, International Conference on*, 0:25–38, 2007.
[81] Shuguang Feng, Shantanu Gupta, and Scott Mahlke. Olay: Combat the signs of aging with introspective reliability management. In *Workshop on Quality-Aware Design (W-QUAD)*, June 2008.
[82] Farshad Firouzi, Saman Kiamehr, Mehdi Tahoori, and Sani Nassif. Incorporating the impacts of workload-dependent runtime variations into timing analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1022–1025, March 2013.
[83] J. Franco, B. Kaczer, M. Toledano-Luque, P.J. Roussel, G. Groeseneken, B. Schwarz, M. Bina, M. Waltl, P.-J. Wagner, and T. Grasser. Reduction of the bti time-dependent variability in nanoscaled mosfets by body bias. In *Reliability Physics Symposium (IRPS), 2013 IEEE International*, pages 2D.3.1–2D.3.6, April 2013.
[84] Xin Fu, Tao Li, and J. Fortes. Nbti tolerant microarchitecture design in the presence of process variation. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 399–410, Nov 2008.
[85] Xin Fu, Tao Li, and Jos A. B. Fortes. Architecting reliable multi-core network-on-chip for small scale processing technology. In *DSN*, pages 111–120. IEEE, 2010.
[86] Amlan Ganguly, Paul Wettin, Kevin Chang, and Partha Pande. Complex network inspired fault-tolerant noc architectures with wireless links. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, pages 169–176, New York, NY, USA, 2011. ACM.
[87] Amirali Ghofrani, Ritesh Parikh, Saeed Shamshiri, Andrew DeOrio, Kwang-Ting Cheng, and Valeria Bertacco. Comprehensive online defect diagnosis in on-chip networks. In *VTS*, pages 44–49. IEEE, 2012.
[88] Kees Goossens, John Dielissen, and Andrei Radulescu. ÆThereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, September 2005.
[89] C. Grecu, A. Ivanov, R. Saleh, and P.P. Pande. Noc interconnect yield improvement using crosspoint redundancy. In *Defect and Fault Tolerance in VLSI Systems, 2006. DFT '06. 21st IEEE International Symposium on*, pages 457–465, Oct 2006.
[90] The Green Grid. Impact of virtualization on data center physical infrastructure. *Techical White Paper*, 2010.
[91] Richard H. Gumpertz. Combining tags with error codes. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*, ISCA '83, pages 160–165, New York, NY, USA, 1983. ACM.
[92] Thomas J. Hacker, Fabian Romero, and Christopher D. Carothers. An analysis of clustered failures on large supercomputing systems. *J. Parallel Distrib. Comput.*, 69(7):652–665, July 2009.
[93] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.
[94] Damien Hardy, Isidoros Sideris, Nikolas Ladas, and Yiannakis Sazeides. The performance vulnerability of architectural and non-architectural arrays to permanent faults. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 48–59, Washington, DC, USA, 2012. IEEE Computer Society.
[95] Damien Hardy, Isidoros Sideris, Nikolas Ladas, and Yiannakis Sazeides. The performance vulnerability of architectural and non-architectural arrays to permanent faults. In *Proceedings of the 45th annual IEEE/ACM International Symposium on Microarchitecture*, 2012.
[96] S.K.S. Hari, Man-Lap Li, P. Ramachandran, Byn Choi, and S.V. Adve. mswat: Low-cost hardware fault detection and diagnosis for multicore systems. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 122–132, Dec 2009.

[97] W. Heirman, T.E. Carlson, Shuai Che, K. Skadron, and L. Eeckhout. Using cycle stacks to understand scaling bottlenecks in multi-threaded workloads. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 38–49, Nov 2011.
[98] Seongmoo Heo, Kenneth Barr, and Krste Asanovic. Reducing power density through activity migration. In *In Proceedings of the International Symposium on Low Power Electronics and Design*, pages 217–222, 2003.
[99] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993.
[100] M.D. Hill and M.R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, July 2008.
[101] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices*, 47(4):199, June 2012.
[102] M.Y. Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM Journal of Research and Development*, 14(4):395–401, July 1970.
[103] V. Huard and M. Denais. Hole trapping effect on methodology for dc and ac negative bias temperature instability measurements in pmos transistors. In *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International*, pages 40–45, April 2004.
[104] Intel. Intel nehalem performance monitoring unit programmiong guide.
[105] Intel. Enhanced intel speedstep technology for the intel pentium m processor. *Techical White Paper*, 2004.
[106] Intel. Intel Turbo Boost Technology in Intel Core microarchitecture (Nehalem) based processors. White paper. Technical report, November 2008.
[107] Intel. Cpu monitoring with dts/peci. *Techical White Paper*, 2010.
[108] Intel. A 32 nm, 3.1 billion transistor, 12 wide issue itanium processor for mission-critical servers. *Techical White Paper*, 2011.
[109] Intel. Intel xeon processor e7 family: Reliability, availability, and serviceability. *Techical White Paper*, 2011.
[110] Michael Isard. Autopilot: Automatic data center management. *SIGOPS Oper. Syst. Rev.*, 41(2):60–67, April 2007.
[111] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, pages 79–90, Feb 2003.
[112] Russ Joseph. Exploring salvage techniques for multi-core architectures. In *In Proceedings of the 2nd Workshop on High Performance Computing Reliability Issues*, 2006.
[113] F. J. Aichelmann Jr. Fault-tolerant design techniques for semiconductor memory applications. *IBM Journal of Research and Development*, 28(2):177–183, 1984.
[114] M.R. Kakoee, V. Bertacco, and L. Benini. A distributed and topology-agnostic approach for on-line noc testing. In *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, pages 113–120, May 2011.
[115] M.R. Kakoee, V. Bertacco, and L. Benini. Relinoc: A reliable network for priority-based on-chip communication. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.
[116] Kunhyuk Kang, Keejong Kim, A.E. Islam, M.A. Alam, and K. Roy. Characterization and estimation of circuit reliability degradation under nbti using on-line iddq measurement. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 358–363, June 2007.
[117] E. Karl, P Singh, D. Blaauw, and D Sylvester. Compact in-situ sensors for monitoring negative-bias-temperature-instability effect and oxide degradation. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 410–623, Feb 2008.
[118] John Keane, Tae hyoung Kim, Xiaofei Wang, and Chris H. Kim. On-chip reliability monitors for measuring circuit degradation. *Microelectronics Reliability*, 50(8):1039 – 1053, 2010.
[119] C.N. Keltcher, K.J. McGrath, A. Ahmed, and P. Conway. The amd opteron processor for multiprocessor servers. *Micro, IEEE*, 23(2):66–76, March 2003.
[120] A. Kerber, M. Rohner, T. Pompl, R. Duschl, and M. Kerber. Lifetime prediction for cmos devices with ultra thin gate oxides based on progressive breakdown. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 217–220, April 2007.
[121] Hyungjun Kim, Arseniy Vitkovskiy, Paul V. Gratz, and Vassos Soteriou. Use it or lose it: Wear-out and lifetime in future chip multiprocessors. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 136–147, New York, NY, USA, 2013. ACM.
[122] Jangwoo Kim, N. Hardavellas, Ken Mai, B. Falsafi, and J.C. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 197–209, Dec 2007.
[123] Seon Wook Kim, Chong-Liang Ooi, Rudolf Eigenmann, Babak Falsafi, and T. N. Vijaykumar. Exploit-

ing reference idempotency to reduce speculative storage overflow. *ACM Trans. Program. Lang. Syst.*, 28(5):942–965, September 2006.
[124] Youngtaek Kim, Lizy Kurian John, Sanjay Pant, Srilatha Manne, Michael Schulte, W Lloyd Bircher, and Madhu S Sibi Govindan. Audit: Stress testing the automatic way. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 212–223. IEEE, 2012.
[125] I. Koren, Z. Koren, and C.H. Stepper. A unified negative-binomial distribution for yield analysis of defect-tolerant circuits. *Computers, IEEE Transactions on*, 42(6):724–734, Jun 1993.
[126] Tushar Krishna, Chia-Hsin Owen Chen, Woo Cheol Kwon, and Li-Shiuan Peh. Breaking the on-chip latency barrier using smart. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 378–389, Washington, DC, USA, 2013. IEEE Computer Society.
[127] A.T. Krishnan and P.E. Nicollian. Analytic extension of the cell-based oxide breakdown model to full percolation and its implications. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 232–239, April 2007.
[128] S.K. Krishnappa and H. Mahmoodi. Comparative bti reliability analysis of sram cell designs in nano-scale cmos technology. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pages 1–6, March 2011.
[129] N. Ladas, Y. Sazeides, and V. Desmet. Performance Implications of Faults in Prediction Arrays. In *2nd HiPEAC Workshop on Design for Reliability*, 2010.
[130] C. LaFrieda, E. Ipek, J.F. Martinez, and R. Manohar. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 317–326, June 2007.
[131] Doowon Lee, Ritesh Parikh, and Valeria Bertacco. Brisk and limited-impact noc routing reconfiguration. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.
[132] Kyeong-Jae Lee and Kevin Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE, 2005.
[133] Charles R. Lefurgy, Alan J. Drake, Michael S. Floyd, Malcolm S. Allen-Ware, Bishop Brock, Jose a. Tierno, and John B. Carter. Active management of timing guardband to save energy in POWER7. *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*, page 1, 2011.
[134] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 265–276, 2008.
[135] Ping-Chung Li and T.K. Young. Electromigration: the time bomb in deep-submicron ics. *Spectrum, IEEE*, 33(9):75–78, Sep 1996.
[136] Xiaoyao Liang, Gu-Yeon Wei, and David Brooks. Revival: A variation-tolerant architecture using voltage interpolation and variable latency. pages 191–202, June 2008.
[137] Shu Lin and Daniel J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983. TUB-HH 2413-469 3.
[138] Fang Liu and Yan Solihin. Understanding the behavior and implications of context switch misses. *ACM Trans. Archit. Code Optim.*, 7(4):21:1–21:28, December 2010.
[139] Fang Liu and Yan Solihin. Studying the impact of hardware prefetching and bandwidth partitioning in chip-multiprocessors. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 37–48. ACM, 2011.
[140] Sheng Ma, Natalie D. Enright Jerger, and Zhiying Wang. Supporting efficient collective communication in nocs. In *HPCA*, pages 165–176. IEEE, 2012.
[141] Mehrtash Manoochehri, Murali Annavaram, and Michel Dubois. Cppc: Correctable parity protected cache. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 223–234, New York, NY, USA, 2011. ACM.
[142] Riccardo Mariani and Gabriele Boschi. Scrubbing and partitioning for protection of memory systems. In *IOLTS*, pages 195–196, 2005.
[143] S.M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*, 2002.
[144] T.C. May and Murray H. Woods. Alpha-particle-induced soft errors in dynamic memories. *Electron Devices, IEEE Transactions on*, 26(1):2–9, Jan 1979.
[145] A. Meixner, M.E. Bauer, and D.J. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. *Micro, IEEE*, 28(1):52–59, Jan 2008.
[146] A. Meixner and D.J. Sorin. Detouring: Translating software to circumvent hard faults in simple cores.

In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 80–89, June 2008.

[147] Maged M. Michael, Martin T. Vechev, and Vijay A. Saraswat. Idempotent work stealing. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '09, pages 45–54, New York, NY, USA, 2009. ACM.

[148] E.R. Minami, S.B. Kuusinen, E. Rosenbaum, P.-K. Ko, and Chenming Hu. Circuit-level simulation of tddb failure in digital cmos circuits. *Semiconductor Manufacturing, IEEE Transactions on*, 8(3):370–374, Aug 1995.

[149] S Mitra, K. Brelsford, Young Moon Kim, H.-H.K. Lee, and Y Li. Robust system design to overcome cmos reliability challenges. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(1):30–41, March 2011.

[150] Eugen I. Muehldorf. Fault clustering: modeling and observation on experimental lsi chips. *Solid-State Circuits, IEEE Journal of*, 10(4):237–244, Aug 1975.

[151] Shubu Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[152] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 29–40, Dec 2003.

[153] A.A. Nair, S. Eyerman, L. Eeckhout, and L.K. John. A first-order mechanistic model for architectural vulnerability factor. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 273–284, June 2012.

[154] Sani R. Nassif, Nikil Mehta, and Yu Cao. A resilience roadmap: (invited paper). In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1011–1016, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[155] Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. Power management architecture of the 2nd generation intel® core microarchitecture, formerly codenamed sandy bridge. 2011.

[156] Panagiota Nikolaou, Yiannakis Sazeides, Lorena Ndreu, Emre Ozer, and Sachin Idgunji. Memory array protection: Check on read or check on write? In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 214–219, March 2013.

[157] Dimitris Nikolos, Antonis M. Paschalis, and George Philokyprou. Efficient design of totally self-checking checkers for all low-cost arithmetic codes. *IEEE Trans. Comput.*, 37(7):807–814, July 1988.

[158] R.A. Oldfield, S. Arunagiri, P.J. Teller, S. Seelam, M.R. Varela, R. Riesen, and P.C. Roth. Modeling the impact of checkpoints on next-generation systems. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*, pages 30–46, Sept 2007.

[159] Ritesh Parikh and Valeria Bertacco. Formally enhanced runtime verification to ensure noc functional correctness. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 410–419, New York, NY, USA, 2011. ACM.

[160] Ritesh Parikh and Valeria Bertacco. Linkmiser: Resource conscious routing and reconfiguration in faulty on-chip networks. In *International Workshop on Logic and Synthesis (IWLS)*, June 2012.

[161] Ritesh Parikh and Valeria Bertacco. udirec: Unified diagnosis and reconfiguration for frugal bypass of noc faults. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 148–159, New York, NY, USA, 2013. ACM.

[162] Sangyoung Park, Jian-Jia Chen, Donghwa Shin, Younghyun Kim, Chia-Lin Yang, and Naehyuck Chang. Dynamic thermal management for networked embedded systems under harsh ambient temperature variation. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 289–294. IEEE, 2010.

[163] J.H. Patel and L.Y. Fung. Concurrent error detection in alu's by recomputing with shifted operands. *Computers, IEEE Transactions on*, C-31(7):589–595, July 1982.

[164] Indrani Paul, Srilatha Manne, Manish Arora, W. Lloyd Bircher, and Sudhakar Yalamanchili. Cooperative boosting: Needy versus greedy power management. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 285–296, New York, NY, USA, 2013. ACM.

[165] A. Pellegrini and V. Bertacco. Cardio: Cmp adaptation for reliability through dynamic introspective operation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 33(2):265–278, Feb 2014.

[166] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 13–23, New York, NY, USA, 2013. ACM.

[167] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems, 2001.

[168] A.F. Pour and M.D. Hill. Performance implications of tolerating cache faults. *Computers, IEEE Transactions on*, 42(3):257–267, Mar 1993.
[169] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and TN Vijaykumar. Gated-v dd: a circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 international symposium on Low power electronics and design*, pages 90–95. ACM, 2000.
[170] Michael D. Powell, Arijit Biswas, Shantanu Gupta, and Shubhendu S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 93–104, 2009.
[171] Andreas Prodromou, Andreas Panteli, Chrysostomos Nicopoulos, and Yiannakis Sazeides. Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 60–71, Washington, DC, USA, 2012. IEEE Computer Society.
[172] V. Puente, J.A. Gregorio, F. Vallejo, and R. Beivide. Immunet: Dependable routing for interconnection networks with arbitrary topology. *Computers, IEEE Transactions on*, 57(12):1676–1689, Dec 2008.
[173] Zhenyu Qi and Mircea R. Stan. Nbti resilient circuits using adaptive body biasing. In *Proceedings of the 18th ACM Great Lakes Symposium on VLSI*, GLSVLSI '08, pages 285–290, 2008.
[174] P. Racunas, K. Constantinides, S. Manne, and S.S. Mukherjee. Perturbation-based fault screening. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 169–180, Feb 2007.
[175] A.-M. Rahmani, P. Liljeberg, K. Latif, J. Plosila, K.R. Vaddina, and H. Tenhunen. Congestion aware, fault tolerant, and thermally efficient inter-layer communication scheme for hybrid noc-bus 3d architectures. In *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, pages 65–72, May 2011.
[176] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D Smith, Gu-Yeon Wei, and David Brooks. Voltage noise in production processors. *IEEE micro*, 31(1):20–28, 2011.
[177] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):300–304, 1960.
[178] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D.I. August. Swift: software implemented fault tolerance. In *Code Generation and Optimization, 2005. CGO 2005. International Symposium on*, pages 243–254, March 2005.
[179] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE micro*, 30(4):65–79, 2010.
[180] D. Rodopoulos, S. B. Mahato, V.V. de Almeida Camargo, B. Kaczer, F. Catthoor, S. Cosemans, G. Groeseneken, A. Papanikolaou, and D. Soudris. Time and workload dependent device variability in circuit simulations. In *IC Design Technology (ICICDT), 2011 IEEE International Conference on*, pages 1–4, May 2011.
[181] Bogdan F. Romanescu and Daniel J. Sorin. Core cannibalization architecture: Improving lifetime chip performance for multicore processors in the presence of hard faults. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 43–51, 2008.
[182] E. Rotenberg. Ar-smt: a microarchitectural approach to fault tolerance in microprocessors. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 84–91, June 1999.
[183] Mohamed M. Sabry, David Atienza, and Francky Catthoor. A hybrid hw-sw approach for intermittent error mitigation in streaming-based embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 1110–1113, San Jose, CA, USA, 2012. EDA Consortium.
[184] D. Sanchez, Y. Sazeides, J.L. Aragon, and J.M. Garcia. An analytical model for the calculation of the expected miss ratio in faulty caches. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 252–257, July 2011.
[185] Smruti Sarangi, Brian Greskamp, Abhishek Tiwari, and Josep Torrellas. EVAL: Utilizing processors with variation-induced timing errors. pages 423–434, November 2008.
[186] T. Sato and Y. Kunitake. A simple flip-flop circuit for typical-case designs for dfm. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 539–544, March 2007.
[187] Yiannakis Sazeides, Emre Özer, Danny Kershaw, Panagiota Nikolaou, Marios Kleanthous, and Jaume Abella. Implicit-storing and redundant-encoding-of-attribute information in error-correction-codes. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 160–171, New York, NY, USA, 2013. ACM.
[188] Dieter K. Schroder. Negative bias temperature instability: What do we understand? *Microelectronics Reliability*, 47(6):841 – 852, 2007.
[189] D.K. Schroder and J.A. Babcock. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of Applied Physics*, 94(1):1–18, Jul 2003.

[190] B. Schroeder and G.A. Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7(4):337–350, Oct 2010.
[191] E. Schuchman and T. N. Vijaykumar. Blackjack: Hard error detection with redundant threads on smt. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 327–337, June 2007.
[192] Saeed Shamshiri, Amirali Ghofrani, and Kwang-Ting (Tim) Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *International Test Conference*. IEEE, IEEE, 09/2011 2011.
[193] Akbar Sharifi, Emre Kultursay, Mahmut Kandemir, and Chita R. Das. Addressing end-to-end memory access latency in noc-based multicores. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 294–304, Washington, DC, USA, 2012. IEEE Computer Society.
[194] Shervin Sharifi, Ayse Kivilcim Coskun, and Tajana Simunic Rosing. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 873–878. IEEE Press, 2010.
[195] Jeonghee Shin, Victor Zyuban, Pradip Bose, and Timothy M. Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 353–362, June 2008.
[196] P. Shivakumar, S.W. Keckler, C.R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Computer Design, 2003. Proceedings. 21st International Conference on*, pages 481–488, Oct 2003.
[197] T. Siddiqua and S. Gurumurthi. Enhancing nbti recovery in sram arrays through recovery boosting. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(4):616–629, April 2012.
[198] Taniya Siddiqua and Sudhanva Gurumurthi. Nbti-aware dynamic instruction scheduling. In *Proceedings of the 5th Workshop on Silicon Errors in Logic-System Effects*. Citeseer, 2009.
[199] Taniya Siddiqua and Sudhanva Gurumurthi. A multi-level approach to reduce the impact of nbti on processor functional units. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, pages 67–72, 2010.
[200] Taniya Siddiqua, Athanasios E Papathanasiou, Arijit Biswas, and Sudhanva Gurumurthi. Analysis and modeling of memory errors from large-scale field data collection, 2013.
[201] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters, 2009.
[202] P Singh, E. Karl, D Sylvester, and D. Blaauw. Dynamic nbti management using a 45nm multi-degradation sensor. In *Custom Integrated Circuits Conference (CICC), 2010 IEEE*, pages 1–4, Sept 2010.
[203] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 2–13, June 2003.
[204] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, 2004.
[205] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. Dtm: Degraded test mode for fault-aware probabilistic timing analysis. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 237–248, July 2013.
[206] Jared C. Smolens, Brian T. Gold, James C. Hoe, Babak Falsafi, and Ken Mai. Detecting emerging wearout faults. In *In Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects*, 2007.
[207] Daniel J Sorin. Fault tolerant computer architecture. *Synthesis Lectures on Computer Architecture*, 4(1):1–104, 2009.
[208] Daniel J Sorin. Fault tolerant computer architecture. *Synthesis Lectures on Computer Architecture*, 4(1):29–30, 2009.
[209] Vilas Sridharan and Dean Liberty. A study of dram failures in the field. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 76:1–76:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
[210] Vilas Sridharan, Jon Stearley, Nathan DeBardeleben, Sean Blanchard, and Sudhanva Gurumurthi. Feng shui of supercomputer memory: Positional effects in dram and sram faults. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 22:1–22:11, New York, NY, USA, 2013. ACM.
[211] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. The case for lifetime reliability-aware microprocessors. In *Proceedings of the 31st International Symposium on Computer Architecture*, pages 276–287, June 2004.
[212] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. Lifetime reliability: Toward an architectural solution. *IEEE Micro*, 25(3):70–80, 2005.

[213] A. Strano, D. Bertozzi, F. Trivino, J.L. Sanchez, F.J. Alfaro, and J. Flich. Osr-lite: Fast and deadlock-free noc reconfiguration framework. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 86–95, July 2012.

[214] L. Subramanian, V. Seshadri, Yoongu Kim, B. Jaiyen, and O. Mutlu. Mise: Providing performance predictability and improving fairness in shared main memory systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 639–650, Feb 2013.

[215] Caixia Sun, Yongwen Wang, and Jinbo Xu. Achieving predictable performance in smt processors by instruction fetch policy. In Weixia Xu, Liquan Xiao, Chengyi Zhang, Jinwen Li, and Liyan Yu, editors, *Computer Engineering and Technology*, volume 396 of *Communications in Computer and Information Science*, pages 186–197. Springer Berlin Heidelberg, 2013.

[216] D Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design Test of Computers, IEEE*, 23(6):484–490, June 2006.

[217] Dong Tang, P. Carruthers, Z. Totari, and M.W. Shapiro. Assessment of the effect of memory page retirement on system ras against hardware faults. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 365–370, 2006.

[218] Radu Teodorescu and Josep Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. *ACM SIGARCH Computer Architecture News*, 2008.

[219] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. Recycle: : pipeline adaptation to tolerate process variation. pages 323–334, June 2007.

[220] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. Recycle:: Pipeline adaptation to tolerate process variation. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 323–334, New York, NY, USA, 2007. ACM.

[221] Abhishek Tiwari and Josep Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 129–140, Washington, DC, USA, 2008. IEEE Computer Society.

[222] Toshiba. Advanced configuration and power interface specification. *Techical White Paper*.

[223] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, volume 23, pages 392–403. ACM, 1995.

[224] Aniruddha N. Udipi, Naveen Muralimanohar, Rajeev Balsubramonian, Al Davis, and Norman P. Jouppi. Lot-ecc: Localized and tiered reliability mechanisms for commodity memory systems. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 285–296, Washington, DC, USA, 2012. IEEE Computer Society.

[225] O.S. Unsal, J.W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. Impact of parameter variations on circuits and microarchitecture. *Micro, IEEE*, 26(6):30–39, Nov 2006.

[226] K. Van Craeynest and L. Eeckhout. The multi-program performance model: Debunking current practice in multi-core simulation. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 26–37, Nov 2011.

[227] Kenzo Van Craeynest and Lieven Eeckhout. Understanding fundamental design choices in single-isa heterogeneous multicore architectures. *ACM Trans. Archit. Code Optim.*, 9(4):32:1–32:23, January 2013.

[228] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 210–221, New York, NY, USA, 2013. ACM.

[229] Priyamvada Vijayakumar, Pritish Narayanan, Israel Koren, C. Mani Krishna, and Csaba Andras Moritz. Impact of nanomanufacturing flow on systematic yield losses in nanoscale fabrics. In *Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures*, NANOARCH '11, pages 181–188, 2011.

[230] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos. A fine-grained link-level fault-tolerant mechanism for networks-on-chip. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 447–454, Oct 2010.

[231] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[232] Jiajing Wang and B.H. Calhoun. Canary replica feedback for near-drv standby vdd scaling in a 90nm sram. In *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE*, pages 29–32, Sept 2007.

[233] Jiajing Wang, A. Hoefler, and B.H. Calhoun. An enhanced canary-based system with bist for sram standby power reduction. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(5):909–914, May 2011.

[234] N.J. Wang and S.J. Patel. Restore: symptom based soft error detection in microprocessors. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 30–39, June 2005.

[235] Ruisheng Wang, Lizhong Chen, and Timothy Mark Pinkston. An analytical performance model for partitioning off-chip memory bandwidth. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 165–176, Washington, DC, USA, 2013. IEEE Computer Society.

[236] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 583–594, New York, NY, USA, 2013. ACM.

[237] C. Weaver, J. Emer, S.S. Mukherjee, and S.K. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 264–275, June 2004.

[238] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 2. USENIX Association, 1994.

[239] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. Trading off cache capacity for reliability to enable low voltage operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 203–214, Washington, DC, USA, 2008. IEEE Computer Society.

[240] Gulay Yalcin, Osman Unsal, Ibrahim Hur, Adrian Cristal, Mateo Valero, et al. Faultm: Fault-tolerance using hardware transactional memory. In *Pespma 2010-Workshop on Parallel Execution of Sequential Programs on Multi-core Architecture*, 2010.

[241] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 607–618, 2013.

[242] Kazuaki Yazawa and Avram Bar-Cohen. Energy efficient cooling of notebook computers. In *Thermal and Thermomechanical Phenomena in Electronic Systems, 2002. ITHERM 2002. The Eighth Intersociety Conference on*, pages 785–791. IEEE, 2002.

[243] Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, volume 1, pages 293–307 vol.1, Feb 1996.

[244] Mahmut Yilmaz, Albert Meixner, Sule Ozev, and Daniel J. Sorin. Lazy error detection for microprocessor functional units. In *Proceedings of the 22Nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, DFT '07, pages 361–369, 2007.

[245] Doe Hyun Yoon and Mattan Erez. Virtualized and flexible ecc for main memory. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 397–408, New York, NY, USA, 2010. ACM.

[246] John W. Young. A first order approximation to the optimum checkpoint interval. *Commun. ACM*, 17(9):530–531, September 1974.

[247] Qiaoyan Yu, Meilin Zhang, and Paul Ampadu. Exploiting inherent information redundancy to manage transient errors in noc routing arbitration. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, pages 105–112, New York, NY, USA, 2011. ACM.

[248] F. Zanini, D. Atienza, and G. De Micheli. A control theory approach for thermal balancing of MPSoC. *2009 Asia and South Pacific Design Automation Conference*, 2009.

[249] Lihang Zhao, Woojin Choi, Lizhong Chen, and Jeffrey Draper. In-network traffic regulation for transactional memory. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 520–531, Washington, DC, USA, 2013. IEEE Computer Society.

[250] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling. In James C. Hoe and Vikram S. Adve, editors, *ASPLOS*, pages 129–142. ACM, 2010.

[251] J.F. Ziegler. Terrestrial cosmic rays. *IBM Journal of Research and Development*, 40(1):19–39, Jan 1996.

[252] J.F. Ziegler, H. W. Curtis, H.P. Muhlfeld, C.J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J.L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T.J. O'Gorman, B. Messina, T.D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus. Ibm experiments in soft fails in computer electronics. *IBM Journal of Research and Development*, 40(1):3–18, Jan 1996.

[253] D. Zoni, S. Corbetta, and W. Fornaciari. Thermal/performance trade-off in network-on-chip architectures. In *System on Chip (SoC), 2012 International Symposium on*, pages 1–8, Oct 2012.

[254] Davide Zoni and William Fornaciari. Sensor-wise methodology to face nbti stress of noc buffers. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1038–1043, March 2013.