Processor Design for Soft Errors: Challenges and State of the Art

TUO LI, University of New South Wales
JUDE ANGELO AMBROSE, Canon Information Systems Research Australia
ROSHAN RAGEL, University of Peradeniya
SRI PARAMESWARAN, University of New South Wales

Today, soft errors are one of the major design technology challenges at and beyond the 22nm technology nodes. This article introduces the soft error problem from the perspective of processor design. This article also provides a survey of the existing soft error mitigation methods across different levels of design abstraction involved in processor design, including the device level, the circuit level, the architectural level, and the program level.

Categories and Subject Descriptors: C.1 [**Processor Architectures**]: General; C.4 [**Performance of Systems**]: Fault Tolerance

General Terms: Design, Reliability

Additional Key Words and Phrases: Processor, recovery, soft error

ACM Reference Format:
Tuo Li, Jude Angelo Ambrose, Roshan Ragel, and Sri Parameswaran. 2016. Processor design for soft errors: Challenges and state of the art. ACM Comput. Surv. 49, 3, Article 57 (November 2016), 44 pages.
DOI: http://dx.doi.org/10.1145/2996357

1. INTRODUCTION

Soft errors are a significant issue for semiconductor-related industry sectors, particularly in the fabrication and design (i.e., CAD) of memory systems and microprocessors [Mitra et al. 2005a]. According to the 2011 version of the ITRS report [Kahng 2013],[1] one of the major design technology challenges at and beyond the 22nm technology node is the issue of soft error. Soft errors are induced by energetic particles (neutron and alpha particles) striking the semiconductor substrate of transistors [Baumann 2005]. Soft errors are essentially transient failures that do not permanently damage the processor and do not recur.

When a strike hits the substrate, the strike releases electron and hole pairs that are absorbed by the source and drain of the affected transistor to alter the state of the device. The output of the gate can thus be flipped to an incorrect value. After error propagation, the incorrect flip at the gate output can be masked or be visible at the

[1]http://www.itrs2.net.

Authors' addresses: T. Li and S. Parameswaran, School of Computer Science and Engineering, University of New South Wales, Sydney NSW 2052, Australia; emails: {tuol, sridevan}@cse.unsw.edu.au; J. A. Ambrose, Canon Information Systems Research Australia, 5 Talavera Rd, Macquarie Park NSW 2113, Australia; email: angelo.ambrose@cisra.canon.com.au; R. Ragel, Department of Computer Engineering, University of Peradeniya, Peradeniya 20400, Sri Lanka; email: roshanr@pdn.ac.lk.

DOI: http://dx.doi.org/10.1145/2996357

system level, which then will affect the system. For example, the output of the program executing on the underlying system can be corrupted or the system can crash.

Soft errors have been observed since the 1970s [Binder et al. 1975]. Wallmark and Marcus [1962] theorized that cosmic rays would eventually limit silicon device dimensions to 10μm. In 1975, Binder et al. [1975] reported the cosmic-ray-induced upsets in space electronics, and in 1976, ground-level cosmic-neutron-induced upsets were recorded in the Cray-1 computers at Los Alamos [Normand et al. 2010]. In 1979, May and Woods [1979] reported alpha-particle-related upsets in dynamic random memories (DRAMs).

Historically, soft errors have been primarily concerned with the design of high-availability systems or systems that are deployed in environments that are hostile to electronics, such as outer space. As devices scale, the soft error rate for each individual device due to radiation is projected to remain approximately constant or expected to decrease (depending on the specific device). However, the exponential increase in the integration of the chip (i.e., the computing system) leads to correspondingly dramatic decreases in reliability.[2]

Soft errors are induced in the physical layer, where the channel in a semiconductor is affected by neutron flux or alpha particles. However, physical solutions have been suggested as inadvisable [Mukherjee et al. 2005] due to excessive cost. Therefore, a large number of soft error mitigation techniques [Iyer et al. 2005] have been proposed across levels of design abstractions, from the transistor level, which is the most primitive and physical form of computation, to the application level, which is highly abstract and is an algorithmic form of computation.[3]

Given that seminal literature by Mukherjee [2008] in 2008 has already elaborated soft error mechanism, testing, and modeling in depth, we review the nature and system effects of soft errors in brief. Then, we devote the major portion of this article to examining the existing soft error mitigation techniques. In order to show a thorough and chronological view of the development of the mitigation techniques, this survey also covers the techniques discussed in Mukherjee [2008].

The rest of the article is structured as follows. Section 2 provides a brief review of the soft error problem, including the subsections reviewing the mechanism, propagation effect, and trend. Section 3 discusses the existing techniques for soft error mitigation. Section 4 concludes the article.

2. SOFT ERROR CHALLENGE: A BRIEF REVIEW

Soft errors (shown in Figure 1) are defined as a category of errors, occurring in the form of a flipped or reversed data state, in semiconductor technologies as memory cells, registers, latches, and flip-flops, and are typically caused by radiation events [Baumann 2005]. The error is called "soft" because the underlying circuit/device is not permanently damaged and new data can still be correctly written and stored in the circuit/device. The radiation events are induced by three primary sources [Baumann 2005], commonly existing in a terrestrial environment. These are as follows: **alpha particles** emitted from packaging materials were observed to be dominantly contributive to soft errors in DRAM devices in the 1970s; **high-energy cosmic rays**, originating from galactic space, were identified as the second important source of soft errors, by producing neutrons at terrestrial altitudes; and **low-energy cosmic rays**, which result in thermal neutron and boron reactions, are the third significant source in most of the cases. Soft error is also referred to as a single event upset (SEU). Depending on the amount of charge disturbance brought in by the radiation event, one SEU can create a single-bit

[2]The soft error trend against technology scaling is further discussed in Section 2.2.
[3]The mitigation techniques at all of these levels will be elaborated in Section 3.

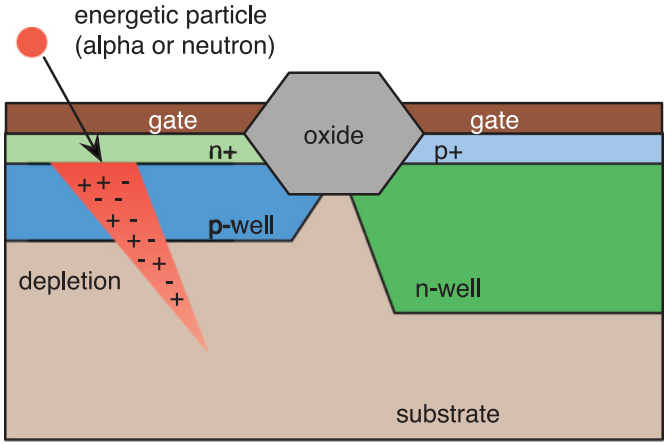

Fig. 1. An energetic particle strike (radiation event) on a semiconductor device, the metal-oxide-semiconductor field-effect transistor (MOSFET), which is the fundamental building block of electronic computing systems.

upset (SBU) or multiple-bit upset (MBU) [Johansson et al. 1999]. In the terrestrial environment, soft errors are known to increase substantially as the altitude increases [Ziegler 1996].

Hereafter, the metric for SEU occurrences in a system is termed as soft error rate (SER). SER is quantified in real-world applications as failure in time (FIT) or mean time between failure (MTBF). The value of MTBF can be obtained by adding the mean time to repair a failed system on top of MTTF.[4] Mean time to failure (MTTF) represents the expected time that a system is supposed to function correctly before a failure. One FIT is defined as one failure in 10^9 device hours. The relationship of FIT to MTTF can be formulated as MTTF $= 1 \times 10^9$/FIT. Since it is additive, the FIT metric is used in the semiconductor industry for expressing reliability. The reliability of a system can be calculated from the reliability of the components [Wang 2007]. The elaboration of field test methodology for SER can be found in Mukherjee [2008] and the JEDEC standard.[5]

2.1. Mechanism

Soft error is caused by radiation events on semiconductor devices (shown in Figure 1). These events are known to produce, either directly or as secondary reaction products, energetic ions that are accountable for causing a transient fault in the underlying devices, by inducing an excess charge. The transient fault is often referred to as a single-event transient (SET) [Hass and Ambles 1999]. This stage is the original mechanism and the beginning of the propagation of the soft error.

SET comes from the excess charge injected into the semiconductor device by a radiation event. The charge generation and collection [Messenger 1982] can be divided into three main phases [Baumann 2005], during a radiation event:

—First, a track of electron-hole pairs and a high carrier concentration is formed in the ion's path.
—Second, as the ion track traverses to the depletion region, the electric field collects the carriers and hence generates a substantial current/voltage transient at the transistor.

[4]In some documents, MTTF is used as MTBF.
[5]http://www.jedec.org/sites/default/files/docs/jesd89a.pdf.

—Third, the diffusion of electrons toward the depletion region begins, and this behavior collects additional charge until all excess carriers have been captured, recombined, or diffused away.

Upon a radiation event, considering the SRAM cell, the collected excess charge, Q_{coll}, can be modeled (e.g., in Ibe et al. [2010]) as

$$Q_{coll} = Q_{all} \exp\left(-\frac{x_c}{L_{max}}\right), \tag{1}$$

where Q_{all} is the total deposited charge in the funneling path with length x_c, and L_{max} denotes the effective maximum funneling length considering the recombination effect of electrons and holes along the funneling path. Q_{coll} normally ranges from 1 to several 100 fC. The precise value of Q_{coll} is dependent on the type of ion, the trajectory, and the energy over the path through or near the junction (most probably at the reverse-biased junction). Therefore, there are a number of factors that are related to the amount of Q_{coll}, such as the size of the device, biasing of the circuit nodes, substrate structure, device doping, the type of ion, the energy of ion, the trajectory, the initial location of the event within the device, and the state of the device. Correspondingly, the change in current pulse can be observed throughout the three phases. The current pulse can be expressed as

$$I(t) = \frac{Q_{coll}}{\tau_\alpha - \tau_\beta}\left(e^{-\frac{t}{\tau_\alpha}} - e^{-\frac{t}{\tau_\beta}}\right), \tag{2}$$

where τ_α stands for the collection time constant and τ_β is the ion-track establishment time constant. τ_α and τ_β are process dependent and have typical values of 164ps and 50ps, respectively.

Besides, SET occurrence is also determined by the sensitivity of the underlying device. This sensitivity is defined as Q_{crit}, meaning the magnitude of the critical charge required to stimulate a change in the data state. The value of Q_{crit} can be expressed as

$$Q_{crit} = \int_0^{T_F} I_D(t)dt, \tag{3}$$

where I_D is the time-dependent drain transient current and T_F represents the flipping time.[6] Q_{crit} is mainly affected by the node capacitance, operating voltage, and strength of feedback transistors; therefore, Q_{crit} is variable depending on the radiation pulse characteristics and the dynamic response of the circuit itself [Dodd and Sexton 1995], which is influenced by the semiconductor technology. The relationship of SET to Q_{coll} and Q_{crit} can be concluded as follows:

—In the case of simple circuits, such as DRAM cells in storage mode, SET only occurs if the relationship $Q_{coll} > Q_{crit}$ holds, at a radiation event. Otherwise, if $Q_{coll} \leq Q_{crit}$, the SET is masked off in the device.
—In the case of complex circuits, such as SRAM cells [Carter and Wilkins 1987], where active feedback is involved, an additional term, which represents the circuit reacting speed, should be considered. This additional term tends to increase Q_{crit}.

In general, there are two types of methodologies that relate Q_{crit} to FIT (SER) [Mukherjee 2008]. The first type semiempirically maps Q_{crit} to FIT, for example, the Hazucha and Svensson Model [Hazucha and Svensson 2000], Burst Generation Rate

[6]Flipping time is defined as the time when the drain voltage is the same as the gate voltage of the struck transistor after the radiation event occurs [Palau et al. 2001].

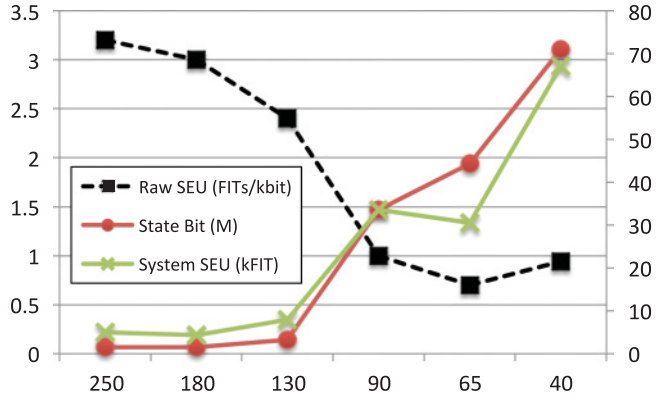

Fig. 2. Soft error trend (dashed lines with left y-axis; others with right y-axis) as a function of technology node in nm (x-axis). Replotted using data from Dixit and Wood [2011].

Method [Ziegler and Lanford 1979], and Neutron Cross-Section Method [Taber and Normand 1993]. The other type uses simulation models to map Q_{crit} to FIT, for example, SEMM [Murley and Srinivasan 1996]. A more detailed discussion on SER modeling can be found in Mukherjee [2008].

With technology scaling, MBU, also referred to as single-event multiple upset (SEMU), is increasingly more prominent. MBU is mainly caused by high-energy neutrons. The probability of MBU is related to the collected charge (Q_{coll}) and the distance between cells. The major impact of MBU is that the memory protection techniques such as parity and error-correcting codes can be corrupted. To mitigate MBU in memory, interleaving memory array columns is typically adopted [Maiz et al. 2003]. A detailed discussion about MBU can be found in Seifert et al. [2006].

2.2. Trend with Technology Scaling

2.2.1. Standard Sequential Cell with CMOS Technology. The SEU rate in standard sequential cells is decreasing, as the technology node scales down. This trend can be explained in two contrasting viewpoints regarding two respective factors:

—The nominal supply voltage (V_{dd}) has been gradually decreasing as the technology node scales down. With V_{dd} scaling, the critical charge (Q_{crit}) also decreases.
—The size of the cells has been decreasing, as a direct ramification of technology scaling. Hence, the size of the sensitive area is reduced, which leads to a reduction in the SEU rate of the cells.

Overall, the SEU rate is expected to be determined by V_{dd} reduction, and thus to rise eventually. Notwithstanding, for flip-flops, this kind of increase in the SEU rate is not seen for technologies down to 28nm [Dixit and Wood 2011]. The main reason is that flip-flops are larger than SRAM cells, and hence the sensitive area reduction possesses more weight over V_{dd} scaling. The other possible reason is the manufacturing method and design style.

2.2.2. Nominal Processor with CMOS Technology. As shown in Figure 2, the SEU rate in nominal processors (“System SEU”) is increasing as the technology node scales down. Since a processor is a system composed of a large number of logic circuits, the overall SEU rate in a processor is determined by two factors: SEU rate per state bit (“Raw SEU” in Figure 2) and the number of state bits. In an abstract form, the relationship

can be expressed as

$$\text{SEU\_per\_Processor} \propto \text{SEU\_per\_Bit} \times \text{Num\_Bit}, \tag{4}$$

which can be explained as follows:

—The SEU rate per state bit is decreasing, determined by V_{dd} reduction, as the technology node scales down in typical microprocessors. This trend largely follows the trend of the two types of the sequential cells, because the actual implementation of a state bit is a sequential cell.
—The number of state bits in a typical microprocessor is increasing. The numbers indicate the evolvement of microprocessor design over the years. The technology scaling from 250nm to 180nm only relied on optical scaling; 130nm started to use additional silicon material on the scaled chip; 90nm invoked the usage of large caches; 65nm node introduced multicore architecture; and 40nm node continued the usage of multicore architecture with larger on-chip memory.

Consequently, these two factors jointly contribute to the increasing trend of soft error (represented by SEU rate) in a typical microprocessor during the last technology generations. The increase of the number of SRAM cells due to the preference of larger caches and memories dominates the trend in soft error per processor, even though the SEU rate per bit decreases.

2.2.3. Emerging Technology. In addition to planar CMOS devices (e.g., MOSFET), a few new devices have been emerging, including FinFET [Ho et al. 2001; King 2005] and CNTFET [Bachtold et al. 2001; Wong et al. 2003]. These two devices are considered to be promising replacements of CMOS FET for process technologies below 28nm.

FinFET,[7] namely, multiple-gate FETs resilience to soft error, has been studied extensively [Endo 2008; Fang and Oates 2011; Wang et al. 2006]. The SER of FinFET is much lower than MOSFET. Specifically, Fang and Oates [2011] used TCAD[8] simulations and obtained the comparison of the SER for a 6-T SRAM cell built with FinFET and a Planar FET. Because only the fin body is connected to the substrate, FinFET collects less excessive charge (Q_{coll}) in a radiation event. The results showed a decrease in SER by a factor of 17 in FinFET SRAM. At sub-28nm technology nodes (FinFET), MBU occurrence is much less than SBU (8.6% of SBU at 22nm and below 1% of SBU at 14nm) [Seifert et al. 2015].

Unlike FinFET, little is known about the soft error in **CNTFET**.[9] Poolakkaparambil et al. [2012] investigated the viability of the standard concurrent error detection (CED) scheme,[10] which is a countermeasure to transient fault, with CNTFET. This work showed that CNTFET CED implementation has less overhead in terms of area, power, and delay, in comparison to its MOSFET counterpart. Moreover, a few additional reliability problems, such as avalanche, joule breakdown, and hysteresis, is more noticeable in CNTFET [Pop et al. 2009] than in MOSFETs.

[7]The term is coined by Huang [1999]. It refers to a nonplanar, double-gate transistor built on an SOI substrate.
[8]TCAD is the abbreviation of technology computer-aided design. It refers to the use of computer simulations to develop and optimize semiconductor processing technologies and devices.
[9]CNTFET, also known as CNFET [Patil et al. 2009], is the abbreviation for carbon nanotube FET, which was introduced in Trans et al. [1998]. It refers to the use of carbon nanotube(s) as the channel material instead of bulk silicon as in a FET.
[10]CED techniques (based on hardware duplication, parity codes, etc.) are widely used to enhance system dependability [Mitra and McCluskey 2000; Pradhan 1996; Sellers et al. 1968].

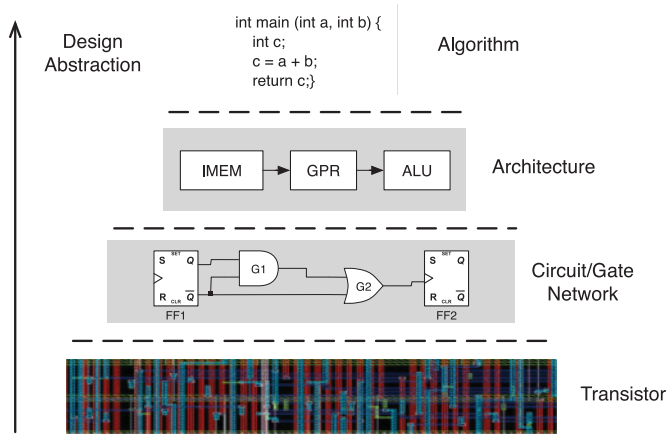

Fig. 3. Soft error propagation through levels of abstractions in a processor system.

2.3. Propagation and System Effect

Consider a large-scale, that is, very-large-scale-integration (VLSI) digital computing system, including embedded processors, which is composed of a number of functional units. Each unit performs a unique function and consists of various logic gates from the bottom level of the design abstraction. As shown in Figure 3, the soft error mechanism involved is much more complex. The effect of the soft error can traverse (or be masked) across a number of layers or levels of design abstraction in a processor system.

2.3.1. Logic and Circuit Level. At circuit level, there are two types of circuits: the combinational and sequential circuits. Combinational circuits have very low soft error sensitivity. **Combinational circuits** (e.g., AND, OR, and other logic gates) themselves are not regarded to be prone to soft errors, since the reversed data state cannot be stored when there is no feedback loop in the underlying circuit. However, the SET in combinational circuits may be latched by the connected sequential circuits [Shivakumar et al. 2002; Gill et al. 2009]. **Sequential circuits** (e.g., SRAM cells, latches, flip-flops, and other memory cells), on the other hand, are highly susceptible to soft errors, since the SET can be held by the feedback loop if the corrupted signal meets the timing requirement [Seifert and Tam 2004].

Regarding the combinational circuit, there are three major masking factors that can suppress SET, a transient fault, from propagating to the output of the circuit [Miskov-Zivanov and Marculescu 2006]:

—**Logical masking** occurs when SET arrives at the input of a logic gate, on the condition that the other inputs of the underlying gate have a dominating value. Therefore, the transient fault is gated, not contaminating other circuits. For example, a transient fault in one input of an AND gate can be masked off if one of the other inputs has a logical “0” value [Blome et al. 2005].
—**Electrical masking** (also known as pulse attenuation) is dependent on the relation between the electrical properties of the gates, which are traversed by the fault, and the properties of the fault, including the duration and amplitude. This factor has a similar impact as logical masking, in most cases [Miskov-Zivanov and Marculescu 2010].
—**Latching-window masking** (also known as temporal masking) happens when the transient fault arrives at the input of a sequential circuit, on the condition that the faulty signal cannot meet the fundamental timing requirement of the corresponding

sequential circuit, that is, setup and hold time [Rabaey 1996]. Therefore, the fault is not successfully latched into the circuit, and hence, it cannot propagate further.

Regarding the sequential circuit, the **timing vulnerability factor (TVF)** [Seifert and Tam 2004] is known to determine the vulnerability of the circuit. Different sequential circuits tend to have different TVFs. An SRAM cell usually has a TVF of 100%, meaning an SRAM cell is always vulnerable to SEU. On the other hand, a flip-flop or latch may have a smaller TVF. The reason for this difference is that: (1) for an SRAM cell, any particle strike during a clock cycle is able to change the value residing in the SRAM cell, and (2) for a flip-flop, when the clock phase is low, the flip-flop is driving data, instead of sampling (at high phase), and therefore the upset in the low phase can be masked. As a result, the TVF of a flip-flop is about 50%.

In addition, the circuit-implementing clock network is also vulnerable to soft errors. There are two major effects of soft errors in a clock network [Seifert et al. 2005]:

—Similar to noise and process variation, soft errors can change the time of a clock edge. Such change is called **clock jitter**. A jitter is usually created upon a particle strike at a clock node, near the time when the clock is asserted.
—In some cases, the excessive charge from a particle strike can be so large that an additional clock pulse is inserted between original clock edges. Such an additional clock edge is able to activate unpredicted operations in the clocked circuit, which might cause **data races** in the clocked latches.

2.3.2. Architecture Level. At the architecture level, the objects under discussion are such microarchitecture components as the program counter (PC), registers, register-file, arithmetic logic unit (ALU), multiplexer (MUX), cache, instruction memory (IMEM), data memory (DMEM), and so forth [Hennessy and Patterson 2006].

Mukherjee [2008] identified different effects on the processor architecture due to soft errors. These are as follows:

—Corrupted data, which is never noticed by the user, is considered benign.
—Silent data corruption (SDC) is when the data is corrupted but noticed much later, after it propagates to a visible error.
—Detected unrecoverable error (DUE) is detected by the system, which then takes action such as crashing the system without corrupting the user's data.
—Corrected errors, which are the soft errors detected and corrected by the system, are not of concern.

In comparison to SDC, DUE is usually considered less significant in terms of harm and damage to the system and users. The reason is that DUE only makes the system unavailable, while SDC results in unnoticed corrupted data.

In processor architecture, each individual architectural component is composed of a specific set of circuit networks, of which the propagation mechanism has been discussed in Section 2.3.1. The program state, which can be defined as the values in the architectural components such as registers and memories, is crucial since those values are directly manipulated by the program. The masking factor at this level is named **architectural masking**. It is defined as the case where an error, which is resident in a component, is not able to be visible to the program and user. For example, a transient fault in the destination register specifier of an NOP (null operation) instruction cannot corrupt any architectural state [Sorin 2009]. Architectural masking has been well studied recently. One recent seminal work is the **architectural vulnerability factor (AVF)** [Mukherjee et al. 2003b]. Following the study of AVF, the state bits in an architecture, such as a microprocessor, can be basically divided into two categories: architecturally correct execution (ACE) bits and un-ACE bits [Mukherjee et al. 2003b].

ACE bits are contributive to error propagation at the program/algorithm level (user visible), while un-ACE bits are not. In-depth elaboration on architectural masking can be seen in Mukherjee [2008].

2.3.3. Algorithm and Program Level. The top level is the program/algorithm level. The common design representation of the program level is a high-level language program (say C/C++), and the main objects are data types, data structure, variables, functions, expressions, and so forth. Soft errors that propagate to this level are considered severe, since the computation of the program might be corrupted by the errors in its elements. The primary masking factor at the program level is **application masking** [Sorin 2009]. Such masking occurs when the transient fault affects a program object that is always restored by the behavior of the program itself. For example, a soft error that corrupts a variable stored in memory is program visible; nonetheless, if the program always assigns a new value to that variable before using it, the soft error is masked. In a typical OS studied by Gu et al. [2003], with a fault injection campaign focusing on the virtual file system interface, about 30% to 50% of activated faults are masked off[11] as a result of the error propagation and masking factor.

3. SOFT ERROR MITIGATION

Within the scope of the processor, the mitigation for soft error can be explored at each level of the design abstraction. As illustrated in Figure 4, the solution space can be divided into four classes, each of which is associated to a level of design abstraction (the transistor level, circuit level, architecture level, and algorithmic level). Each class of solutions can be further divided into subcategories. The details of these mitigation methods are elaborated in later sections. Notably, a recent study has implemented and compared some soft error countermeasures, with a unified setup (i.e., same metrics, benchmarks, test methodology, and baseline processors), by using a cross-layer framework [Cheng et al. 2016a, 2016b]. The authors stated that, by performing a thorough analysis of an application, circuit-level mitigation techniques (as opposed to techniques at other levels) are the most cost-effective, in terms of energy consumption. However, given the magnitude of the countermeasures surveyed in this manuscript, implementing and comparing all the techniques will be prohibitively expensive. Therefore, in this article, the examination is based on the data reported in the corresponding literature.

3.1. Mitigation at the Transistor Level

Transistor-level solutions mostly rely on process technology to shield radiation events (such as alpha and neutron strike) and to minimize the amount of charge (Q_{coll}) collected into a transistor node upon a radiation event. Table I depicts the major approaches at this level. By careful fabrication of the transistors, the critical charge is increased or the material is changed so that the collected charge is reduced such that $Q_{coll} < Q_{crit}$, which then avoids SET occurrence.

Coating for soft errors (in particular for alpha particles) has been employed in the semiconductor industry.[12] These coating techniques generally adopt a thick polyimide[13] (100μm reported by Itoh [1980]) as an alpha-particle protection layer, due to the thermal and electrical properties of the polymide. It has been shown that coating leads to an effective reduction of alpha-originated SER to around 20% of neutron-originated SER [Kohara et al. 1990]. Despite the effectiveness of the coating-like solutions against

[11]Categorized as "not manifested," other than "fail silent," "crash," and "hang."

[12]Technical details can be obtained in IBM's patents [Cairns and Ziegler 1985, 1989].

[13]For semiconductor devices, there are other applications of polyimide including insulation dielectric layer and stress-relief buffer coating [Tummala et al. 1997].

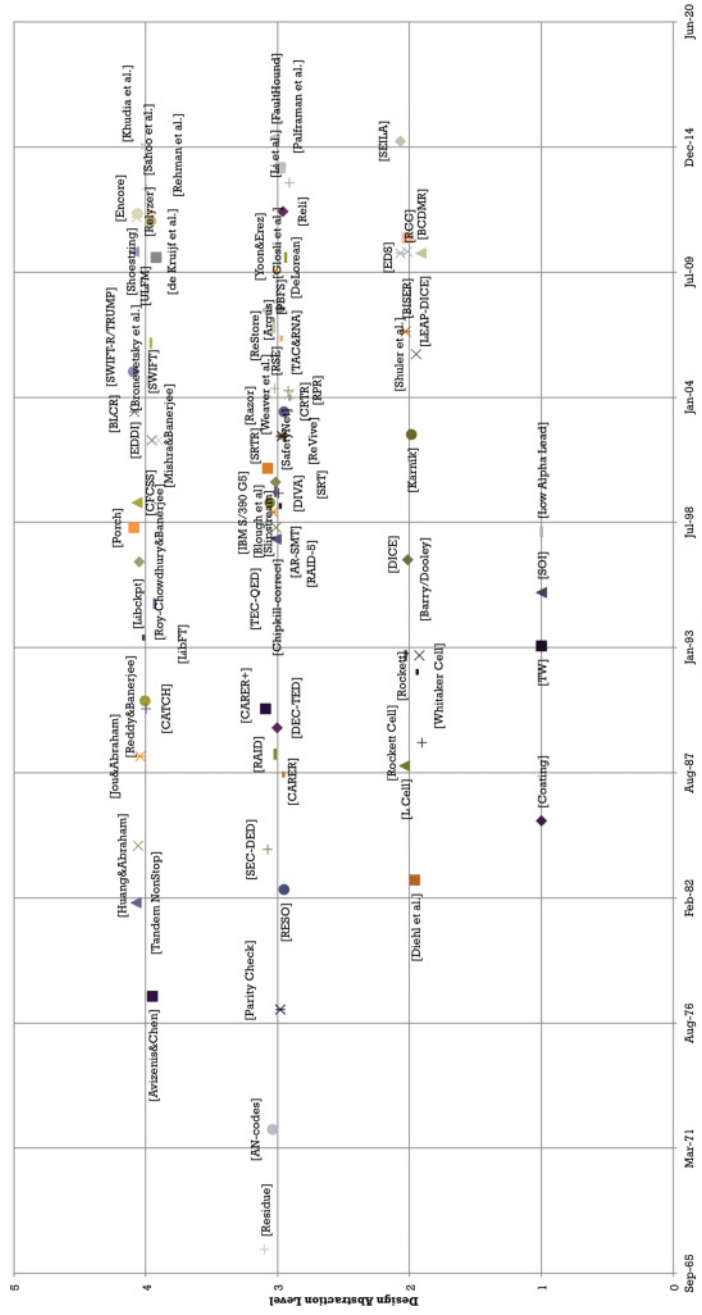

Fig. 4. Spectrum of soft error mitigation discussed in this article. The techniques are categorized by publication date and design abstraction level (1: transistor, 2: circuit, 3: architecture, 4: algorithm). For visibility, at each design abstraction level, the mitigation techniques are shuffled up and down randomly.

Table I. Summary of Transistor-Level Mitigation

Approach	Year	Notes
Coating	1985	Only effective for alpha-originated SER but neutron-originated SER
Triple Well	1993	Unreliable SER reduction as technology scales
Silicon-on-insulator	1995	Difficult volume manufacturing
Low-Alpha Lead Solder	1998	High expense, alpha particle only

alpha-particle-induced soft error, these solutions continue to suffer in the face of neutron strikes. In order to shield the chip from neutron strikes, the protection layer should be approximately 10 feet thick in concrete, which is impractical in most circumstances [Mukherjee et al. 2005]. Thus, coating is not able to solve the soft error problem, since neutron-SER is starting to dominate, and the cost of reducing alpha-SER by the use of coating is still prohibitive.

On the other side, regarding solder-bumped packaging, MCNC[14] researchers reported that **low alpha lead** (or Pb-free) is promising for chip alpha-SER reduction [Roberson 1998], since Pb's radioactivity can generate alpha particles. However, in comparison to a normal PbSn (i.e., lead) solder, a low alpha lead solder is more expensive and less reliable in terms of vibration and thermal environment. The details of low alpha lead solder can be found in the studies of various space agencies, such as NASA[15] in the United States and DSTO (Air Vehicles Division) [Lim 2010] in Australia. In the IC industry, R&D efforts regarding low alpha solder have been seen as well [Lee 2000], including the patent filled by AMD [Master et al. 2007].

Silicon-on-insulator (SOI) process technology[16] has shown promising properties against soft error. These properties are superior to those of bulk CMOS process technology. Due to its lower parasitic capacitance (caused by isolation from silicon), SOI further improves the device in terms of latency and power consumption. SOI transistors are able to reduce the collected charge, that is, Q_{coll}, from the radiation event. The reason for this reduction is that the isolation of the device from bulk silicon can cause excess charge in bulk silicon, induced by the radiation event, to not affect the sensitive region of the device.[17] IBM achieved 5× reduction in SER for SRAM by using partially depleted SOI process technology [Cannon et al. 2004]. In recent years, IBM obtained promising results for reducing SER with SOI process technology [Oldiges et al. 2009]. They further showed methods of hardening latch design based on SOI [Rodbell et al. 2011]. The major drawback of SOI-based solutions is the difficulty in volume manufacturing.

Additionally, research [Burnett et al. 1993] in the creation of new well structures for CMOS process technology has been reported for mitigating soft errors. **Triple-well** (TW) process techniques,[18] such as those of Burnett et al. [1993], have been proposed to decrease SEU sensitivity in the device. Typically, a TW device contains a buried n-well layer that isolates the p-well from the p-substrate. The additional benefits of applying the triple-well process include the switching speed improvement for the digital CMOS device, noise reduction for the analog CMOS device, and density and performance

[14]http://www.rtp.org/mcnc-ncren/.

[15]http://teerm.nasa.gov/LeadFreeSolderTestingForHighReliability_Proj1.htm.

[16]SOI process technology refers to the use of a layered silicon-insulator-silicon substrate in place of conventional silicon substrates [Bruel 1995].

[17]As discussed in Section 2.1, only the excess charge in the thin silicon layer above the oxide insulator layer is accountable for generating Q_{coll} effectively.

[18]The triple-well process differs from the common CMOS process, which constructs twin-well transistors. This solution has been extended to multiple-well processes, such as the quadruple-well process proposed in Hayden [1994] and the patented techniques of Snyder and Larson [2007, 2011], with the same fundamental idea.

Table II. Summary of Circuit-Level Mitigation

Approach	Year	Genre	Notes
Diehl et al.	1982	Resistive hardening	Only effective for alpha-originated SER
L cell	1987	Resistive hardening	Unclear effect in flip-flop and latch
Rockett cell	1988	Classic redundancy	1.4× transistors, 1.7× write time
Whitaker cell	1991	Classic redundancy	Great static leakage, 17%+ area
Rockett	1992	Resistive hardening	Unreliable SER reduction, 1.8× to 2.5× write time
Barry/Dooley design	1992	Classic redundancy	Lower power cost, larger area cost than prior
DICE	1996	Classic redundancy	2× area and power
Karnik	2002	Capacitance hardening	Limited SER decrease
Shuler et al.	2005	C-element	Limited SER reduction, 2× area
BISER	2006	C-element	Reusing DFT, 20× SER reduction
RCC	2010	Node engineering	Charge sharing, 3× SER reduction
LEAP-DICE	2010	Redundancy&Layout	2,000× SER reduction, 1.3× area
BCDMR	2010	C-element	Based on BISER, 150× SER reduction than BISER
EDS	2011	Classic redundancy	Timing failure detection
SEILA	2015	Redundancy & layout	1FIT/Mbit (0.001 alpha/hour/cm^2)

increases for flash memory [Bourdelle et al. 2002]. Numerous patented TW solutions [Hsingya et al. 2013; Han et al. 2001; Yang and Huang 2000; Kar-Roy et al. 2006] with respect to a variety of devices (e.g., SRAM, DRAM, and Flash) have been filled by different institutions.

3.2. Mitigation at the Logic and Circuit Level

The state-of-the-art approaches at the circuit level primarily focus on two major methods to increase soft error resiliency. These approaches are briefly described in Table II. The first method optimizes (by raising the node capacitance or supply voltage) the sensitive node in the target circuit, which can contribute to a reduction of SER. In addition, the second method introduces redundant circuits into the target sequential circuit to enable recovery from soft errors in cases where soft errors occur.

3.2.1. Circuit Node Engineering. Selective node engineering is based on the relationship involving major electrical characteristics that affect SER at the circuit level. The proposed solutions have been categorized into three:

(1) Increasing the size of the transistors, of which the circuit node is composed, which can raise the capacitance and supply voltage [Mitra et al. 2005a]
(2) Coupling an additional resistor or capacitor to the underlying node, which contributes to the increase of the resistance or capacitance of the node, which further reduces SEU rates [Diehl et al. 1982]
(3) Forward-biasing the body-source junction of the node, which contributes to capacitance rise, depletion charge collection reduction, and feedback loop enhancement [Iyer et al. 2005]

Nonetheless, the downside of those approaches is that they may increase power consumption and/or degrade speed.

One of the earliest solutions to circuit-level soft error solutions was resistive hardening (polysilicon intracell resistors). At the circuit level, Diehl et al. [1982] conducted a study using simulations with an analytical model, for reducing RAM cell SER in the early 1980s. Diehl et al. [1982] proposed inserting additional resistors into the RAM cell. These additional resistors slow the feedback of gate voltage variations, which is necessary to reverse the logic state of the inverter pair in the RAM cell. This slowing effect is achieved by diminishing the coupling between the inverter pair. In other words, this insertion increases the time constant of the feedback path in the inverter

pair. The result in Diehl et al. [1982] showed that the technique of resistor insertion can effectively reduce the probability of SEU in the tested RAM. This solution is easy to implement and has little overhead in performance at room temperature. However, the merits of this solution might not hold for sequential cells other than SRAM.[19]

The L cell [Weaver 1987] is another seminal work in the category of resistive hardening.[20] The focus of the L cell is to address the radiation event located at the "off" n- and p-channel drains in SRAM. The L cell has additional resistors between the output and the input of the inverters in the SRAM cell. Apart from the normal feedback resistors, a pair of new resistors is placed between the p-drain and the information node in the inverters. The purpose of this placement is to enable voltage division. Therefore, the amplitude of the transient (SET) is limited to the values below the state switching point of the inverters. This condition (i.e., below state switching point) is necessary for avoiding state corruption in the SRAM cell. The main overhead of this solution is reduction in performance.

Reduced performance and the difficulty in using the technique in high-density SRAMs made resistive hardening eventually obsolete. Instead, Rockett [1992] proposed the use of gated resistors to harden SRAM cells in the early 1990s. This solution [Rockett 1992] places the gated resistors in the cross-coupled interconnect segments of the SRAM cell, in the same way the nongated resistors were implemented in conventional resistive hardening solutions. By switching the resistance to high and low levels, immunity to soft error is achieved (i.e., switch to high resistance), and the impact upon the write operation to the SRAM cell is reduced (i.e., switch to low resistance). The solution in Rockett [1992] can achieve lower overheads in terms of write time (i.e., a reduction by a factor of six against the conventional solutions in the best case) while maintaining the same level of effectiveness against radiation. However, in comparison to an unprotected cell, the write time increases greatly (around $1.8\times$ to $2.5\times$).

By using capacitance to achieve logic hardening, Karnik [2002][21] proposed selectively designing sequential or domino nodes in circuits for SER reduction. This approach adds an explicit capacitor on the feedback/keeper node (referred to as "fb"), which is naturally excluded by the d-q (input to output) path. Since the capacitor is not in the d-q path, the overhead in circuit timing is mostly (70%) noncritical. The stack-node technique splits the capacitors and switches them via the clocked devices in the feedback driver. The switching mechanism ensures that the explicit capacitor is automatically disconnected when the latch is in write mode. In addition, an algorithm was proposed to find suitable nodes that are not in the critical path, aiming to reduce circuit SER without sacrificing speed. The results showed that chip-level (i.e., a processor built in 180nm technology) SER decreased on average by a factor of 1.29, whereas power overhead increased by 2.5% with no speed degradation.

Reinforcing charge collection (RCC) design [Seifert et al. 2010] aims to increase Q_{crit} in a static storage cell, where a pair of cross-coupled inverters are deployed. In order to increase Q_{crit}, RCC design allows both the "ON" and "OFF" transistors in each inverter to collect charge together (referred as charge sharing). As "OFF" transistors (i.e., victim diffusion) are naturally capable of collecting charge, the key of RCC design is to engineer the circuit, so that the "ON" transistors have better charge collection efficiency. Therefore, RCC design inserts two pairs of dummy gates, that is, additional "OFF" transistors, which significantly increase the probability of charge sharing. RCC

[19]The work in Rockett [1988] has shown the impracticality of applying resistive hardening to latch cells, especially in terms of performance penalty.

[20]The detail of resistive hardening cell can be obtained in the patent in Houston [1990].

[21]The practical detail of using capacitive hardening in SRAM can be found in the description of the patent [Blake and Houston 1993].

design increases timing (6.4%), leakage power (28%), and area (10%) slightly and reduces SER (about 3× reduction) in comparison to a normal storage cell.

3.2.2. Redundant Circuit Design. Deploying redundant state elements (storage cells) has been seen in a large number of research publications and patents as a viable solution for mitigating soft errors in sequential circuits. Those solutions can be divided into two:

(1) Classic latch or RAM cell design including the Rockett cell [Rockett 1988], Whitaker cell [Whitaker et al. 1991], Barry/Dooley design [Barry 1992], and dual interlocked storage cell (DICE) [Calin et al. 1996] of the 90s
(2) Incremental explorations in the last decade, including publications [Shuler et al. 2005; Zhang 2006] and patents [Shuler, Jr. 2002a, 2002b].

In the late 1980s, the Rockett cell [Rockett 1988] was proposed to harden the latch design to gain SEU protection. In this solution, the immunity to SEU is obtained by adding six PMOS transistors to the baseline design, which consisted of 10 transistors. This solution uses clock signal levels to load and offload the state-redundant transistors. During the opaque (or retention) phase, loading the state-redundant transistors prohibits the high data state of the latch from discharging. In this way, the state of the latch cannot be changed upon a soft error event. During a write-latch operation, offloading the state-redundant transistors enables the state of the latch to be updated. The Rockett cell increased the number of transistors by almost 40% and induced an approximately 1.7× write response time in comparison to the unhardened latch cell.

The Whitaker cell [Whitaker et al. 1991][22] is a RAM cell with two storage structures, with a special configuration of strengths of transistors in the cell (called "ratioing"). The Whitaker cell is motivated by the following observation: SET-induced current flows are always directed from the n-type diffusion to the p-type diffusion through the n-p junction. If memory cells were created with transistors of a single type, then p-transistors (PMOS) can safely store 1 while n-transistors (NMOS) can safely store 0. Therefore, one storage structure is implemented with p-transistors, while the other storage structure is implemented with n-transistors. To realize the special configuration of transistor strengths, the Whitaker cell must size the transistors carefully. Although the Whitaker cell achieves SEU immunity, it can reduce signal levels due to the presence of PMOS pull-down transistors and the presence of NMOS pull-up transistors. The degraded signal levels can result in significant static leakage current. As a two-storage structure, the area overhead is 100% (six more transistors) over a normal 6T-SRAM cell.

Following the design of the Whitaker cell, the low-power Whitaker cell [Liu and Whitaker 1992] was proposed to reduce power consumption. The low-power Whitaker cell adds two sets of complementary devices (transistors) between the power supply VDD (VSS) and n-type (p-type) storage structures, respectively. Further, the low-power Whitaker cell has two fewer input pass transistors. In comparison to the original Whitaker cell, the low-power cell has two more transistors in total, equivalent to a 17% area increase over the unimproved one. The reduction in static current was shown to be more than four orders of magnitude, obtained by the use of SPICE simulations (HP $1.0\mu m$ double-metal CMOS technology).

DICE [Calin et al. 1996] achieves SEU resistance while alleviating the drawback of the Whitaker cell, that is, fixed transistor strength ratio, which might render the Whitaker cell inapplicable to high-density systems. The merits of DICE include (1) imposing no particular constraint on transistor sizes and (2) being applicable to

[22]The Whitaker cell has been patented. More technical details can be found in the patent description [Whitaker 1992].

both CMOS SRAM and other sequential logic circuits such as latches and flip-flops. DICE takes advantage of the idea of dual-node feedback control. This idea leads to the implementation of a four-node storage structure, where each node is a pair of cross-coupled inverters. The logic state in each one of the four nodes in DICE is controlled by two adjacent nodes. Upon an SEU, which tries to flip the state bit of a single node,[23] the interlocked interconnection between the inverters can stop the contaminated signal from finishing a feedback loop. Simultaneously, the redundant uncorrupted data is sent to recover the affected node. However, considering its high cost in area (100% overhead in comparison to normal cells) and power,[24] the potential application largely lies in the selective deployment in the critical paths of a system.

Targeting the modern 40nm bulk technology node, DICE was studied again by Loveless et al. [2011] in 2011. This study examined a DICE flip-flop in comparison to a normal D flip-flop under neutron and proton exposure. The results showed that DICE only reduces SER by 30% to 50%, in comparison to a normal D flip-flop. Loveless et al. suggested that (1) the reason is that the charge-sharing mechanism between transistors in a DICE flip-flop raises the vulnerability to SEU and lowers DICE's effectiveness and (2) the effectiveness of DICE will continue diminishing as the bulk technology node scales down.

The Whitaker cell and DICE share a few major drawbacks for low-power applications [Maki et al. 2003]:

(1) Due to the degraded signal levels in the storage structure, it is difficult to lower the supply voltage.
(2) Operating at low supply voltage requires the transistors to have low threshold voltages. Therefore, the subthreshold leakage of these transistors may be quite considerable.

To overcome those drawbacks, the Barry/Dooley design [Dooley 1994; Barry 1992] was proposed. The Barry-Dooley design is a four-node design and only needs one state transition to enable recovery for any one of the four nodes upon a radiation event. To support this feature, the n-transistors only pass logic value 0 and p-transistors only pass logic value 1. However, in comparison to the prior mitigation methods using redundancy (such as the Whitaker cell and DICE), the Barry/Dooley design has a higher area cost. For example, a Barry/Dooley latch has 16 transistors, while a DICE latch has eight transistors.

In order to overcome the area and power overhead induced by redundancy, a number of pieces of work are proposed to economically design circuits for data redundancy in sequential circuits. Built-In Soft-Error Resilience (BISER) [Zhang 2006] is one seminal work in this category. BISER considers reusing the underutilized design-for-testability (DFT) and design-for-debug logic circuits. For example, a large number of DFT circuits, in the form of scan latches, can be added by utilizing scan chain methodology.[25] The BISER design is the composition of two flip-flops joined with a C-element.[26] During normal operation (not DFT operation) of the circuit, a group of the scan latches, called the second latches, are unused. BISER can reuse those latches during normal operation

[23]The state bit-flip means the turning from logic 0 to 1 or vice versa.
[24]DICE draws a large amount of current because the circuit has to go through many high-current transitions from an unstable to a stable state.
[25]Scan chain [Crouch et al. 1997; Kim and Schultz 1996; Williams and Angell 1973] has been a de facto standard DFT methodology, which can provide high-quality testing at low overhead in the processor CAD procedure.
[26]C-element is developed by Muller and Bartky [1959] as a two-input SET filter in asynchronous circuit theory.

as redundant data storage. The simulation result has shown that BISER achieves a reduction by a factor of 20 in SER, with increased power consumption.[27]

Other than using conventional C-element, Shuler et al. [2005][28] proposed the use of "transitions and/or NAND gate," called TAG, to protect latch cells. TAG has two inputs, one output, and a simple structure, that is, a series of "AND" wired transistors. The output of TAG only transitions when both inputs have transitioned. Hence, TAG can be used as a special voting circuit, which votes on three values: the existing TAG output value and the two new values at two inputs. Shuler et al. [2005] reported an implementation of a four-TAG latch design, which is a "two-storage structure," similar to the Whitaker cell. In the four-TAG latch, the inverters, which hold the logic value in the latch, are carefully replaced with TAGs. This replacement allows (1) the two input signals to be voted when setting the latch and (2) the original and redundant latches to be cross-coupled and vote on each other's internal state. Hence, four-TAG latch can overcome both transient fault occurrences in the input signals and the internal circuit. In comparison to the Whitaker cell, the four-TAG latch does not involve any bias currents or critical transistor sizing. Therefore, the four-TAG latch has lower overheads in both power and performance. However, as a two-storage structure, four-TAG latch doubles the area of the latch.

Based on the BISER design, the Bistable Cross-coupled Dual Modular Redundancy (BCDMR) flip-flop [Furuta et al. 2010] further enhances the soft error immunity by protecting the C-element from the bit-flip caused by particle strike. The BCDMR design duplicates the C-element in a cross-coupled fashion. As a result, the SET from one particle hit on one C-element cannot flip both the original and the additional flip-flop in BISER. At 65nm technology node, the BCDMR flip-flop shows 150× better soft error immunity than the BISER flip-flop when tested at 160MHz clock frequency, with almost the same area, power, and performance.

LEAP-DICE [Kelin et al. 2010; Lee et al. 2011], where LEAP is short for "layout design through error-aware transistor positioning," utilizes both the LEAP layout design and existing DICE circuit design. The LEAP layout design places transistors in such a way that allows the transistors in the inverters of one storage cell to have a combined response to a particle strike. The combined response reduces charge collection, which keeps excessive charge below Q_{crit}. With the LEAP layout, LEAP-DICE can cover MBU, which DICE alone cannot cover. Experimental results showed that LEAP-DICE could achieve 2,000× and 5× SER reduction in comparison to a normal design and DICE. LEAP-DICE incurs almost negligible timing overhead while introducing 133% more area and 54% more power, in comparison to a normal design. A more recent implementation of LEAP-DICE with 20nm bulk technology is described in Lilja et al. [2013].

Error Detection Sequential (EDS) circuit design [Bowman et al. 2011] is able to detect critical path timing failure due to dynamic variations, which are typical for near-threshold operation with lower supply voltage. The EDS design inserts a shadow flip-flop, which samples the input data on a different clock edge in comparison to the datapath latch. The data values from the shadow flip-flop and datapath latch are compared using one XOR gate. The mismatch of the values will signal an error. In one pipeline stage, multiple EDS circuits will be embedded for different datapath latches. The error signals from each EDS circuit in one pipeline stage are ORed together

[27]While applying selective BISER deployment in flip-flops, chip-level (microprocessor) SER can be improved by 10×, with 10% power overhead.

[28]The patents related to this work can be seen in Shuler, Jr. [2002a, 2002b]. More details can be obtained in the patents.

to generate one pipeline error signal. The EDS design results in 3.8% area overhead and 2.2% power overhead.

SEILA [Uemura et al. 2015] is a hardened latch design targeting the specific issue, which is the charge collection at multiple storage nodes (CCM) in latches. CCM can raise errors in previous redundancy-based hardened latch design (e.g., DICE). SEILA is designed on top of DICE. To overcome CCM in DICE, SEILA utilizes multiple height cell design (MHC), which is a layout optimization. MHC is based on the observation that the charge collection can be canceled due to the connection between the areas of PMOS drain and NMOS drain. Moreover, SEILA also mitigates single event transient in local clock (SELTC) with a dual-clock buffer design (DCB). SEILA's SER is less than 1 FIT/Mbit in the experiment with 0.001 alpha/hour/cm^2.

3.3. Mitigation at the Architecture Level

As a large-scale and highly complex organization (in terms of the number of transistors), a processor system has two major distinct architectural components: (1) the large data storage, for example, the register-file, cache, and memory, and (2) the pipeline including the functional units, such as the arithmetic logic unit, multiplier, and divider. Therefore, we divide the discussion of the architecture-level mitigation (summarized in Table III) into two major categories:

(1) The mitigation methods for **storage architecture**. These methods exploit the properties of storage and target data corruption by the use of coding theory in the storage elements.
(2) The mitigation methods considering pipeline design or entire **processor architecture**. These methods are varied and attempt to achieve correct execution of instructions in a processor (primarily the pipeline execution), as well as maintaining the correct values in the system during execution.

In addition, the architecture-level soft error mitigation methods can be categorized as error detections and error recovery:

(1) **Error detection.** Error detection is a prerequisite to resolving soft error effects. The metrics for evaluating an error detection technique include the latency, coverage, and overhead in terms of area, performance, and power. Error detection is noted in Table III as "D."
(2) **Error recovery.** Error recovery is the actual process of correcting the manifestation of an error. The metrics for evaluating error recovery techniques are similar to error detection. However, the overhead of error recovery has two aspects: the fault-free overhead and the actual recovery overhead. Error recovery is noted in Table III as "R."

3.3.1. Storage Architecture. Storage architectures account for a large portion of soft error occurrences in a processor-based system. In fact, all the early soft-error-related R&D efforts were focused on SRAM and DRAM storage, since soft errors were only prominent in storage at that time (this point is discussed in prior sections). Considering the storage architectures as "a large lookup table" with "fragile table elements," the storage architectures needed to be rigorously handled when affected by soft error. The handling method is essentially about how to organize the "table element," that is, the information. Therefore, mitigation methods in storage architecture are heavily backed by coding/information theories and mathematical properties [MacKay 2002; Shannon 2001].

Table III. Summary of Architecture Level Mitigation

Approach	Year	Architecture	Function	Notes
Residue codes	1966	Processor (ALU, FPU)	D&R	Coding
AN codes	1972	Processor (ALU)	D	Coding
Parity check	1977	Storage	D	Coding
RESO	1982	Processor (ALU)	D	Temporal redundancy
SEC-DED	1984	Storage	D&R	Coding
CARER	1987	Processor	R	Spatial redundancy
RAID	1988	Storage	D&R	Coding
DEC-TED	1989	Storage	D&R	Coding
CARER+	1990	Processor	R	Spatial redundancy, multiprocessor
Chipkill	1997	Storage (DRAM)	D&R	Coding
TEC-QED	1998	Storage	D&R	Coding
RAID-5	1999	Storage	D&R	Coding
AR-SMT	1999	Processor	D	Temporal redundancy
DIVA	1999	Processor	D&R	Spatial redundancy
IBM S/390 G5	1999	Processor	R	Spatial redundancy
Blough et al.	1999	Processor (datapath)	R	Spatial redundancy, high-level synthesis
SRT	2000	Processor	D	Temporal redundancy
Slipstream	2000	Processor	D	Temporal redundancy
SRTR	2002	Processor	D	Temporal redundancy
SafetyNet	2002	Processor	R	Spatial redundancy, multiprocessor
ReVive	2002	Processor	R	Spatial redundancy, multiprocessor
CRTR	2003	Processor	D	Temporal redundancy
Razor	2003	Processor	D&R	Circuit & architecture, asynchronous clock
Weaver et al.	2004	Processor	NA	Circuit & AVF analysis based
RPR	2004	Processor	D&R	Spatial redundancy
RSE	2004	Processor	D&R	Spatial redundancy, hardware assertion
ReStore	2006	Processor	D	Fault symptom based
TAC&RNA	2006	Processor	D	Hardware assertion
PBFS	2007	Processor	D	Fault symptom based
Argus	2007	Processor	D	Signature based
Glosli et al.	2007	Processor (GPU)	R	L1 cache
DeLorean	2009	Processor	D&R	Spatial redundancy
Yoon&Erez	2010	Storage	D&R	Coding
Reli	2012	Processor	R	Spatial redundancy, custom instruction
Li et al.	2013	Processor	NA	IVI analysis based
Palframan et al.	2014	Processor (GPU)	NA	Floating-point precision
FaultHound	2015	Processor	D&R	Fault symptom based

D: Error detection.
R: Error recovery.
NA: Not applicable.

The simplest mitigation method for storage is the use of parity bit or parity check.[29] Parity bit is implemented by inducing one more redundant bit to one piece (e.g., 1 byte) of information (in the form of binary bits). The value of this redundant bit is assigned according to the number of bits, which are 1s, in the original information. By assigning either value 0 or 1 to the redundant bit, the information is specially encoded to have one invariable property, that is, the parity. Upon a soft error, in terms of a bit-flip, parity bit can detect the error by recomputing the parity of the information when fetched (a decoding process). However, in terms of MBU, parity bit may not be able to detect the error.

[29]Also referred to as single error-detecting codes by Hamming [1950].

The parity bit uses either an even or odd scheme to add the redundant bit. For instance, for the even parity scheme, if the parity becomes odd, it means that one bit of the information is flipped (corrupted). Thus, the error can be detected before it is propagated any further. Theoretically, parity bit is only effective on the condition that the error effect, namely, the bit-flips, causes the corrupted information with Hamming distance[30] of 1. This property roots in the mathematical property of parity: (1) only odd number addition can modify the parity of the number being added, either from odd to even or from even to odd, and (2) the redundant information is not enough to calculate the exact odd number. Despite this limitation, parity bit is widely applied in practice.[31]

For protecting hard disks against soft errors, parity bit is further extended to parity data in the Redundant Arrays of Inexpensive Disk (RAID) technique [Chen et al. 1994; Patterson et al. 1988].[32] RAID is a large-scale redundant system of disks with high reliability. Parity data, in the form of redundant disks, is involved in restoring the content of the other disks. The property of exclusive-or is utilized in parity data:

if the equation

$$\text{data1} \oplus \text{data2} = \text{parity\_data} \tag{5}$$

is true, then the following two equations:

$$\text{data1} = \text{parity\_data} \oplus \text{data2} \tag{6}$$

and

$$\text{data2} = \text{parity\_data} \oplus \text{data1} \tag{7}$$

must be true. Parity data can guarantee that the content of any individual disk can be recovered by the content of the other two disks.

Superior to parity bit, the mitigation method called error-correcting code (ECC)[33] provides both detection and correction for variable numbers of bit-flips. ECC is similar to parity bit in that it also relies on bitwise exclusive-or operation across the original data to produce the redundant data. When the data under protection is requested, the data can be decoded by ECC to detect and recover bit-flips. The decoding mechanism first tests the value of the redundant bits to rule out the error-free case (i.e., syndrome $=$ 0) and then compares the redundant bits with the ECC word to locate and correct the erroneous bits, in case of bit-flips (i.e., syndrome $\neq$ 0).

From a mathematical perspective, ECC is a type of binary linear code. Therefore, the detection and recovery capability, in terms of the coverage, is bounded. Suppose the minimum Hamming distance is d. The capability of one ECC code follows the rule:

$$\text{N\_correct} = n, \quad \text{N\_detect} = n + 1, \quad \text{if and only if} \quad d > 2n + 1, \tag{8}$$

where N_correct stands for the number of correctable bit errors and N_detect denotes the number of detectable bit errors. There are a range of ECC methods proposed with varied configurations and coverage, including SEC-DED,[34] DEC-TED,[35] and

[30]Hamming distance is the minimum number of substitutions demanded to transform one string into the other, or the minimum number of errors for transforming one string into the other [MacKay 2002].

[31]Such as the early version patented in Kim [1977] and Barker [1977], and later improved versions, two-dimensional parity in Kemmetmueller [1980] and DES parity in Marino, Jr. [1981].

[32]RAID is a combination of several error-detecting and -correcting codes. In practice, the RAID-5-based technique can be seen in the patent [Mann et al. 1999].

[33]Also referred to as Hamming codes [Fujiwara and Pradhan 1990; Hamming 1950].

[34]SEC-DED denotes single-error-correcting and double-error-detecting, which is patented by IBM [Bannon and Bhansali 1984].

[35]DEC-TED stands for double-error-correcting and tripe-error-detecting. It is early patented by IBM [Chen 1989].

TEC-QED.[36] In the case of MBU, the SEC-DED scheme might not easily meet the reliability goal, and hence, the ECC scheme must be carefully selected.

At present, in industry circles, one typical state-of-the-art ECC scheme for off-chip DRAM is chipkill-correct [Dell 1997] from IBM. The main bottleneck of chipkill-correct is that it either increases memory access granularity, requiring more energy and restricting possible DRAM configurations, or increases the required level of redundancy, which again increases cost [Ahn et al. 2009]. Another good example in practice is the new Fermi graphics processing units (GPUs) architecture [Nvidia 2009][37] from Nvidia, which extensively deploys ECC support (i.e., SEC-DED) on a wide range of storages, including register files, L1 and L2 caches, and shared and DRAM memories. In academic circles, the recent advances in off-chip memory ECC have shown great progress in DRAMs [Yoon and Erez 2010].

In summary, the coding-based soft error mitigation methods, such as parity bit and ECC, have prevailed in storage architecture since the early ages of computers. The reason is twofold:

(1) Timing. The encoding and decoding in ECC cost timing, and thus reduce the storage's performance, in terms of read/write latency. Because this latency is only induced at the read/write cycles of the storage, the timing overhead might not be substantial to the entire system.
(2) Hardware. The encoding and decoding in ECC are usually implemented in hardware. Hence, the relevant hardware costs area and power. However, in a typical storage, such as SRAM memory, the number of read/write ports is limited. Therefore, the hardware overhead of encoding and decoding is actually amortized across the entire data arrays.

On the other side, the drawbacks of these coding-based soft error mitigation methods are also prominent:

(1) Timing. According to Amdahl's Law,[38] the performance cost to the entire system will be larger if the protected storage is more frequently visited. Considering an ECC circuit with a 3.2ns access time penalty [Osada et al. 2003], ECC methods will dramatically slow down the processor pipeline, where the storage is visited very frequently.
(2) Hardware. The residence of the redundant bits also occupies a considerable amount of logic circuits. The hardware overhead, in terms of area and power, increases with the volume of data in the storage, even though the hardware overhead decreases as the width of data increases. For example, ECC on SRAM with 128-bit word-size will only incur 9.7% area overhead and 19mW power dissipation (with a 70ns cycle) [Osada et al. 2003]. In addition, given that the structure of a pipeline is usually composed of many latches with varied data widths, the encoding and decoding circuits can be very costly as well.

3.3.2. Processor and Pipeline Architecture. Similar to the storage architecture classification, the architecture-level soft error mitigation methods for processors and pipelines can also be divided into error detection and recovery. The architecture-level mitigation methods are mostly based on redundancy. However, for error detection in

[36]TEC-QED is short for triple-error-correcting and quadruple-error-detecting. The application in practice can be seen in the patent [Babb 1998].
[37]The web resource about Fermi architecture can be obtained by visiting http://www.nvidia.com/object/fermi-architecture.html.
[38]Also known as Amdahl's argument [Amdahl 1967], it is widely adopted to model the maximum expected improvement to an overall system, given the improvement in a part of the system.

arithmetic units, for example, for ALU [Avizienis 1971; Garner 1966] and FPU [Lipetz and Schwarz 2011], coding-based techniques are typically used.

The architecture-level redundancy is similar to the circuit-level redundancy, which duplicates (or triplicates) the target system. The computation results from the system with redundancy can be compared (or voted) to mitigate errors. There are two major ways to implement architecture-level redundancy: temporal (time) redundancy and spatial (space) redundancy. Temporal redundancy-based mitigation methods allow a computation to be performed multiple times on the same hardware and compare or vote on the computation results [Sohi et al. 1989]. These mitigation methods essentially trade performance for reliability (or the ability to mitigate soft errors). The key drawback of temporal redundancy is performance loss. For example, a typical temporal redundancy-based mitigation method using multithreading can induce nearly 40% overhead in performance even in fault-free situations (i.e., without error occurrence).

Spatial redundancy-based mitigation methods do not require multiple repetitions of the computation, but fully or partially duplicated hardware is used to implement redundant computation. Spatial redundancy-based mitigation methods trade space or hardware to gain reliability. Both redundancy-based mitigation methods follow the classic fault-tolerance idea of N-modular redundancy, such as dual modular redundancy (DMR, for error detection), triple modular redundancy (TMR, for error detection and recovery), or checkpoint recovery[39] (for error recovery) [Avizienis et al. 2004].

Besides, there are other mitigation techniques based on concepts such as signature, symbol, and so forth. In this subsection, we will first discuss the existing coding-based and redundancy-based mitigation methods, and then introduce the techniques based on the concepts other than redundancy.

Arithmetic codes can be used to protect arithmetic operations. As the simplest form of arithmetic codes, AN codes [Peterson and Weldon 1972] can be applied to detect errors in addition or subtraction operations. An AN code is calculated by multiplying a data word N by a constant A. For a given arithmetic operation op and two data words N_1 and N_2, the usage of AN code can be expressed as the following equation: $A \cdot (N_1 \,\mathrm{op}\, N_2) = (A \cdot N_1)\,\mathrm{op}\,(A \cdot N_2)$. The left side is the code generated by the result of the arithmetic operation, while the right side is the code generated by the operands of the arithmetic operation. By comparing the two sides, the error during the arithmetic operation can be detected (i.e., mismatch). By changing the choice of A, different codes will be generated. AN codes cannot cover logical operations.

As another type of arithmetic codes, residue codes [Garner 1966] are classically used for protecting arithmetic units from errors in a processor architecture. The power of error detecting and correcting of a residue code is related to the minimum arithmetic distance between distinct code digits, namely, d_{min}. A linear residue code can correct all patterns of t or fewer errors if and only if it has minimum distance at least $2t + 1$ [Massey and García 1972]. For example, a single-error detection and correction residue code [Brown 1960] is a linear residue code with $d_{min} = 2$. One successful example of residue codes in practice is the IBM POWER6 processor [Sanda et al. 2008], where residue codes are applied in floating-point units. However, residue codes are not capable of detecting some errors in logical operations [Patel and Fung 1982].

Other than arithmetic codes, parity prediction [Nicolaidis et al. 1997] can detect errors in addition, subtraction, multiplication, and division operations by leveraging the properties of carry chains. Two parity prediction values are computed to detect errors for an operation. The first value is computed using the source operands of the operation, while the second value is computed using the result of the operation. In

[39]At the architecture level, it is also referred to as backward error recovery (BER).

practice, parity prediction has been deployed in a Fujitsu Sparc64 processor [Ando et al. 2003a, 2003b].

One of the early works in the category of temporal redundancy and arithmetic unit in processor architecture is RESO (REcomputing with Shifted Operands) [Patel and Fung 1982]. RESO implements error detection for arithmetic logic units (ALUs) in a pipeline. RESO uses left-shift operation as the coding function and right-shift operation as the decoding function. In each ALU computation, the operands are first unshifted and computed. The temporary result at this step is stored in a redundant register. At the next step, called recomputation, the operands are encoded and then computed by ALU. This result is decoded and then compared with the temporary result of the first step in the redundant register. A mismatch in the comparison indicates an error. The only drawback of RESO is that it only covers the execution stage of the processor pipeline.

In the early 2000s, with the advent of simultaneous multithreading (SMT) [Marr et al. 2002] processor technology, which targets high performance in general-purpose processors (GPPs), new mitigation methods mostly focus on temporal redundancy-based soft error mitigation for SMT architectures. These mitigation methods were mainly inspired by the possibility of utilizing additional resources in the SMT architecture, which enables redundant program execution.

AR-SMT [Rotenberg 1999] is the first to use SMT to execute two copies of the same program. AR-SMT utilizes the delayed execution on a duplicated redundant program, called instruction stream, to realize an error detection mechanism. Error detection is implemented by comparing the update of the architectural state (e.g., PC, registers, and memory) made by the original program (called A-stream) and the redundant program (called R-stream). The delay time is determined by the length of the delay buffer. Since AR-SMT assumes an SMT baseline processor architecture,[40] the overhead for error detection is less, in comparison to implementing temporal redundancy in other processor architectures, for example, single-issue in-order architecture. The performance overhead for a single thread with AR-SMT is about 10% to 30% in simulation.

SRT (simultaneous and redundantly threaded) [Reinhardt and Mukherjee 2000] was proposed after AR-SMT. Similar to AR-SMT, SRT also aims at transient error detection by executing identical copies of the same program simultaneously as independent threads. In addition, SRT achieves higher performance because it dynamically schedules the hardware resources among the redundant copies. However, dynamic scheduling makes SRT difficult to work with lock stepping, because the instructions from redundant threads might not be executed in the same order as the instructions from the original threads.

In comparison to AR-SMT, SRT has two new mechanisms implemented, that is, slack fetch and branch outcome queue. These two mechanisms can enhance the performance of an SRT processor by allowing one thread to prefetch cache misses and branch results for the other thread. According to the experimental results, an SRT processor provides better performance (16% increase on average) than lockstepping, that is, cycle-by-cycle instruction output comparison on fully duplicated hardware. SRT has been patented by Intel in Mukherjee et al. [2008] and Reinhardt et al. [2008].

SRTR (simultaneously and redundantly threaded processors with recovery) [Vijaykumar et al. 2002] aims to improve SRT in two aspects, since (1) SRT experiences unexpected pipeline stalls, because the leading nonstore instruction (A-stream) may commit before the checking occurs, and the detection can only be activated by the trailing thread (R-stream), and (2) it is difficult (in terms of bandwidth pressure) for SRT to always check the results written to registers, because at the time of checking,

[40]The A-stream and R-stream can be processed simultaneously by SMT architecture. Hence, it allows the results from R-stream to be compared with that of A-stream while fetching and executing procedures.

the results have been updated to register-file. The respective improvements of SRTR are (1) checking is performed as soon as the trailing instruction finishes, by exploiting the time between the completion and committing of the leading instruction, and (2) register value queue (RVQ) is added to store register values and other information for checking. The recovery mechanism is similar to the technique in Ray et al. [2001], which relies on interrupts and speculative execution. The performance of SRTR can be 26% worse than SRT. In addition, a number of queue structures must be added into the system.

For chip multiprocessors (CMPs), Slipstream [Sundaramoorthy et al. 2000] and CRTR [Gomaa et al. 2003] (chip-level redundantly threaded multiprocessor with recovery) were proposed to extend the previous SMT-based mitigation methods. Slipstream is closely related to AR-SMT. The key idea is to create a "shorter but otherwise equivalent version of the original program." To create this shorter program, it is necessary to identify the removable computation that is ineffectual or form highly predictable control flows. Slipstream uses the special shorter version of the original program for two purposes: (1) to partially duplicate instruction executions for transient fault tolerance and (2) to improve performance. At runtime, the Slipstream processor executes the shorter redundant program (called A-stream) speculatively before the original program (called R-stream) and feeds back the control and data flow outcomes to the original program. The error coverage in Slipstream is limited to the faults that cannot flip the architectural state bits for R-stream. Slipstream has a performance improvement of about 7% on average over the baseline.

CRTR minimizes the performance overhead of soft error mitigation in CMPs, which is caused by interprocessor communication required for comparing duplicated threads. In CRTR, the error detection mechanism is based on SRT, while the error recovery mechanism is based on SRTR. Similar to Slipstream, CRTR executes a leading thread before a trailing thread by maintaining a long slack. This way of committing is called asymmetric commit. CRTR commits the trailing thread after checking the results. Therefore, the architectural state of the trailing thread can be used for recovery. CRTR limits the bandwidth requirement (for interprocessor communication) to 7.1 bytes/cycle. The performance overhead of CRTR in comparison to a baseline CMP is 10% at best and 60% at worst.

Existing architecture-level soft error mitigation methods exploit spatial (time) redundancy by using completely redundant pipelines (full duplication), reduced redundant pipelines (partial duplication), or checkpoint recovery. The first two types of solutions can be used to detect and recover errors, while the last one only aids error recovery.

At the architectural level, the earliest and most classical mitigation method using spatial redundancy is lockstepping (or lockstep) [Klecka et al. 2002]. Lockstepping is widely seen in various commercial fault-tolerant processors, such as Compaq (HP) NonStop Himalaya [Wood 1999] and IBM S/390 G5 [Slegel et al. 1999], to detect any transient faults. Lockstepping fully duplicates the original processor (or pipeline) and synchronizes the comparison of instruction output cycle by cycle. Upon a mismatch of instruction outcome, recovery operations are performed. This technique is a spatial DMR and increases hardware overhead greatly, due to the duplication (at least $2\times$ hardware), and also reduces performance slightly due to output comparison and synchronization.

One typical spatial redundancy-based mitigation with partial duplication is DIVA [Austin 1999]. DIVA uses a back-end checker to verify instruction results. The checker is essentially a small and functionally simple coprocessor, running in parallel with the original processor. Besides the purpose of dynamic verification and debugging, DIVA can detect and correct soft errors. To function correctly, DIVA makes two assumptions: (1) the registers and memory are protected with ECC, and (2) instructions must be

correctly passed to the checker. For error detection, the DIVA checker has two pipelines for checking. The CHKcomp pipeline verifies the integrity of the computation of all functional units in the original processor, while the CHKcomm pipeline verifies the communication between register, memory, and processor. To supply the inputs to the checker pipelines, DIVA requires additional register and memory ports dedicated to the checker. DIVA reports errors by raising exceptions. These exceptions are handled by checkers by fixing the corrupted values, provided by CHKcomp or CHKcomm. In experiments, DIVA showed about a 0.7% increase in execution time, in a fault-free scenario. When an error is detected, DIVA incurs at least eight cycles for handling the error exception.

Reduced Precision Redundancy (RPR) [Shim et al. 2004] is another seminal work based on partial spatial redundancy, which is a design of a soft-error-resilient datapath. RPR roughly follows the concept of algorithmic noise tolerance for digital signal processing systems. In RPR, a replica of the baseline datapath, with reduced precision operands, is implemented. The output of the replica is used if a soft error is detected in the main datapath. In order to avoid soft error in the replicated datapath, the critical path delay of the replica must be smaller than the sample period of the main system.

As an important class of spatial redundancy-based mitigation, checkpoint recovery[41] is implemented in hardware and software to recover transient faults.[42] Practical examples of the architecture-level hardware-based checkpoint recovery can be found in the IBM S/390 G5 microprocessor design [Slegel et al. 1999],[43] IBM POWER6 [Sanda et al. 2008],[44] and Fujitsu's Sparc64 processor design [Ando et al. 2003a, 2003b]. The goal of checkpoint recovery is to roll back the state of the processor as soon as the occurrence of transient faults [Siewiorek and Swarz 1998] is detected. Checkpoint recovery techniques typically use additional storage (assumed to be reliable and fault-free) to hold the checkpoint data. Checkpoint data is the correct processor state saved after the most recent check. Hence, if an error occurs, the previous processor state can allow the processor to go back to a previous point where the most recent checkpointing was performed. Checkpoint recovery requires additional storage for holding checkpoint data and special hardware for rollback. Moreover, checkpoint recovery impacts the execution time for saving the processor state in the fault-free scenario and for rollback after errors are detected.

Checkpoint recovery was investigated for the datapath generation (typical in a processor system) in Blough et al. [1999] from a high-level synthesis perspective. Blough et al. [1999] studied the problem of selecting a "good" set of recovery points for a given datapath, modeled as a control data flow graph (CDFG). Based on the branch-and-bound exploration algorithm, Blough et al. [1999] proposed two specific algorithms utilizing different cost functions (focusing on functional unit cost and/or register cost) to solve the recovery point insertion problem. These two algorithms achieve optimal checkpoint recovery datapath design with short computation time (generated a design in less than 10 minutes).

CARER (cache-aided rollback error recovery) [Hunt and Marinos 1987] focuses on checkpoint recovery for memory content by modifying the management of cache. In CARER, cache is treated as the checkpoint storage for holding the data from unreliable computation (before error checking). These data will not be written back to memory

[41]Checkpoint recovery has been patented by various entities including HP [Fremont 1987] and IBM [Kogge et al. 1990].

[42]In this section, we mainly discuss the hardware-based checkpoint recovery and leave the software-based ones to Section 3.4.

[43]IBM S/390 G5 is also an example implementing full duplication of the pipeline (with interlock).

[44]A more advanced recovery technique by IBM based on POWER6 can be found in Gupta et al. [2009].

until error checking is performed and the data are deemed to be error-free. If any error is detected, dirty lines in the cache will be flushed. To facilitate this function, an additional status bit is used for each cache line. CARER does not need additional checkpoint storage for checkpointing memory by reusing existing cache. However, reusing cache also leads to additional memory traffic (especially with the associativity lower than four-way). CARER lacks the flexibility of selecting when to commit checkpoint, and hence, the checkpointing is relatively unpredictable. In addition, hardware modification of cache is necessary. CARER was extended by Ahmed et al. [1990] for multiprocessor systems by adding modifications to the shared bus.

Another seminal work dedicated to the multiprocessor systems is SafetyNet [Sorin et al. 2002]. SafetyNet is a lightweight global checkpoint recovery scheme. SafetyNet maintains multiple globally consistent checkpoints periodically (at a coarse granularity, i.e., 100,000 cycles) for shared memory multiprocessor systems. In a checkpoint period, SafetyNet logs memory and the coherence state in special buffers called checkpoint log buffers (CLBs) and the register state in register checkpoint buffers. All these buffers, memories, and caches are assumed reliable and fault-free. SafetyNet has three design considerations: (1) in order to reduce the amount of spatial redundancy, a large period for checkpointing is used; (2) for consistency in the shared memory, global logical time and "logically atomic" coherence transactions are used; and (3) to reduce fault-free performance cost, checking is pipelined and overlapped with normal execution. During the recovery, SafetyNet first discards all the state related to coherence transactions. Then, the memories undo the update by using the data from CLBs, caches invalidate all blocks written, and processors restore the registers by using the data from register buffers. SafetyNet has little fault-free performance overhead (0.1%). The additional cache access required by SafetyNet ranges from 0.3% for a million-cycle checkpoint interval to 4% for a 5,000-cycle checkpoint interval. SafetyNet requires large buffers to avoid severe performance loss (e.g., more than 50% performance drop with 256kB CLB).

Similar to SafetyNet, ReVive [Prvulovic et al. 2002] is also a checkpoint recovery method for shared memory multiprocessor systems. ReVive differs from SafetyNet in four ways: (1) ReVive's error coverage includes permanent faults, (2) ReVive does not require change of caches, (3) ReVive uses memory instead of CLB to log the checkpoint data, and (4) ReVive has more network and memory traffic. The maximum checkpoint data (log) size of ReVive is 2.5MB for each processor node. The worst-case fault-free execution time increase of ReVive is 22%. The recovery time (unavailable time) for ReVive can be as large as 59ms.

Reli [Li et al. 2012] explores customizing the instruction set and adding micro-operations, which perform checkpoint rollback functionalities, into instructions. The native instructions are enhanced to automatically save the processor state from being modified. The additional instructions are added to write the saved values back to the processor after errors are detected. The Reli processor can perform fine-grained checkpoint recovery at the basic-block level with moderate fault-free performance cost and rollback time (within 100 cycles). In conducted experiments, Reli assumes a control flow check technique [Ragel and Parameswaran 2006], which is implemented in instruction micro-operations for error detection.

Reliability and Security Engine (RSE) [Nakka et al. 2004] is a comprehensive technique for both reliability and security purposes, targeting superscalar processor architectures. On the reliability side, RSE performs error detection via control flow checking after an instruction is fetched. Such a check is implemented in special hardware called the instruction checker module and scheduled in the program by inserting a special CHECK instruction (requires extending the original instruction set). RSE's error recovery is page-level checkpoint recovery. Checkpoint data is collected by special hardware

called the data dependency tracking (DDT) module. The actual rollback mechanism is adapted from Xu [2003]. RSE's performance overhead is 8% on average for error detection (ICM), and 7% to 8% for error recovery (fault-free, DDT). The major source of performance overhead with DDT is the time spent in saving memory page.

Reddy et al. [2006] proposed two methods for checking transient faults in an out-of-order superscalar architecture. Both methods implement assertions to find the errors. One method is called register name authentication (RNA), which targets the rename unit in the processor. The other method is named timestamp-based assertion checking (TAC) aiming at the issue unit. TAC asserts the time-orderliness of the instruction execution by assigning timestamps to instructions at issue. The timestamp of one instruction is compared with the timestamps of the producer instructions. A mismatch indicates a fault in the issue unit. On the other hand, RNA has different checks including previous mapping check, write-back state check, and source register renaming check.

ReStore [Wang and Patel 2006] is an architecture-level soft error detection method using fault symptoms. The symptoms include exceptions, control flow misspeculations, and cache or transaction look-aside buffer misses. In ReStore, error recovery is assumed as an ideal checkpoint recovery, with no performance overhead. In comparison to conventional error detection via full duplication, ReStore trades fault coverage for error detection overhead, because detecting the symptoms does not require full duplication of the entire processor or pipeline. In the conducted experiments, ReStore, implemented in a superscalar processor, achieved a 2× increase in mean time between failures (MTBF), with little performance and area overhead.

Another architecture-level error detection method that does not require full duplication is Argus [Meixner et al. 2007]. Argus is a unified method, which focuses on four major types of errors in the processor: control flow error, data flow error, computation error, and memory access error. For these four types of errors, Argus implements four respective error detection mechanisms. In Argus, control flow and data flow checking rely on basic-block signature, called data flow and control signature (DCS), based on a previous technique [Meixner and Sorin 2007]. The signature is computed both offline and at runtime. A mismatch indicates an error. For computation checking, Argus has one subchecker and one state history signature (SHS) computation unit for each functional unit, including ALU, multiplier, and divider. At runtime, the results from functional units and corresponding SHS units are sent to subcheckers to indicate errors. For memory checking, Argus uses an adder checker to detect address calculation errors and uses a sign extension (RSSE) subchecker to detect data realignment errors. The area overhead of Argus in the processor core is 16.6%. The performance overhead is 3.9% on average.

Toward efficient soft error mitigation, several architecture-level soft error analysis/estimation methods [Mukherjee et al. 2003a; Biswas et al. 2005; Li et al. 2005; Sridharan and Kaeli 2009; Nair et al. 2012; Rehman et al. 2011; Nair et al. 2015] have been proposed. These analysis methods aim to guide the soft error mitigation to focus on the most vulnerable components in the pipeline, and hence effectively reduce the soft error rate.

Architectural vulnerability factor (AVF) [Mukherjee et al. 2003a] is a seminal work on soft error analysis, proposed by the researchers from the Massachusetts Microprocessor Design Center (MMDC), Intel. The key motivation of AVF is that some SEUs will not manifest any errors in program output. In AVF, the set of processor state bits is divided into two subsets. The first subset is architectural correct execution (ACE) bits. Any fault in the architectural components containing any ACE bits will cause errors in the program output. The other subset is called un-ACE bits. A fault corrupting un-ACE bits will not propagate to program output. AVF can be implemented into the performance

model for a processor. AVF has been used in a public performance simulator, called SimSODA [Fu et al. 2006], for Alpha processor architecture. Given the fact that nearly 50% of the silent data corruption errors come from the sequential circuits in Intel processors [Mitra et al. 2005b], researchers from Intel and Nvidia continued the study of AVF modeling for finding the most vulnerable sequential circuits rapidly and accurately in a processor [Raasch et al. 2015].

Based on AVF (typically for single-bit fault), Wilkening et al. [2014] modeled AVF for spatial multibit transient faults, that is, MB-AVF. The MB-AVF of a hardware component is defined as the probability that a fault of a given multibit fault mode, in the corresponding hardware component, will propagate to a visible error in the program output. The GPU vector general-purpose register file (VGPR) is tested as a case study of MB-AVF's application in GPUs.

Weaver et al. [2004] proposed two techniques leveraging AVF to reduce SER in high-performance microprocessors. The first is instruction squashing, which reduces the time spent by ACE bits in a vulnerable architectural component. Upon a pipeline stall, instruction squashing selectively removes instructions from the instruction queue and brings the instructions back when pipeline resumes execution. The second technique is tracking, which tracks un-ACE bits by adding a new bit called π bit into the pipeline. Tracking allows the processor to avoid false error recovery for bit-flips in un-ACE bits, and hence increases the efficiency of soft error mitigation.

Following Weaver et al., there are other analysis-based architecture-level mitigation methods, such as cost-efficient runtime methods for SER reduction in single processors [Li et al. 2013a] and in multicore System-on-Chips [Li et al. 2013b] based on the instruction vulnerability index (IVI) [Rehman et al. 2011], as well as hardening SpaceWire computing node [Seshia et al. 2007] using formal verification (symbolic model checking) [McMillan 1993].

Perturbation-based fault screening (PBFS) [Racunas et al. 2007] is a runtime detection mechanism that probabilistically examines the processors' states to see whether the states have been corrupted by transient faults. The difference between the established and abnormal program behavior is defined as a perturbation. If a perturbation is found, the pipeline will be flushed and the instruction will be executed again. A perturbation might come from normal program behavior without any error. In this case, the pipeline flush will still happen and incur execution time overhead.

Inspired by PBFS [Racunas et al. 2007], FaultHound [Nitin et al. 2015] is a value-locality-based soft fault-tolerance scheme, including five mechanisms with regards to detection and recovery. FaultHound targets false-positive rate reduction and recovery overhead (performance and energy) reduction. In the experiment, which is implemented in a high-level simulator (McPAT and CACTI), FaultHound outperforms PBFS by achieving 75% fault coverage and 3% false-positive rates, with 10% performance overhead and 25% energy overhead.

Another popular technique, the Razor pipeline [Ernst et al. 2003; Blaauw et al. 2008], is a unique error detection and recovery design, which combines both circuit- and architectural-level techniques. Razor mainly aims at timing errors in the processor when performing at ultra-low-power/voltage operating points. Razor can also detect SEUs [Cheng et al. 2016a]. First, Razor includes a special flip-flop design (named Razor flip-flop), which has an additional shadow latch controlled by a delayed clock. The delayed clock frequency is carefully determined considering the target operating voltage to ensure that the shadow latch always has the correct value. The Razor flip-flop can double-sample pipeline stage values. By comparing the two values in the original latch and the shadow latch, a timing error in the pipeline is detected. Upon a timing error, the value in the shadow latch will be used for recovery.

In addition, Razor has two recovery mechanisms for timing errors. The first is a circuit-level technique, that is, global clock gating. Clock gating forces the entire pipeline to stall for one cycle and recompute the result using the value from the shadow latches from Razor flip-flops. Global clock gating is implementation-wise simple, while not being suitable for an aggressively clocked pipeline. The second is an architecture-level technique, based on counterflow pipelining [Sproull et al. 1994]. Razor's counterflow pipelining first nullifies the pipeline stage, where a timing error is detected. Then, the correct value from the corresponding shadow latch is fetched and written back into the pipeline. Hence, the erroneous instruction can continue with correct values. The Razor pipeline incurs 3.1% power overhead, when there is no timing error. In the case of a large error rate (10% error rate), Razor's error recovery overhead is 1%.

With the rise of research in general-purpose GPUs (GPGPUs) for high-performance computing, mitigating soft errors in general-purpose GPUs has become an interesting problem and attracted a large amount of research activities from both industry and academia [Gomez et al. 2014]. At the moment, a substantial number of research efforts on GPGPUs are about evaluating (including field testing) and modeling soft errors in GPGPUs [Tiwari et al. 2015; Martino et al. 2015; El-Sayed and Schroeder 2013; de Oliveira et al. 2016].

For IBM's BlueGene/L scientific computing system, Glosli et al. [2007] proposed an error recovery method based on parity error detection. This recovery technique targets the transient fault in L1 cache. In this method, there are two ways to recover the error: (1) forcing write-through so as to refresh the L1 cache with the correct data from the lower levels of caches and memory, and (2) transferring control to the interrupt handler in case the parity error is not recoverable (in this case checkpoint recovery can be applied).

Inspired by AVF, in order to reduce the cost of mitigating soft errors in general-purpose GPUs, Palframan et al. [2014] proposed a precision-aware mitigation method for an execution unit and register file targeting general-purpose GPU applications, since GPGPU applications typically involve a large number of floating-point computations, which are often tolerant of small errors, in comparison to large magnitude errors. The execution unit is selectively hardened considering floating-point value correctness at the gate level, while the register file only has the most important bits stored and protected by hardened SRAM cells. In this way, only the large-magnitude errors in floating-point values and integer values are covered. This method can reduce 87% SER for multipliers and 31% for adders on average with 5% area overhead.

3.4. Mitigation at the Algorithm and Program Level

Table IV summarizes the algorithm/program-level approaches. Early works utilized traditional fault-tolerance methods, which employ massive redundancy, while recent methods explored ways of improving the cost efficiency of soft error mitigation.

3.4.1. Traditional Software Fault Tolerance. Traditional software-based fault-tolerance methods can be applied to soft error resiliency, and these methods fall into three categories: (1) implementing redundancy in a straightforward manner, for example, N-version programming [Chen and Avižienis 1995]; (2) exploiting algorithms' behavior, such as algorithm-based fault tolerance (ABFT) [Roy-Chowdhury and Banerjee 1996]; and (3) checkpoint recovery (software implemented), such as BLCR [Duell 2003], libFT [Huang 1999], and Libckpt [Plank et al. 1995].

The N-version programming method [Avižienis and Chen 1977; Chen and Avižienis 1995] independently generates a number of programs from the same initial specification. The number is defined as $N (N \geq 2)$. The generated programs are called versions, and these versions must be functionally equivalent. N-version programming diversifies

Table IV. Summary of Algorithm- and Program-Level Mitigation

Approach	Year	Function	Notes
Avižienis & Chen	1977	D&R	N-version programming
Tandem NonStop	1981	R	Checkpoint recovery
Huang & Abraham	1984	D&R	ABFT, matrix multiplication
Jou & Abraham	1988	D	ABFT, QR decomposition
Reddy & Banerjee	1990	D&R	ABFT, matrix multiplication
CATCH	1990	R	Checkpoint recovery, compiler
LibFT	1993	R	Checkpoint recovery, library
Libckpt	1995	R	Checkpoint recovery, library
Roy-Chowdhury & Banerjee	1996	D&R	ABFT, matrix multiplication
Porch	1997	R	Checkpoint recovery, compiler
Mishra & Banerjee	1999	D	ABFT, multigrid
EDDI	2002	D	Code duplication, assembly
CFCSS	2002	D	Signature, assembly
BLCR	2003	R	Checkpoint recovery, Linux kernel
Bronevetsky et al.	2004	R	Checkpoint recovery, shared memory
SWIFT	2005	D	Code duplication, assembly
SWIFT-R/TRUMP	2006	D&R	Code triplication/AN-code
Shoestring	2010	D	Symptoms
de Kruijf et al.	2010	D&R	Idempotent property
Encore	2011	D&R	Idempotent property
Rehman et al.	2011	NA	Masking, code generation
Relyzer	2012	NA	Fault pruning
ULFM	2012	D&R	MPI programming
Sahoo et al.	2013	D	Fault pruning
Khudia et al.	2014	NA	Compiler, reduced duplication

D: Error detection.
R: Error recovery.
NA: Not applicable.

the implementation style of the versions. In N-version programming, a supervisory program exists to coordinate the execution of the N versions. The execution of the versions demands three mechanisms: the comparison vectors, the comparison status indicators, and synchronization. The major limitation of N-version programming is resource cost (approximately N times of the original program without protection). In the context of soft errors, N-version programming can be an overengineered solution, as it not only performs time-redundant fault tolerance but also enhances each redundant instance (i.e., a version) of the program by applying a unique implementation style.

Algorithm-based fault tolerance (ABFT) [Roy-Chowdhury and Banerjee 1996; Huang and Abraham 1984] can be used in matrix computations that are vital to numerous scientific and engineering applications. Various ABFT methods have been proposed for matrix multiplication [Roy-Chowdhury and Banerjee 1996; Huang and Abraham 1984] and QR decomposition [Reddy and Banerjee 1990], Fast Fourier Transform (FFT) [Jou and Abraham 1988], and multigrid algorithms [Mishra and Banerjee 1999, 2003].

ABFT introduces additional calculations to matrix transformations to enable error checking at the end of the transformation. Such a mechanism achieves high error coverage with low performance overhead. ABFT implementation involves one additional checksum row and column for the input matrices. Each element of the checksum column and row is assigned with a value. This value should be the sum of the elements of the original matrix that are in the same row or column. At the end of the matrix operation, if the row and column of the result matrix is not the sum of the elements of the same row or column, an error is signaled. The performance degradation of ABFT is reported to be between 10% and 14%, depending on the matrix dimension, using a case

study of matrix multiplication operation [Prata and Silva 1999]. The major limitations of ABFT are the memory overheads (increasing the matrix size) and the complexity of the algorithm (increasing the code lines).

Software-based checkpoint recovery requires no hardware modification but additional software components, in terms of extra functions or code lines. Moreover, as the level of the design abstraction is higher, software-based checkpoint recovery usually performs at a coarse granularity in the order of milliseconds or even seconds. The data, which can be checkpointed by software-based checkpoint recovery, include (1) programs' variables that are transparent in high-level languages such as C language and (2) architectural states (such as register and memory) that are transparent in assembly instructions.

As a classic fault-tolerance method, software-implemented checkpoint recovery was adopted in commercial computer systems as early as the 1980s by Tandem (nowadays a division of HP) [Bartlett and Spainhower 2004; Bartlett 1981]. In Tandem NonStop computers (a distributed computing system), the support for checkpoint recovery is implemented as a part of the operating system kernel. The checkpoint recovery protocol includes a pair of processes (by adding one redundant process running on a redundant processor node) for one original process. The checkpoint is established when an operation is requested on the original server process. The checkpoint data consists of the context information, which is transferred to the redundant process. When an error in the original process is detected on the original processor node, the operation is recovered and continued in the redundant process.

In academic circles, one example kernel-based checkpoint recovery method is Berkeley Linux Checkpoint/Restart (BLCR) [Duell 2003],[45] a Linux kernel module that allows system-level checkpoints on a variety of Linux systems. BLCR can be used either as a standalone system for recovering applications on a single machine or as a component of a parallel communication library for recovering parallel jobs running on multiple machines.

The other way of implementing checkpoint recovery is integrating the mechanism within a software library. Such methods include libFT [Huang 1999; Wang et al. 1993] and Libckpt [Plank et al. 1995]. There are compiler-based checkpoint recovery methods such as Porch [Ramkumar and Strumpen 1997][46] and CATCH [Li and Fuchs 1990]. Other than reducing the performance and power/energy overheads, the major objectives of these methods are (1) transparency, which favors less modification to user programs in library-based solutions, and (2) portability, which increases with the ease of porting the underlying compiler to various architectures.

For symmetric multiprocessors, Bronevetsky et al. [2004] proposed a method to enhance the shared-memory multithreaded programs with checkpoint recovery capabilities. In this method, the program must be specially instrumented by the programmer so as to have potentialCheckpoint() function calls in the code lines, where taking a checkpoint is safe. This method was tested on three different processor/OS systems. The performance of the enhanced program is decreased by 6% on Linux and 20% on Alpha at the worst, when checkpointing is turned off. Each checkpointing takes 43 seconds, while each recovery takes 24 seconds, in the worst case.

3.4.2. Other Program-Level Mitigations. Given the evolution of processor architecture and development of soft error knowledge, the other algorithm/program-level mitigation methods mostly (1) explore using application information to relax the mitigation cost and (2) exploit new soft error masking mechanisms. A large body of recent

[45]Web resource is available at http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/BLCR.
[46]Web resource can be found at http://supertech.csail.mit.edu/porch/.

software-based mitigation methods are proposed as in situ optimization phases during compiler processing.

A software redundancy approach called EDDI [Oh et al. 2002b] was proposed to check data errors in the early 2000s. EDDI is integrated into the compiler. Error detection functionality is implemented by duplicating the original instructions (with duplicated register and memory locations) and inserting appropriate checking instructions (comparison and branch instructions) before memory-store instructions. Hence, to generate the proper resultant assembly code, EDDI tries to find the storeless basic blocks (SBBs), which must end with a store or branch instruction. Based on each SBB, the duplication is generated considering resource utilization. The sphere of replication (SoR)[47] of EDDI is the entire processor core and the memory. On average, EDDI can achieve over 98% fault coverage for error detection while incurring around 100% performance overhead at the worst case in the experiment.

CFCSS [Oh et al. 2002a], on the other hand, checks control flow errors, based on signatures. CFCSS generates and inserts signatures into the program during compile time. The signatures are part of the instruction code and are unique for each basic block[48] in the program. The check instructions are also inserted during compile time. When the program is executed, the runtime signatures will be generated and compared with the inserted signatures at the check instructions, block by block. A mismatch of the signatures indicates a control flow error. CFCSS increases the error detection coverage for control flow errors by an order of magnitude. The overhead of CFCSS is about 25% to 43% on code size and 16% to 69% on execution time.

SWIFT [Reis et al. 2005] (software-implemented fault tolerance) is another compiler-based error detection solution. SWIFT generates additional lines of code that allow the system to duplicate the computation result of an instruction and compare the two results to decide if there is an error. SWIFT refines EDDI and integrates a software-only signature-based control flow checking scheme. While EDDI's SoR includes the memory subsystem, SWIFT's SoR does not. SWIFT assumes that memory structures can be well protected by hardware-based methods, for example, parity and ECC. Therefore, SWIFT gains a performance benefit from having as little as 50% of the memory usage of EDDI.

On top of compiler-based error detection methods, there are several compiler-based error recovery methods proposed, such as SWIFT-R and TRUMP [Chang et al. 2006; Reis et al. 2007]. Based on SWIFT, SWIFT-R adds majority voting to the original assembly instructions to recover errors, which almost triplicates the amount of code that needs to be executed. TRUMP uses AN-code instead of majority voting for arithmetic instructions to reduce the recovery overhead. AN-code adds more multiplication operations, to implement a special encoding into the original program (more multiplication instructions). The resultant program would have two versions of computations: the original and the AN-encoded ones. By comparing the results from two versions of each instruction, the error would be identified and recovered. Because AN-code does not propagate through many logical operations, the applicability of TRUMP is limited.

In order to effectively reduce the cost of error detection, Shoestring [Feng et al. 2010] explores using symptoms including fatal traps or application aborts to identify both hardware faults and transient errors. Shoestring sacrifices error coverage for reduced overhead. To compensate for the loss in error coverage, Shoestring uses the compiler analysis to identify vulnerable instructions. Those instructions are enhanced with instruction duplication.

[47]The logical domain of redundant execution [Reinhardt and Mukherjee 2000].

[48]Basic block signature has been widely explored targeting varied processors [Schuette and Shen 1987; Ragel and Parameswaran 2006].

Encore [Feng et al. 2011] utilizes the statistically idempotent property[49] of the program to reduce the recovery cost and provide flexibility in the degree of fault tolerance. The objective of Encore is providing software-only error recovery functionality to the low-cost processors that lack hardware support for error recovery. In implementation, Encore includes program analysis, data profiling, and code transformations.

Some other methods utilized the soft error masking effects, that is, self-resilience of the application. Such methods include soft-error-aware code generation [Rehman et al. 2011] and scheduling [Rehman et al. 2012]. The motivation of these methods is that the soft error masking effects can be enhanced by manipulating the instruction code. In comparison to adding error detection and recovery code into the original code, the cost of enhancing soft error masking effects is flexibly controlled by using the soft error vulnerability estimation model, called an instruction vulnerability index (IVI). The manipulation used includes soft-error-aware loop unrolling, data type optimization, and execution sequence organization. Similar to other recent methods, these two works are also implemented as part of a compiler.

The Relyzer method [Hari et al. 2012] aims to estimate the system effects of soft errors given an application at design time. The motivation of Relyzer is that estimating silent data corruptions (SDCs) at the program level is difficult as the possible fault locations (i.e., "fault sites") are too many for fault injection test. To overcome this difficulty, Relyzer carefully prunes the fault sites, instead of random sampling. The pruning process is performed by either predicting the fault outcome or finding the equivalence between the faults. As a result, Relyzer is able to prune around 99.8% of the total faults in the experiment, with 96% accuracy on average.

Similar to Relyzer, Sahoo et al. [2013] proposed to use program invariants to automatically isolate the fault locations offline. The likely program invariants use training inputs, which are close to the failing input. Along with the likely program invariants, filtering techniques are also applied. The resultant fault locations at the program level can be pruned to as small as five to 17 program expressions.

Khudia and Mahlke [2014] proposed a compiler-based technique that aims to reduce the cost of software-based error detection. This technique at first identifies a small subset of critical variables, which can guarantee the correctness of macro-operation of the program. These critical variables are then protected by a traditional software-based technique such as duplication and comparison. The other variables only affect micro-operation of the program and are defined as noncritical variables. For those noncritical variables, a loose check is inserted, so that if the results are close enough to the expected results, no error will be signaled. This technique reduces the silent data corruption from 15% to 1.2%, with 19.5% performance overhead.

To mitigate faults in parallel computing, the User Level Failure Mitigation (ULFM) interface [Bland 2012][50] was proposed and developed by the Fault Tolerance Working Group in a message-passing interface (MPI) forum. Based on MPI version 3, ULFM allows the applications and libraries to recover the state of MPI and tolerate failures by enriching the MPI programming interface with fault-tolerance semantics. In recent case studies [Laguna et al. 2014], ULFM has demonstrated good practicality for lightweight and dynamically balanced applications. However, on the other hand, for bulk-synchronous scientific computing applications, ULFM is difficult to apply.

[49]Idempotent property has also been exploited by the work in de Kruijf et al. [2010] for fault tolerance in multimedia and data-mining applications.

[50]http://svn.mpi-forum.org/trac/mpi-forum-web/ticket/323.

Following ULFM, many promising research activities[51] regarding fault-tolerant MPI have been proposed.

4. CONCLUSION

This article presents a discussion of soft error from a processor design perspective. We have introduced a soft error challenge and also surveyed the existing soft error mitigation methods, including both the classic methods and recent advances, considering different design abstraction levels.

REFERENCES

R. E. Ahmed, R. C. Frazier, and P. N. Marinos. 1990. Cache-aided rollback error recovery (CARER) algorithm for shared-memory multiprocessor systems. In *Proceedings of the 20th International Symposium on Fault-Tolerant Computing, 1990 (FTCS-20'90), Digest of Papers*. 82–88. DOI:http://dx.doi.org/10.1109/FTCS.1990.89338

J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. 2009. Future scaling of processor-memory interfaces. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC'09)*. ACM, New York, NY. Article 42, 12 pages. DOI:http://dx.doi.org/10.1145/1654059.1654102

G. M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference (AFIPS'67 (Spring))*. ACM, New York, NY. 483–485. DOI:http://dx.doi.org/10.1145/1465482.1465560

H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama. 2003b. A 1.3 GHz fifth generation SPARC64 microprocessor. In *Proceedings of the IEEE International Solid-State Circuits Conference, 2003, Digest of Technical Papers. (ISSCC'03)*. Vol. 1. 246–491. DOI:http://dx.doi.org/10.1109/ISSCC.2003.1234286

H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama. 2003a. A 1.3GHz fifth generation SPARC64 microprocessor. In *Proceedings of the Design Automation Conference, 2003.* 702–705. DOI:http://dx.doi.org/10.1109/DAC.2003.1219109

T. M. Austin. 1999. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO 32'99)*. IEEE Computer Society, 196–207.

A. Avižienis and L. Chen. 1977. On the implementation of N-version programming for software fault tolerance during execution. In *Proceedings of the IEEE International Computer Software and Applications Conference*. 149–155.

A. Avizienis. 1971. Arithmetic error codes: Cost and effectiveness studies for application in digital system design. *IEEE Trans. Comput.* 20, 11 (1971), 1322–1331. DOI:http://dx.doi.org/10.1109/T-C.1971.223134

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.* 1, 1 (Jan. 2004), 11–33. DOI:http://dx.doi.org/10.1109/TDSC.2004.2

B. J. Babb. 1998. Error detection and correction circuit. (May 1998). Patent No. 5751744, Filed Jan. 27, 1995, Issued May 12, 1998.

A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker. 2001. Logic circuits with carbon nanotube transistors. *Science* 294, 5545 (2001), 1317–1320. DOI:http://dx.doi.org/10.1126/science.1065824

R. D. Bannon and M. M. Bhansali. 1984. Digital data storage error detecting and correcting system and method. (April 1984). Patent No. 0042966, Filed May 19, 1981, Issued Apr. 18, 1984.

W. B. Barker. 1977. Error-checking scheme. (July 1977). Patent No. 4035766, Filed Aug. 1, 1975, Issued July 12, 1977.

M. J. Barry. 1992. Radiation resistant sram memory cell. (Oct. 1992). Patent No. 5157625, Filed May 22, 1990, Issued Oct. 20, 1992.

J. F. Bartlett. 1981. A nonstop kernel. *SIGOPS Oper. Syst. Rev.* 15, 5 (1981), 22–29. DOI:http://dx.doi.org/10.1145/1067627.806587

[51]Presentation in Salishan Conference on High Speed Computing 2015 (http://salishan.ahsc-nm.org/uploads/4/9/7/0/49704495/laguna.pdf).

W. Bartlett and L. Spainhower. 2004. Commercial fault tolerance: A tale of two systems. *IEEE Trans. Depend. Secure Comput.* 1, 1 (Jan. 2004), 87–96. DOI:http://dx.doi.org/10.1109/TDSC.2004.4

R. Baumann. 2005. Soft errors in advanced computer systems. *IEEE Des. Test* 22, 3 (May 2005), 258–266. DOI:http://dx.doi.org/10.1109/MDT.2005.69

D. Binder, E. C. Smith, and A. B. Holman. 1975. Satellite anomalies from Galactic cosmic rays. *IEEE Trans. Nucl. Sci.* 22, 6 (1975), 2675–2680. DOI:http://dx.doi.org/10.1109/TNS.1975.4328188

A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan. 2005. Computing architectural vulnerability factors for address-based structures. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*. IEEE Computer Society. 532–543. DOI:http://dx.doi.org/10.1109/ISCA.2005.18

D. Blaauw, S. Kalaiselvan, K. Lai, Wei-Hsiang Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. 2008. Razor II: In situ error detection and correction for PVT and SER tolerance. In *Proceedings of the IEEE International Solid-State Circuits Conference, 2008 (ISSCC'08), Digest of Technical Papers.* 400–622. DOI:http://dx.doi.org/10.1109/ISSCC.2008.4523226

T. G. W. Blake and T. W. Houston. 1993. Memory cell with capacitance for single event upset protection. (April 1993). Patent No. 5204990, Filed Sept. 7, 1988, Issued Apr. 20, 1993.

W. Bland. 2012. User Level Failure Mitigation in MPI. In *Proceedings of the Euro-Par 2012: Parallel Processing Workshops: BDMC, CGWS, HeteroPar, HiBB, OMHI, Paraphrase, PROPER, Resilience, UCHPC, VHPC. Revised Selected Papers*. Springer, Berlin. 499–504. DOI:http://dx.doi.org/10.1007/978-3-642-36949-0_57

J. Blome, S. Mahlke, D. Bradley, and K. Flautner. 2005. A microarchitectural analysis of soft error propagation in a production-level embedded microprocessor. In *Proceedings of the 1st Workshop on Architecture Reliability*.

D. M. Blough, F. J. Kurdahi, and S. Y. Ohm. 1999. High-level synthesis of recoverable VLSI microarchitectures. *IEEE Trans. Very Large Scale Integr. Syst.* 7, 4 (Dec. 1999), 401–410. DOI:http://dx.doi.org/10.1109/92.805747

K. K. Bourdelle, S. Chaudhry, and J. Chu. 2002. The effect of triple well implant dose on performance of NMOS transistors. *IEEE Trans. Electron Devices* 49, 3 (March 2002), 521–524. DOI:http://dx.doi.org/10.1109/16.987125

K. A. Bowman, J. W. Tschanz, S. L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De. 2011. A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE J. Solid-State Circuits* 46, 1 (Jan. 2011), 194–208. DOI:http://dx.doi.org/10.1109/JSSC.2010.2089657

G. Bronevetsky, D. Marques, K. Pingali, P. Szwed, and M. Schulz. 2004. Application-level checkpointing for shared memory programs. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI'04)*. ACM, New York, NY. 235–247. DOI:http://dx.doi.org/10.1145/1024393.1024421

D. T. Brown. 1960. Error detecting and correcting binary codes for arithmetic operations. *IRE Trans. Electron. Comput.* EC-9, 3 (Sept. 1960), 333–337. DOI:http://dx.doi.org/10.1109/TEC.1960.5219855

M. Bruel. 1995. Silicon on insulator material technology. *Electron. Lett.* 31, 14 (1995), 1201–1202. DOI:http://dx.doi.org/10.1049/el:19950805

D. Burnett, C. Lage, and A. Bormann. 1993. Soft-error-rate improvement in advanced BiCMOS SRAMs. In *Proceedings of the 31st Annual Proceedings of the International Reliability Physics Symposium, 1993.* 156–160. DOI:http://dx.doi.org/10.1109/RELPHY.1993.283330

J. A. Cairns and J. F. Ziegler. 1985. Coated ceramic substrates for mounting integrated circuits. (July 1985). Patent No. 4528212, Filed July 22, 1982, Issued July 9, 1985.

J. A. Cairns and J. F. Ziegler. 1989. Coated ceramic substrates for mounting integrated circuits and methods of coating such substrates. (March 1989). Patent No. 0099570, Filed July 19, 1983, Issued Mar. 29, 1989.

T. Calin, M. Nicolaidis, and R. Velazco. 1996. Upset hardened memory design for submicron CMOS technology. *IEEE Trans. Nucl. Sci.* 43, 6 (1996), 2874–2878. DOI:http://dx.doi.org/10.1109/23.556880

E. H. Cannon, D. D. Reinhardt, M. S. Gordon, and P. S. Makowenskyj. 2004. SRAM SER in 90, 130 and 180 nm bulk and SOI technologies. In *Proceedings of the 42nd Annual IEEE International Reliability Physics Symposium Proceedings, 2004.* 300–304. DOI:http://dx.doi.org/10.1109/RELPHY.2004.1315341

P. M. Carter and B. R. Wilkins. 1987. Influences on soft error rates in static RAMs. *IEEE J. Solid-State Circuits* 22, 3 (June 1987), 430–436. DOI:http://dx.doi.org/10.1109/JSSC.1987.1052743

J. Chang, G. A. Reis, and D. I. August. 2006. Automatic instruction-level software-only recovery. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. 83–92. DOI:http://dx.doi.org/10.1109/DSN.2006.15

C.-L. Chen. 1989. Double error correction - triple error detection code for a memory. (Aug. 1989). Patent No. 0107038, Filed Sept. 20, 1983, Issued Aug. 23, 1989.

L. Chen and A. Avižienis. 1995. N-version programming: A fault-tolerance approach to rellablllty of software operatlon. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing, 1995, Highlights from 25 Years*. 113–119. DOI:http://dx.doi.org/10.1109/FTCSH.1995.532621

P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. 1994. RAID: High-performance, reliable secondary storage. *ACM Comput. Surv.* 26, 2 (June 1994), 145–185. DOI:http://dx.doi.org/10.1145/176979.176981

E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra. 2016a. CLEAR: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores. *CoRR* abs/1604.03062 (2016). http://arxiv.org/abs/1604.03062

E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra. 2016b. CLEAR: Cross-layer exploration for architecting resilience - combining hardware and software techniques to tolerate soft errors in processor cores. In *Proceedings of the 53rd Annual Design Automation Conference (DAC'16)*. ACM, New York, NY. Article 68, 6 pages. DOI:http://dx.doi.org/10.1145/2897937.2897996

A. L. Crouch, M. D. Pressly, J. C. Circello, and R. Duerden. 1997. Serial scan chain architecture for a data processing system and method of operation. (July 1997). Patent No. 5592493, Filed Sept. 13, 1994, Issued July 1, 1997.

M. de Kruijf, S. Nomura, and K. Sankaralingam. 2010. Relax: An architectural framework for software recovery of hardware faults. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY. 497–508. DOI:http://dx.doi.org/10.1145/1815961.1816026

D. A. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech. 2016. Evaluation and mitigation of radiation-induced soft errors in graphics processing units. *IEEE Trans. Comput.* 65, 3 (March 2016), 791–804. DOI:http://dx.doi.org/10.1109/TC.2015.2444855

T. Dell. 1997. A white paper on the benefits of Chipkill-correct ECC for PC server main memory. In *IBM Microelectronics Division*.

S. E. Diehl, A. Ochoa, Jr., P. V. Dressendorfer, P. Koga, and W. A. Kolasinski. 1982. Error analysis and prevention of cosmic ion-induced soft errors in static CMOS RAMs. *IEEE Trans. Nucl. Sci.* 29 (Dec. 1982), 2032–2039. DOI:http://dx.doi.org/10.1109/TNS.1982.4336491

A. Dixit and A. Wood. 2011. The impact of new technology on soft error rates. In *2011 IEEE International Reliability Physics Symposium (IRPS'11)*. 5B.4.1–5B.4.7. DOI:http://dx.doi.org/10.1109/IRPS.2011.5784522

P. E. Dodd and F. W. Sexton. 1995. Critical charge concepts for CMOS SRAMs. *IEEE Trans. Nucl. Sci.* 42, 6 (Dec. 1995), 1764–1771. DOI:http://dx.doi.org/10.1109/23.488777

J. G. Dooley. 1994. Seu-immune latch for gate array, standard cell, and other asic applications. (May 1994). Patent No. 5311070, Filed June 26, 1992, Issued May 10, 1994.

J. Duell. 2003. *The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart*. Technical Report LBNL-54941. Lawrence Berkeley National Laboratory, Berkeley, CA.

N. El-Sayed and B. Schroeder. 2013. Reading between the lines of failure logs: Understanding how HPC systems fail. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*. 1–12. DOI:http://dx.doi.org/10.1109/DSN.2013.6575356

K. Endo. 2008. Enhancing SRAM cell performance by using independent double-gate FinFET. In *2008 IEEE International Electron Devices Meeting (IEDM'08)*. 1–4. DOI:http://dx.doi.org/10.1109/IEDM.2008.4796833

D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36'03)*. IEEE Computer Society. 7–. http://dl.acm.org/citation.cfm?id=956417.956571

Y. P. Fang and A. S. Oates. 2011. Neutron-induced charge collection simulation of bulk FinFET SRAMs compared with conventional planar SRAMs. *IEEE Trans. Device Mat. Reliabil.* 11, 4 (Dec. 2011), 551–554. DOI:http://dx.doi.org/10.1109/TDMR.2011.2168959

S. Feng, S. Gupta, A. Ansari, and S. Mahlke. 2010. Shoestring: Probabilistic soft error reliability on the cheap. *SIGARCH Comput. Archit. News* 38, 1 (March 2010), 385–396. DOI:http://dx.doi.org/10.1145/1735970.1736063

S. Feng, S. Gupta, A. Ansari, S. A. Mahlke, and D. I. August. 2011. Encore: Low-cost, fine-grained transient fault recovery. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44'11)*. ACM, New York, NY. 398–409. DOI:http://dx.doi.org/10.1145/2155620.2155667

M. J. Fremont. 1987. Method and apparatus for fault recovery within a computing system. (Oct. 1987). Patent No. 4703481, Filed Aug. 16, 1985, Issued Oct. 27, 1987.

X. Fu, T. Li, and J. A. B. Fortes. 2006. Sim-SODA: A unified framework for architectural level software reliability analysis. In *Proceedings of the Workshop on Modeling, Benchmarking and Simulation*.

E. Fujiwara and D. K. Pradhan. 1990. Error-control coding in computers. *Computer* 23, 7 (1990), 63–72. DOI:http://dx.doi.org/10.1109/2.56853

J. Furuta, C. Hamanaka, K. Kobayashi, and H. Onodera. 2010. A 65nm bistable cross-coupled dual modular redundancy flip-flop capable of protecting soft errors on the c-element. In *Proceedings of the 2010 Symposium on VLSI Circuits*. 123–124. DOI:http://dx.doi.org/10.1109/VLSIC.2010.5560329

H. L. Garner. 1966. Error codes for arithmetic operations. *IEEE Trans. Electron. Comput.* EC-15, 5 (Oct. 1966), 763–770. DOI:http://dx.doi.org/10.1109/PGEC.1966.264566

B. Gill, N. Seifert, and V. Zia. 2009. Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node. In *Proceedings of the 2009 IEEE International Reliability Physics Symposium*. 199–205. DOI:http://dx.doi.org/10.1109/IRPS.2009.5173251

J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, J. A. Gunnels, and F. H. Streitz. 2007. Extending stability beyond CPU millennium: A micron-scale atomistic simulation of Kelvin-Helmholtz instability. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07)*. ACM, New York, NY. Article 58, 11 pages. DOI:http://dx.doi.org/10.1145/1362622.1362700

M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. 2003. Transient-fault recovery for chip multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03)*. ACM, New York, NY. 98–109. DOI:http://dx.doi.org/10.1145/859618.859631

L. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. S. Reorda. 2014. GPGPUs: How to combine high computational power with high reliability. In *Proceedings of the 2014 Design, Automation Test in Europe Conference Exhibition (DATE'14)*. 1–9. DOI:http://dx.doi.org/10.7873/DATE.2014.354

W. Gu, Z. Kalbarczyk, Ravishankar, K. Iyer, and Z. Yang. 2003. Characterization of Linux kernel behavior under errors. In *Proceedings of the International Conference on Dependable Systems and Networks, 2003*. 459–468. DOI:http://dx.doi.org/10.1109/DSN.2003.1209956

M. S. Gupta, J. A. Rivers, P. Bose, G. Y. Wei, and D. Brooks. 2009. Tribeca: Design for PVT variations with local recovery and fine-grained adaptation. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 435–446.

R. W. Hamming. 1950. Error detecting and error correcting codes. *Bell Syst. Tech. J.* 26, 2 (1950), 147–160.

J.-J. Han, D.-H. Hwang, B.-K. Kim, and B.-K. Lee. 2001. Semiconductor device having triple-well. (May 2001). Patent No. 6225199, Filed July 7, 1999, Issued May 1, 2001.

S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran. 2012. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII'12)*. ACM, New York, NY. 123–134. DOI:http://dx.doi.org/10.1145/2150976.2150990

K. J. Hass and J. W. Ambles. 1999. Single event transients in deep submicron CMOS. In *42nd Midwest Symposium on Circuits and Systems, 1999*. Vol. 1. 122–125. DOI:http://dx.doi.org/10.1109/MWSCAS.1999.867224

J. D. Hayden. 1994. A quadruple well, quadruple polysilicon BiCMOS process for fast 16 Mb SRAM's. *IEEE Trans. Electron Devices* 41, 12 (1994), 2318–2325. DOI:http://dx.doi.org/10.1109/16.337444

P. Hazucha and C. Svensson. 2000. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. Nucl. Sci.* 47 (Dec. 2000), 2586–2594. DOI:http://dx.doi.org/10.1109/23.903813

J. L. Hennessy and D. A. Patterson. 2006. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers, San Francisco, CA.

R. Ho, K. W. Mai, and M. A. Horowitz. 2001. The future of wires. *Proc. IEEE* 89, 4 (April 2001), 490–504. DOI:http://dx.doi.org/10.1109/5.920580

T. W. Houston. 1990. Memory cell with improved single event upset rate reduction circuitry. (Sept. 1990). Patent No. 4956814, Filed Sept. 30, 1988, Issued Sept. 11, 1990.

A. W. Hsingya, J.-C. Young, and S. K. Ming. 2013. Triple well flash memory cell and fabrication process. (Feb. 2013). Patent No. 0810667, Filed May 30, 1997, Issued Feb. 27, 2013.

K.-H. Huang and J. A. Abraham. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.* 33, 6 (June 1984), 518–528. DOI:http://dx.doi.org/10.1109/TC.1984.1676475

X. Huang. 1999. Sub 50-nm FinFET: PMOS. In *IEDM*. 67–70. DOI:http://dx.doi.org/10.1109/IEDM.1999.823848

D. Hunt and P. Marinos. 1987. A general purpose cache-aided rollback error recovery (CARER) technique. In *Proceedings of the 17th International Symposium on Fault-Tolerant Computing Systems*. 170–175.

E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba. 2010. Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. *IEEE Trans. Electron Devices* 57, 7 (2010), 1527–1538. DOI:http://dx.doi.org/10.1109/TED.2010.2047907

K. Itoh. 1980. A single 5V 64K dynamic RAM. In *ISSCC*. Vol. XXIII. 228–229. DOI:http://dx.doi.org/10.1109/ISSCC.1980.1156076

R. K. Iyer, N. M. Nakka, Z. T. Kalbarczyk, and S. Mitra. 2005. Recent advances and new avenues in hardware-level reliability support. *IEEE Micro* 25, 6 (Nov. 2005), 18–29. DOI:http://dx.doi.org/10.1109/MM.2005.119

K. Johansson, M. Ohlsson, N. Olsson, J. Blomgren, and P. U. Renberg. 1999. Neutron induced single-word multiple-bit upset in SRAM. *IEEE Trans. Nucl. Sci.* 46, 6 (Dec. 1999), 1427–1433. DOI:http://dx.doi.org/10.1109/23.819103

J.-Y. Jou and J. A. Abraham. 1988. Fault-tolerant FFT networks. *IEEE Trans. Comput.* 37, 5 (May 1988), 548–561. DOI:http://dx.doi.org/10.1109/12.4606

A. B. Kahng. 2013. The ITRS design technology and system drivers roadmap: Process and status. In *Proceedings of the 50th Annual Design Automation Conference (DAC'13)*. ACM, New York, NY. Article 34, 6 pages. DOI:http://dx.doi.org/10.1145/2463209.2488776

A. Kar-Roy, M. Racanelli, and J. Zhang. 2006. Deep N wells in triple well structures and method for fabricating same. (May 2006). Patent No. 7052966, Filed Apr. 9, 2003, Issued May 30, 2006.

T. Karnik. 2002. Selective node engineering for chip-level soft error rate improvement [in CMOS]. In *VLSIC*. 204–205. DOI:http://dx.doi.org/10.1109/VLSIC.2002.1015084

L. Hsiao-Heng Kelin, L. Klas, B. Mounaim, R. Prasanthi, I. R. Linscott, U. S. Inan, and M. Subhasish. 2010. LEAP: Layout design through error-aware transistor positioning for soft-error resilient sequential cell design. In *Proceedings of the 2010 IEEE International Reliability Physics Symposium (IRPS'10)*. 203–212. DOI:http://dx.doi.org/10.1109/IRPS.2010.5488829

G. H. Kemmetmueller. 1980. RAM error correction using two dimensional parity checking. (Jan. 1980). Patent No. 4183463, Filed July 31, 1978, Issued Jan. 15, 1980.

D. S. Khudia and S. Mahlke. 2014. Harnessing soft computations for low-budget fault tolerance. In *Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 319–330. DOI:http://dx.doi.org/10.1109/MICRO.2014.33

D. R. Kim. 1977. Longitudinal parity generator for use with a memory. (April 1977). Patent No. 4016409, Filed Mar. 1, 1976, Issued Apr. 5, 1977.

K. S. Kim and L. J. Schultz. 1996. Method and apparatus for multi-frequency, multi-phase scan chain (April 1996). Patent No. 5504756, Filed Sept. 30, 1993, Issued Apr. 2, 1996.

T.-J. King. 2005. FinFETs for nanoscale CMOS digital integrated circuits. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-aided Design (ICCAD'05)*. IEEE Computer Society. 207–210. http://dl.acm.org/citation.cfm?id=1129601.1129631

J. S. Klecka, W. F. Bruckert, and R. L. Jardine. 2002. Error self-checking and recovery using lock-step processor pair architecture. (May 2002). Patent No. 6393582, Filed Dec. 10, 1998, Issued May 21, 2002.

P. M. Kogge, K. T. Truong, D. A. Rickard, and R. L. Schoenike. 1990. Checkpoint retry mechanism. (March 1990). Patent No. 4912707, Filed Aug. 23, 1988, Issued Mar. 27, 1990.

M. Kohara, Y. Mashiko, K. Nakasaki, and M. Nunoshita. 1990. Mechanism of electromigration in ceramic packages induced by chip-coating polyimide. *IEEE Trans. Compon. Hybrids Manufact. Technol.* 13, 4 (1990), 873–878. DOI:http://dx.doi.org/10.1109/33.62532

I. Laguna, D. F. Richards, T. Gamblin, M. Schulz, and B. R. de Supinski. 2014. Evaluating user-level fault tolerance for MPI applications. In *Proceedings of the 21st European MPI Users' Group Meeting (EuroMPI/ASIA'14)*. ACM, New York, NY, Article 57, 6 pages. DOI:http://dx.doi.org/10.1145/2642769.2642775

H. H. K. Lee, K. Lilja, M. Bounasser, I. Linscott, and U. Inan. 2011. Design framework for soft-error-resilient sequential cells. *IEEE Trans. Nucl. Sci.* 58, 6 (Dec. 2011), 3026–3032. DOI:http://dx.doi.org/10.1109/TNS.2011.2168611

N.-C. Lee. 2000. Lead-free soldering and low alpha solders for Wafer level interconnects. In *Proceedings of SMTA International Conference*.

C. C. J. Li and W. K. Fuchs. 1990. CATCH-compiler-assisted techniques for checkpointing. In *Proceedings of the 20th International Symposium Fault-Tolerant Computing, 1990 (FTCS-20'90), Digest of Papers.* 74–81. DOI:http://dx.doi.org/10.1109/FTCS.1990.89337

T. Li, R. Ragel, and S. Parameswaran. 2012. Reli: Hardware/software checkpoint and recovery scheme for embedded processors. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'12)*. 875–880. DOI:http://dx.doi.org/10.1109/DATE.2012.6176621

T. Li, M. Shafique, J. A. Ambrose, S. Rehman, J. Henkel, and S. Parameswaran. 2013a. RASTER: Runtime adaptive spatial/temporal error resiliency for embedded processors. In *Proceedings of the 50th Annual Design Automation Conference (DAC'13)*. ACM, New York, NY. Article 62, 7 pages. DOI:http://dx.doi.org/10.1145/2463209.2488809

T. Li, M. Shafique, S. Rehman, J. A. Ambrose, J. Henkel, and S. Parameswaran. 2013b. DHASER: Dynamic heterogeneous adaptation for soft-error resiliency in ASIP-based multi-core systems. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 646–653. DOI:http://dx.doi.org/10.1109/ICCAD.2013.6691184

X. Li, S. V. Adve, P. Bose, and J. A. Rivers. 2005. SoftArch: An architecture-level tool for modeling and analyzing soft errors. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 496–505. DOI:http://dx.doi.org/10.1109/DSN.2005.88

K. Lilja, M. Bounasser, S. J. Wen, R. Wong, J. Holst, N. Gaspard, S. Jagannathan, D. Loveless, and B. Bhuva. 2013. Single-event performance and layout optimization of flip-flops in a 28-nm bulk technology. *IEEE Trans. Nucl. Sci.* 60, 4 (Aug. 2013), 2782–2788. DOI:http://dx.doi.org/10.1109/TNS.2013.2273437

R. Lim. 2010. *Investigation into Lead-Free Solder in Australian Defence Force Applications*. Technical Report DSTO-TN-0970. DSTO Defence Science and Technology Organization, Air Vehicles Division.

D. Lipetz and E. Schwarz. 2011. Self checking in current floating-point units. In *2011 20th IEEE Symposium on Computer Arithmetic (ARITH'11)*. 73–76. DOI:http://dx.doi.org/10.1109/ARITH.2011.18

M. N. Liu and S. Whitaker. 1992. Low power SEU immune CMOS memory circuits. *IEEE Trans. Nucl. Sci.* 39 (Dec. 1992), 1679–1684. DOI:http://dx.doi.org/10.1109/23.211353

T. D. Loveless, S. Jagannathan, T. Reece, J. Chetia, B. L. Bhuva, M. W. McCurdy, L. W. Massengill, S. J. Wen, R. Wong, and D. Rennie. 2011. Neutron- and proton-induced single event upsets for D- and DICE-flip/flop designs at a 40 nm technology node. *IEEE Trans. Nucl. Sci.* 58, 3 (June 2011), 1008–1014. DOI:http://dx.doi.org/10.1109/TNS.2011.2123918

D. J. C. MacKay. 2002. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY.

J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. 2003. Characterization of multi-bit soft error events in advanced SRAMs. In *IEEE International Electron Devices Meeting, 2003 (IEDM'03), Technical Digest.* 21.4.1–21.4.4. DOI:http://dx.doi.org/10.1109/IEDM.2003.1269335

G. Maki, K. Haas, S. Quan, and J. Murguia. 2003. Conflict free radiation tolerant storage cell. (June 2003). Patent No. 6573773, Filed Feb. 2, 2001, Issued June 3, 2003.

B. E. Mann, P. J. Trasatti, M. D. Carlozzi, J. A. Ywoskus, and E. J. McGrath. 1999. Loosely coupled mass storage computer cluster. (Jan. 1999). Patent No. 5862312, Filed Oct. 24, 1995, Issued Jan. 19, 1999.

J. T. Marino Jr. 1981. DES Parity check system. (April 1981). Patent No. 4262358, Filed June 28, 1979, Issued Apr. 14, 1981.

D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, A. J. Miller, and M. Upton. 2002. Hyper-threading technology architecture and microarchitecture. *Intel Technol. J.* 6, 1 (Feb. 2002), 4–15.

C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer. 2015. Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 25–36. DOI:http://dx.doi.org/10.1109/DSN.2015.50

J. L. Massey and O. N. García. 1972. Error-correcting codes in computer arithmetic. In *Advances in Information Systems Science*. Vol. 4. Springer US, Boston, MA.. 273–326. DOI:http://dx.doi.org/10.1007/978-1-4615-9053-8_5

R. N. Master, S. A. Anand, S. Parthasarathy, and Y. C. Mui. 2007. Lead-free semiconductor package. (May 2007). Patent No. 7215030, Filed June 27, 2005, Issued May 8, 2007.

T. C. May and M. H. Woods. 1979. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. Electron Devices* 26, 1 (1979), 2–9. DOI:http://dx.doi.org/10.1109/T-ED.1979.19370

K. L. McMillan. 1993. Symbolic model checking. In *Symbolic Model Checking*. Springer US. 25–60. DOI:http://dx.doi.org/10.1007/978-1-4615-3190-6_3

A. Meixner, M. E. Bauer, and D. Sorin. 2007. Argus: Low-cost, comprehensive error detection in simple cores. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40'07)*. IEEE Computer Society. 210–222. DOI:http://dx.doi.org/10.1109/MICRO.2007.8

A. Meixner and D. J. Sorin. 2007. Error detection using dynamic dataflow verification. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT'07)*. IEEE Computer Society. 104–118. DOI:http://dx.doi.org/10.1109/PACT.2007.30

G. C. Messenger. 1982. Collection of charge on junction nodes from ion tracks. *IEEE Trans. Nucl. Sci.* 29, 6 (1982), 2024–2031. DOI:http://dx.doi.org/10.1109/TNS.1982.4336490

A. Mishra and P. Banerjee. 1999. An algorithm based error detection scheme for the multigrid algorithm. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers.* 12–19. DOI:http://dx.doi.org/10.1109/FTCS.1999.781029

A. Mishra and P. Banerjee. 2003. An algorithm-based error detection scheme for the multigrid method. *IEEE Trans. Comput.* 52, 9 (2003), 1089–1099. DOI:http://dx.doi.org/10.1109/TC.2003.1228507

N. Miskov-Zivanov and D. Marculescu. 2006. Circuit reliability analysis using symbolic techniques. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.* 25, 12 (Dec. 2006), 2638–2649. DOI:http://dx.doi.org/10.1109/TCAD.2006.882592

N. Miskov-Zivanov and D. Marculescu. 2010. Formal modeling and reasoning for reliability analysis. In *Proceedings of the 47th Design Automation Conference (DAC'10)*. ACM, New York, NY. 531–536. DOI:http://dx.doi.org/10.1145/1837274.1837406

S. Mitra, T. Karnik, N. Seifert, and M. Zhang. 2005a. Logic soft errors in sub-65Nm technologies design and CAD challenges. In *Proceedings of the 42nd Annual Design Automation Conference (DAC'05)*. ACM, New York, NY. 2–4. DOI:http://dx.doi.org/10.1145/1065579.1065585

S. Mitra and E. J. McCluskey. 2000. Which concurrent error detection scheme to choose? In *Proceedings of the 2012 IEEE International Test Conference.* 985. DOI:http://dx.doi.org/10.1109/TEST.2000.894311

S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. 2005b. Robust system design with built-in soft-error resilience. *Computer* 38, 2 (2005), 43–52. DOI:http://dx.doi.org/10.1109/MC.2005.70

S. Mukherjee. 2008. *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers, San Francisco, CA.

S. S. Mukherjee, J. Emer, and S. K. Reinhardt. 2005. The soft error problem: An architectural perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. 243–247. DOI:http://dx.doi.org/10.1109/HPCA.2005.37

S. S. Mukherjee, J. S. Emer, S. K. Reinhardt, and C. T. Weaver. 2008. Implementing check instructions in each thread within a redundant multithreading environments. (April 2008). Patent No. 7353365, Filed Sept. 29, 2004, Issued Apr. 1, 2008.

S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. 2003a. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36'03)*. IEEE Computer Society. 29–. http://dl.acm.org/citation.cfm?id=956417.956570

S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. 2003b. Measuring architectural vulnerability factors. *IEEE Micro* 23, 6 (Nov. 2003), 70–75. DOI:http://dx.doi.org/10.1109/MM.2003.1261389

D. E. Muller and W. S. Bartky. 1959. A theory of asynchronous circuits. In *Proceedings of International Symposium on the Theory of Switching*. Harvard University Press.

P. C. Murley and G. R. Srinivasan. 1996. Soft-error Monte Carlo modeling program, SEMM. *IBM J. Res. Dev.* 40, 1 (Jan. 1996), 109–118. DOI:http://dx.doi.org/10.1147/rd.401.0109

A. A. Nair, S. Eyerman, J. Chen, L. K. John, and L. Eeckhout. 2015. Mechanistic modeling of architectural vulnerability factor. *ACM Trans. Comput. Syst.* 32, 4, Article 11 (Jan. 2015), 32 pages. DOI:http://dx.doi.org/10.1145/2669364

A. A. Nair, S. Eyerman, L. Eeckhout, and L. Kurian John. 2012. A first-order mechanistic model for architectural vulnerability factor. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA'12)*. IEEE Computer Society. 273–284. http://dl.acm.org/citation.cfm?id=2337159.2337191

N. Nakka, Z. Kalbarczyk, R. K. Iyer, and J. Xu. 2004. An architectural framework for providing reliability and security support. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*. 585–594. DOI:http://dx.doi.org/10.1109/DSN.2004.1311929

M. Nicolaidis, R. O. Duarte, S. Manich, and J. Figueras. 1997. Fault-secure parity prediction arithmetic operators. *IEEE Des. Test Comput.* 14, 2 (1997), 60–71. DOI:http://dx.doi.org/10.1109/54.587743

Nitin, I. Pomeranz, and T. N. Vijaykumar. 2015. FaultHound: Value-locality-based soft-fault tolerance. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. ACM, New York, NY. 668–681. DOI:http://dx.doi.org/10.1145/2749469.2750372

E. Normand, J. L. Wert, H. Quinn, T. D. Fairbanks, S. Michalak, G. Grider, P. Iwanchuk, J. Morrison, S. Wender, and S. Johnson. 2010. First record of single-event upset on ground, cray-1 computer at Los

Alamos in 1976. *IEEE Trans. Nucl. Sci.* 57, 6 (2010), 3114–3120. DOI:http://dx.doi.org/10.1109/TNS.2010.2083687

Nvidia. 2009. *Fermi Compute Architecture Whitepaper*.

N. Oh, P. P. Shirvani, and E. J. McCluskey. 2002a. Control-flow checking by software signatures. *IEEE Trans. Reliabil.* 51, 1 (March 2002), 111–122. DOI:http://dx.doi.org/10.1109/24.994926

N. Oh, P. P. Shirvani, and E. J. McCluskey. 2002b. Error detection by duplicated instructions in super-scalar processors. *IEEE Trans. Reliabil.* 51, 1 (March 2002), 63–75. DOI:http://dx.doi.org/10.1109/24.994913

P. Oldiges, R. Dennard, D. Heidel, T. Ning, K. Rodbell, H. Tang, M. Gordon, and L. Wissel. 2009. Technologies to further reduce soft error susceptibility in SOI. In *Proceedings of the 2009 IEEE International Electron Devices Meeting (IEDM'09)*. 1–4. DOI:http://dx.doi.org/10.1109/IEDM.2009.5424338

K. Osada, Y. Saitoh, E. Ibe, and K. Ishibashi. 2003. 16.7-fA/cell tunnel-leakage-suppressed 16-Mb SRAM for handling cosmic-ray-induced multierrors. *IEEE J. Solid-State Circuits* 38, 11 (Nov. 2003), 1952–1957. DOI:http://dx.doi.org/10.1109/JSSC.2003.818138

J. M. Palau, G. Hubert, K. Coulie, B. Sagnes, M. C. Calvet, and S. Fourtine. 2001. Device simulation study of the SEU sensitivity of SRAMs to internal ion tracks generated by nuclear reactions. *IEEE Trans. Nucl. Sci.* 48, 2 (April 2001), 225–231. DOI:http://dx.doi.org/10.1109/23.915368

D. J. Palframan, N. S. Kim, and M. H. Lipasti. 2014. Precision-aware soft error protection for GPUs. In *Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. 49–59. DOI:http://dx.doi.org/10.1109/HPCA.2014.6835966

J. H. Patel and L. Y. Fung. 1982. Concurrent error detection in ALU's by recomputing with shifted operands. *IEEE Trans. Comput.* 31, 7 (July 1982), 589–595. DOI:http://dx.doi.org/10.1109/TC.1982.1676055

N. Patil, Jie Deng, S. Mitra, and H.-S. P. Wong. 2009. Circuit-level performance benchmarking and scalability analysis of carbon nanotube transistor circuits. *IEEE Trans. Nanotechnol.* 8, 1 (2009), 37–45. DOI:http://dx.doi.org/10.1109/TNANO.2008.2006903

D. A. Patterson, G. Gibson, and R. H. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD'88)*. ACM, New York, NY. 109–116. DOI:http://dx.doi.org/10.1145/50202.50214

W. W. Peterson and E. J. Weldon. 1972. *Error-Correcting Codes*. MIT Press, Cambridge, MA.

J. S. Plank, M. Beck, G. Kingsley, and K. Li. 1995. Libckpt: Transparent checkpointing under unix. In *Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95)*. USENIX Association, Berkeley, CA. 18–18. http://dl.acm.org/citation.cfm?id=1267411.1267429

M. Poolakkaparambil, J. Mathew, A. M. Jabir, and S. P. Mohanty. 2012. An investigation of concurrent error detection over binary Galois fields in CNTFET and QCA technologies. In *ISVLSI*. 141–146. DOI:http://dx.doi.org/10.1109/ISVLSI.2012.57

E. Pop, S. Dutta, D. Estrada, and Albert Liao. 2009. Avalanche, joule breakdown and hysteresis in carbon nanotube transistors. In *Proceedings of the 2009 IEEE International Reliability Physics Symposium*. 405–408. DOI:http://dx.doi.org/10.1109/IRPS.2009.5173287

D. K. Pradhan (Ed.). 1996. *Fault-Tolerant Computer System Design*. Prentice-Hall, Upper Saddle River, NJ.

P. Prata and J. G. Silva. 1999. Algorithm based fault tolerance versus result-checking for matrix computations. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers.* 4–11. DOI:http://dx.doi.org/10.1109/FTCS.1999.781028

M. Prvulovic, Z. Zhang, and J. Torrellas. 2002. ReVive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. IEEE Computer Society. 111–122. http://dl.acm.org/citation.cfm?id=545215.545228

S. Raasch, A. Biswas, J. Stephan, P. Racunas, and J. Emer. 2015. A fast and accurate analytical technique to compute the AVF of sequential bits in a processor. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48'15)*. ACM, New York, NY. 738–749. DOI:http://dx.doi.org/10.1145/2830772.2830829

J. M. Rabaey. 1996. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Upper Saddle River, NJ.

P. Racunas, K. Constantinides, S. Manne, and S. S. Mukherjee. 2007. Perturbation-based fault screening. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*. 169–180. DOI:http://dx.doi.org/10.1109/HPCA.2007.346195

R. G. Ragel and S. Parameswaran. 2006. IMPRES: Integrated monitoring for processor reliability and security. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06)*. ACM, New York, NY. 502–505. DOI:http://dx.doi.org/10.1145/1146909.1147041

B. Ramkumar and V. Strumpen. 1997. Portable checkpointing for heterogeneous architectures. In *Proceedings of the 27th Annual International Symposium on Fault-Tolerant Computing, 1997 (FTCS-27'97). Digest of Papers*. 58–67. DOI:http://dx.doi.org/10.1109/FTCS.1997.614078

J. Ray, J. C. Hoe, and B. Falsafi. 2001. Dual use of superscalar datapath for transient-fault detection and recovery. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO 34'01)*. IEEE Computer Society. 214–224. http://dl.acm.org/citation.cfm?id=563998.564027

A. L. N. Reddy and P. Banerjee. 1990. Algorithm-based fault detection for signal processing applications. *IEEE Trans. Comput.* 39, 10 (1990), 1304–1308. DOI:http://dx.doi.org/10.1109/12.59860

V. K. Reddy, A. S. Al-Zawawi, and E. Rotenberg. 2006. Assertion-based microarchitecture design for improved fault tolerance. In *Proceedings of the 2006 International Conference on Computer Design*. 362–369. DOI:http://dx.doi.org/10.1109/ICCD.2006.4380842

S. Rehman, M. Shafique, and J. Henkel. 2012. Instruction scheduling for reliability-aware compilation. In *Proceedings of the 49th Annual Design Automation Conference (DAC'12)*. ACM, New York, NY. 1292–1300. DOI:http://dx.doi.org/10.1145/2228360.2228601

S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. 2011. Reliable software for unreliable hardware: Embedded code generation aiming at reliability. In *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'11)*. ACM, New York, NY. 237–246. DOI:http://dx.doi.org/10.1145/2039370.2039408

S. K. Reinhardt and S. S. Mukherjee. 2000. Transient fault detection via simultaneous multithreading. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00)*. ACM, New York, NY. 25–36. DOI:http://dx.doi.org/10.1145/339647.339652

S. K. Reinhardt, S. S. Mukherjee, J. S. Emer, and C. T. Weaver. 2008. Managing external memory updates for fault detection in redundant multithreading systems using speculative memory support. (Oct. 2008). Patent No. 7444497, Filed Dec. 30, 2003, Issued Oct. 28, 2008.

G. A. Reis, J. Chang, and D. I. August. 2007. Automatic instruction-level software-only recovery. *IEEE Micro* 27, 1 (Jan. 2007), 36–47. DOI:http://dx.doi.org/10.1109/MM.2007.4

G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. 2005. SWIFT: Software implemented fault tolerance. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO'05)*. IEEE Computer Society. 243–254. DOI:http://dx.doi.org/10.1109/CGO.2005.34

M. W. Roberson. 1998. Soft error rates in solder bumped packaging. In *Proceedings of the 4th International Symposium on Advanced Packaging Materials, 1998.* 111–116. DOI:http://dx.doi.org/10.1109/ISAPM.1998.664444

L. R. Rockett. 1988. An SEU-hardened CMOS data latch design. *IEEE Trans. Nucl. Sci.* 35, 6 (Dec. 1988), 1682–1687. DOI:http://dx.doi.org/10.1109/23.25522

L. R. Rockett. 1992. Simulated SEU hardened scaled CMOS SRAM cell design using gated resistors. *IEEE Trans. Nucl. Sci.* 39, 5 (Oct. 1992), 1532–1541. DOI:http://dx.doi.org/10.1109/23.173239

K. P. Rodbell, D. F. Heidel, J. A. Pellish, P. W. Marshall, H. H. K. Tang, C. E. Murray, K. A. LaBel, M. S. Gordon, K. G. Stawiasz, J. R. Schwank, M. D. Berg, H. S. Kim, M. R. Friendlich, A. M. Phan, and C. M. Seidleck. 2011. 32 and 45 nm radiation-hardened-by-design (RHBD) SOI latches. *IEEE Trans. Nucl. Sci.* 58, 6 (Dec. 2011), 2702–2710. DOI:http://dx.doi.org/10.1109/TNS.2011.2171715

E. Rotenberg. 1999. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers*. 84–91. DOI:http://dx.doi.org/10.1109/FTCS.1999.781037

A. Roy-Chowdhury and P. Banerjee. 1996. Algorithm-based fault location and recovery for matrix computations on multiprocessor systems. *IEEE Trans. Comput.* 45, 11 (1996), 1239–1247. DOI:http://dx.doi.org/10.1109/12.544480

S. K. Sahoo, J. Criswell, C. Geigle, and V. Adve. 2013. Using likely invariants for automated software fault localization. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13)*. ACM, New York, NY. 139–152. DOI:http://dx.doi.org/10.1145/2451116.2451131

P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones. 2008. Soft-error resilience of the IBM POWER6 processor. *IBM J. Res. Dev.* 52, 3 (May 2008), 275–284. DOI:http://dx.doi.org/10.1147/rd.523.0275

M. A. Schuette and J. P. Shen. 1987. Processor control flow monitoring using signatured instruction streams. *IEEE Trans. Comput.* C-36, 3 (March 1987), 264–276. DOI:http://dx.doi.org/10.1109/TC.1987.1676899

N. Seifert, V. Ambrose, B. Gill, Q. Shi, R. Allmon, C. Recchia, S. Mukherjee, N. Nassif, J. Krause, J. Pickholtz, and A. Balasubramanian. 2010. On the radiation-induced soft error performance of hardened sequential elements in advanced bulk CMOS technologies. In *Proceedings of the 2010 IEEE International Reliability Physics Symposium (IRPS'10)*. 188–197. DOI:http://dx.doi.org/10.1109/IRPS.2010.5488831

N. Seifert, S. Jahinuzzaman, J. Velamala, R. Ascazubi, N. Patel, B. Gill, J. Basile, and J. Hicks. 2015. Soft error rate improvements in 14-nm technology featuring second-generation 3d tri-gate transistors. *IEEE Trans. Nucl. Sci.* 62, 6 (Dec. 2015), 2570–2577. DOI:http://dx.doi.org/10.1109/TNS.2015.2495130

N. Seifert, P. Shipley, M. D. Pant, V. Ambrose, and B. Gill. 2005. Radiation-induced clock jitter and race. In *Proceedings of the 43rd Annual IEEE International Reliability Physics Symposium, 2005.* 215–222. DOI:http://dx.doi.org/10.1109/RELPHY.2005.1493087

N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz. 2006. Radiation-induced soft error rates of advanced CMOS bulk devices. In *Proceedings of the 2006 IEEE International Reliability Physics Symposium Proceedings*. 217–225. DOI:http://dx.doi.org/10.1109/RELPHY.2006.251220

N. Seifert and N. Tam. 2004. Timing vulnerability factors of sequentials. *IEEE Trans. Device Materials Reliabil.* 4, 3 (2004), 516–522. DOI:http://dx.doi.org/10.1109/TDMR.2004.831993

F. F. Sellers, M. Xiao, and L. W. Bearnson. 1968. *Error Detecting Logic for Digital Computers*. McGraw-Hill, New York, NY.

S. A. Seshia, W. Li, and S. Mitra. 2007. Verification-guided soft error resilience. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'07)*. EDA Consortium, San Jose, CA. 1442–1447. http://dl.acm.org/citation.cfm?id=1266366.1266681

C. E. Shannon. 2001. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.* 5, 1 (Jan. 2001), 3–55. DOI:http://dx.doi.org/10.1145/584091.584093

B. Shim, S. R. Sridhara, and N. R. Shanbhag. 2004. Reliable low-power digital signal processing via reduced precision redundancy. *IEEE Trans. Very Large Scale Integration (VLSI) Syst.* 12, 5 (May 2004), 497–510. DOI:http://dx.doi.org/10.1109/TVLSI.2004.826201

P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. 2002. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks, 2002 (DSN'02)*. 389–398. DOI:http://dx.doi.org/10.1109/DSN.2002.1028924

R. L. Shuler, C. Kouba, and P. M. O'Neill. 2005. SEU performance of TAG based flip-flops. *IEEE Trans. Nucl. Sci.* 52 (Dec. 2005), 2550–2553. DOI:http://dx.doi.org/10.1109/TNS.2005.860712

R. L. Shuler Jr. 2002a. Method and apparatus for reducing the vulnerability of latches to single event upsets. (April 2002). Patent No. 6377097, Filed Mar. 13, 2000, Issued Apr. 23, 2002.

R. L. Shuler Jr. 2002b. Method and apparatus for reducing the vulnerability of latches to single event upsets. (Dec. 2002). Patent No. 6492857, Filed Apr. 20, 2001, Issued Dec. 10, 2002.

D. P. Siewiorek and R. S. Swarz. 1998. *Reliable Computer Systems (3rd ed.): Design and Evaluation*. A. K. Peters, Natick, MA.

T. J. Slegel, R. M. Averill III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. 1999. IBM's S/390 G5 microprocessor design. *IEEE Micro* 19, 2 (1999), 12–23. DOI:http://dx.doi.org/10.1109/40.755464

J. Snyder and J. Larson. 2007. CMOS device with zero soft error rate. (April 2007). Patent No. 20070080406, Filed Oct. 12, 2006, Issued Apr. 12, 2007.

J. P. Snyder and J. M. Larson. 2011. Method of manufacturing a cmos device with zero soft error rate. (Feb. 2011). Patent No. 20110034016, Filed Oct. 20, 2010, Issued Feb. 10, 2011.

G. S. Sohi, M. Franklin, and K. K. Saluja. 1989. A study of time-redundant fault tolerance techniques for high-performance pipelined computers. In *Proceedings of the 19th International Symposium on Fault-Tolerant Computing, 1989 (FTCS-19'89). Digest of Papers.* 436–443. DOI:http://dx.doi.org/10.1109/FTCS.1989.105616

D. J. Sorin. 2009. Fault tolerant computer architecture. *Synthesis Lectures on Computer Architecture* 4, 1 (2009), 1–104. DOI:http://dx.doi.org/10.2200/S00192ED1V01Y200904CAC005

D. J. Sorin, M. M. K. Martin, M. D. Hill, and D. A. Wood. 2002. SafetyNet: Improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. IEEE Computer Society, Washington, DC. 123–134. http://dl.acm.org/citation.cfm?id=545215.545229

R. F. Sproull, I. E. Sutherland, and C. E. Molnar. 1994. The counterflow pipeline processor architecture. *IEEE Des. Test* 11, 3 (July 1994), 48–59. DOI:http://dx.doi.org/10.1109/MDT.1994.303847

V. Sridharan and D. R. Kaeli. 2009. Eliminating microarchitectural dependency from architectural vulnerability. In *Proceedings of the 2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 117–128. DOI:http://dx.doi.org/10.1109/HPCA.2009.4798243

K. Sundaramoorthy, Z. Purser, and E. Rotenburg. 2000. Slipstream processors: Improving both performance and fault tolerance. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX'00)*. ACM, New York, NY. 257–268. DOI:http://dx.doi.org/10.1145/378993.379247

A. Taber and E. Normand. 1993. Single event upset in avionics. *IEEE Trans. Nucl. Sci.* 40 (April 1993), 120–126. DOI:http://dx.doi.org/10.1109/23.212327

D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 331–342. DOI:http://dx.doi.org/10.1109/HPCA.2015.7056044

S. J. Trans, A. R. M. Verschueren, and C. Dekker. 1998. Room-temperature transistor based on a single carbon nanotube. *Nature* 393, 5545 (1998), 49–52. Issue 6680. DOI:http://dx.doi.org/10.1038/29954

R. R. Tummala, E. J. Rymaszewski, and A. G. Klopfenstein. 1997. *Microelectronics Packaging Handbook: Semiconductor Packaging*. Springer.

T. Uemura, T. Kato, H. Matsuyama, and M. Hashimoto. 2015. Soft error immune latch design for 20 nm bulk CMOS. In *Proceedings of the 2015 IEEE International Reliability Physics Symposium*. SE.4.1–SE.4.6. DOI:http://dx.doi.org/10.1109/IRPS.2015.7112825

T. N. Vijaykumar, I. Pomeranz, and K. Cheng. 2002. Transient-fault recovery using simultaneous multithreading. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. IEEE Computer Society. 87–98.

J. T. Wallmark and S. M. Marcus. 1962. Minimum size and maximum packing density of nonredundant semiconductor devices. *Proc. IRE* 50, 3 (March 1962), 286–298. DOI:http://dx.doi.org/10.1109/JRPROC.1962.288321

F. Wang, Y. Xie, K. Bernstein, and Y. Luo. 2006. Dependability analysis of nano-scale FinFET circuits. In *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*. DOI:http://dx.doi.org/10.1109/ISVLSI.2006.35

N. J. Wang. 2007. *Cost Effective Soft Error Mitigation In Microprocessors*. Ph.D. Dissertation. University of Illinois at Urbana-Champaign, Champaign, IL. Advisor(s) Sanjay J. Patel. AAI3290421.

N. J. Wang and S. J. Patel. 2006. ReStore: Symptom-based soft error detection in microprocessors. *IEEE Trans. Dependable Secur. Comput.* 3, 3 (July 2006), 188–201. DOI:http://dx.doi.org/10.1109/TDSC.2006.40

Y. M. Wang, Y. Huang, and W. K. Fuchs. 1993. Progressive retry for software error recovery in distributed systems. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing, 1993 (FTCS-23'93). Digest of Papers.* 138–144. DOI:http://dx.doi.org/10.1109/FTCS.1993.627317

C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt. 2004. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*. IEEE Computer Society. 264–. http://dl.acm.org/citation.cfm?id=998680.1006723

H. T. Weaver. 1987. An SEU tolerant memory cell derived from fundamental studies of SEU mechanisms in SRAM. *IEEE Trans. Nucl. Sci.* 34, 6 (12 1987), 1281–1286. DOI:http://dx.doi.org/10.1109/TNS.1987.4337466

S. Whitaker, J. Canaris, and K. Liu. 1991. SEU hardened memory cells for a CCSDS Reed-Solomon encoder. *IEEE Trans. Nucl. Sci.* 38, 6 (12 1991), 1471–1477. DOI:http://dx.doi.org/10.1109/23.124134

S. R. Whitaker. 1992. Single event upset hardening CMOS memory circuit. (May 1992). Patent No. 5111429, Filed Nov. 6, 1990, Issued May 5, 1992.

M. Wilkening, V. Sridharan, S. Li, F. Previlon, S. Gurumurthi, and D. R. Kaeli. 2014. Calculating architectural vulnerability factors for spatial multi-bit transient faults. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 293–305. DOI:http://dx.doi.org/10.1109/MICRO.2014.15

M. J. Y. Williams and J. B. Angell. 1973. Enhancing testability of large-scale integrated circuits via test points and additional logic. *IEEE Trans. Comput.* 22, 1 (Jan. 1973), 46–60. DOI:http://dx.doi.org/10.1109/T-C.1973.223600

H. S. P. Wong, J. Appenzeller, V. Derycke, R. Martel, S. Wind, and P. Avouris. 2003. Carbon nanotube field effect transistors - fabrication, device physics, and circuit implications. In *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers (ISSCC'03)*. 370–500 vol. 1. DOI:http://dx.doi.org/10.1109/ISSCC.2003.1234339

A. Wood. 1999. Data integrity concepts, features, and technology. (4 1999). White paper, Tandem Division, Compaq Computer Corporation.

J. Xu. 2003. *Software and Hardware Techniques for Masking Security Vulnerabilities*. Ph.D. Dissertation. Champaign, IL, USA. Advisor(s) Iyer, Ravishankar K. AAI3111659.

J. Yang and H.-W. Huang. 2000. Triple well structure. (Aug. 2000). Patent No. 6111283, Filed Feb. 1, 1999, Issued Aug. 29, 2000.

D. H. Yoon and M. Erez. 2010. Virtualized and flexible ECC for main memory. In *Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS XV'10)*. ACM, New York, NY. 397–408. DOI:http://dx.doi.org/10.1145/1736020.1736064

M. Zhang. 2006. Sequential element design with built-in soft error resilience. *IEEE TVLSI* 14, 12 (12 2006), 1368–1378. DOI:http://dx.doi.org/10.1109/TVLSI.2006.887832

J. F. Ziegler. 1996. Terrestrial cosmic rays. *IBM J. Res. Devel.* 40, 1 (Jan. 1996), 19–39. DOI:http://dx.doi.org/10.1147/rd.401.0019

J. F. Ziegler and W. A. Lanford. 1979. Effect of cosmic rays on computer memories. *Science* 206, 4420 (1979), 776–788. DOI:http://dx.doi.org/10.1126/science.206.4420.776

Received December 2015; revised August 2016; accepted September 2016