Soft-Error Tolerant Cache Architectures

(耐ソフト・エラーのキャッシュ・アーキテクチャ)

Supervisor: Professor Shuichi Sakai

Luong Dinh Hung

Department of Information Science and Technology,
The University of Tokyo

December 2006

Abstract

The problem of soft errors caused by radiation events are expected to get worse with technology scaling. This thesis focuses on mitigation of soft errors to improve the reliability of memory caches. We survey existing mitigation techniques and discuss their issues. We then propose 1) a technique that can mitigate soft errors in caches with lower costs than the widely-used Error Correcting Code (ECC), 2) a technique to mitigate soft errors in Content Addressable Memories, and 3) a cost-effective cache architecture achieving both variation-induced defect and soft-error tolerance.

ECC is widely used to detect and correct soft errors in memory caches. Maintaining ECC on a per-word basis, which is preferred for caches with word-based access, is expensive. Chapter 3 proposes **Zigzag-HVP**, a cost-effective technique to detect and correct soft errors for such caches. Zigzag-HVP utilizes horizontal-vertical parity (HVP). Basic HVP can detect and correct a single bit error (SBE), but not a multi-bit error (MBE). By dividing the data array into multiple HVP domains and interleaving different domains, a spatial MBE can be converted to multiple SBEs, each of which can be detected and corrected by the corresponding parity domain. Vertical parity update and error recovery in Zigzag-HVP can be performed efficiently by modifications to the cache data paths, write-buffer, and Built-In Self Test. Evaluation results indicate that the area and power overheads of Zigzag-HVP caches are lower than those of ECC-based ones.

Chapter 4 proposes **STCAM**, a soft-error tolerant Content-Addressable Memory (CAM). Soft-error mitigation in a CAM is difficult due to the unavailability of data outside the cell array in a CAM access. Since CAMs are used in several components of a processor, making those CAMs being resilient against soft errors is required to attain high processor's reliability. STCAM can successfully detect and correct false hits and false misses caused by soft errors in a CAM. This is achieved through subdividing a CAM and

providing backup checking for cases the input tag is partially matched in the CAM. An original encoding scheme is proposed to reduce the frequency of backup checking. Modifications to support STCAM do not increase access latency. Performance degradation incurred by backup checking is very low.

Chapter 5 presents **SEVA**, a soft-error- and variation-aware cache architecture. As memory devices are scaled down, the number of variation-induced defective cells increases rapidly. Combination of ECC, particularly Single-Error Correction Double-Error Detection (SECDED), with a redundancy technique can effectively tolerate a high number of defects. While SECDED can repair a defective cell in a hardware block, the block becomes vulnerable to soft errors. SEVA exploits SECDED to tolerate variation-induced defects while preserving high resilience against soft errors. Information about the defectiveness and data dirtiness is maintained for each SECDED block. SEVA allows only the clean data to be stored in the defective blocks. An error occurring in a defective block can be detected and the correct data can be obtained from the lower level of the memory hierarchy. SEVA improves both yield and reliability with low overheads.

Having memory caches to be tolerable from soft errors is essential for attaining high processor's reliability. Incurring low area and power overheads, **Zigzag-HVP** allows support for soft-error tolerance to be more affordable and therefore pervasive. **STCAM** increases in the coverage of soft error protection in a processor. Finally, **SEVA** shows that soft-error tolerance for reliability and defect tolerance for yield can be achievable with reasonable costs, paving the way for successful SRAM designs in future process technology.

Acknowledgment

First of all, I would like to express my gratitude to my advisor, Professor Shuichi Sakai, for his continuous guidance, support, and encourage given to me throughout the six years I has spent in his laboratory. I fully appreciate his earnest effort to provide the students with the best-equipped research environments and the highest opportunity to pursuit their own research.

I address thanks to Associate Professor Masahiro Goshima for his invaluable technical comments and suggestions on my research. He also gave me helpful advice on how to improve presentation skill.

I am also grateful for the referee members of this thesis: Professor Masaru Kitsuregawa, Professor Tohru Asami, Professor Hitoshi Aida, and Associate Professor Kenjiro Taura for their invaluable comments and suggestions to the thesis.

I'm also deeply gratitude to Professor Hidehiko Tanaka, now at Institute of Information Security, for supervising me during my master course. I admire his great leadership at the laboratory previously as well as at the CREST project that my research has been involved.

I thank the rest of laboratory members for all the support and invaluable discussions, both technical and non-technical ones. I thank Dr. Hidetsugu Irie for leading and taking care of the research project. I am especially indebted to those who spent a lot of their time for maintaining the network and computers in the laboratory, particularly Hiroyuki Kurita, Kazuto Shimizu, Ryota Shioya, Hironori Ichibayashi, and Kenichi Watanabe. Special thanks go to Mrs. Harumi Yagihara, Ms. Miwa Tsukimura, and Ms. Anzu Uchida for their dedications to make administration affairs run smoothly. I also thank former members, especially Dr. Niko Demus Barli and Ms. Chitaka Iwama, for assisting me a lot when I started my research at the laboratory.

I also express my thanks to Kansai Paint Scholarship and Tokyo Marine Kagami Memorial Foundation. Without their generous financial support, I

would have not been able to fully dedicate myself to the research.

Luong Dinh Hung
The University of Tokyo, Japan
December 2006

Contents

1 Introduction **15**
- 1.1 Background . . . 15
  - 1.1.1 Soft Error Problem . . . 15
  - 1.1.2 Soft-Error Tolerance in Memory Caches . . . 20
- 1.2 Research Contributions . . . 22
- 1.3 Thesis Organization . . . 23

2 Existing Work on Soft-Error Tolerance in Memory Caches **25**
- 2.1 Process- and Technology-Level Hardening Techniques . . . 25
- 2.2 Circuit- and System-Level Hardening Techniques . . . 27
- 2.3 Architectural Level Hardening Techniques . . . 29
- 2.4 Room for Improvements . . . 31
  - 2.4.1 Overhead Reduction . . . 31
  - 2.4.2 Soft Error Mitigation in Content-Addressable Memory 32
  - 2.4.3 Pursuit of Soft-Error- and Defect Tolerance . . . 32

3 Zigzag-HVP: Soft-Error Mitigation in Caches with Word-based Access **33**
- 3.1 Introduction . . . 33
- 3.2 Related Work . . . 35
- 3.3 Horizontal-Vertical Parity and Its Limitations . . . 36
  - 3.3.1 HVP Concept . . . 36
  - 3.3.2 Limitations with Basic HVP . . . 37
  - 3.3.3 Multi-Bit Errors and Characteristics . . . 38
- 3.4 Zigzag-HVP . . . 39
  - 3.4.1 Bit Interleaving Schemes . . . 39
  - 3.4.2 Parity Update Mechanism . . . 42
  - 3.4.3 Error Recovery . . . 43

3.5 Applications of Zigzag-HVP 44
3.5.1 L1 Write-back Caches 44
3.5.2 L2 Caches with Write-through L1 Caches 44
3.6 Evaluation 49
3.6.1 Evaluation Methodology 49
3.6.2 Physical Configurations of L2 caches 51
3.6.3 Area Overhead 51
3.6.4 Power Overhead 52
3.6.5 Unrecoverable Soft Error Rate 53
3.7 Summary 55

4 STCAM: Soft-Error Tolerant Content-Addressable Memory 57
4.1 Random Access Memory and Content Addressable Memory . . 57
4.2 CAM-RAM Caches 59
4.3 False Hits and False Misses 61
4.4 STCAM Architecture 63
4.4.1 Mitigation of False Hits 63
4.4.2 Mitigation of False Misses 63
4.4.3 Cache Access Algorithm 65
4.5 Close Hit Rate and Tag Encoding Scheme 67
4.5.1 Close Hit Rate 67
4.5.2 Distribution of Access Addresses 68
4.5.3 Tag Encoding Scheme 69
4.6 Overheads of STCAM 71
4.6.1 Cache Access Latency 71
4.6.2 Processor Performance 73
4.7 Summary 74

5 SEVA: Soft-Error- and Variation-Aware Cache Architecture 75
5.1 Introduction 75
5.2 Variation-induced Defects and Defect Tolerance 77
5.2.1 Variation-induced Defects 77
5.2.2 Defect Tolerance Techniques 79
5.3 Limitation of Existing Techniques 80
5.4 SEVA Architecture 81
5.4.1 Cache Structure and Block Classification 81
5.4.2 Inclusion Properties in Multi-level Cache Hierarchies . 83
5.4.3 Assurance Update 84

5.4.4 Assurance Update Reduction 84
5.4.5 Example SEVA Cache 89
5.5 Defect and Yield Analysis 90
5.5.1 Results 91
5.5.2 Discussion 93
5.6 Performance and Reliability Evaluation 93
5.6.1 Evaluation Methodology 93
5.6.2 Evaluation Results 95
5.7 Summary 99

6 Conclusions **101**
6.1 Conclusions 101
6.2 Suggestions for Future Work 103

Publications **105**

List of Figures

1.1 Charge generation and collection in a reverse-biased junction. 16
1.2 Error check diagram in SPARC64 processor. 21

2.1 Structure and ionized charge collection of SOI. 26
2.2 Memory protection using BICS. 29
2.3 Principle of logic design hardened storage cell. 30

3.1 Horizontal-vertical parity 37
3.2 Examples of multi-bit errors. 38
3.3 Interleaving schemes to deal with MBEs. 41
3.4 Distribution of partially- and fully-modified ECC codewords. . 46
3.5 Modified data path of L2 cache with write-through L1 cache. . 47
3.6 Write miss rate of L1 data cache. 48
3.7 An entry of write buffer. 48
3.8 Breakdown of power consumption. 53
3.9 Illustration of unrecoverable error. 54

4.1 Structures of RAM and CAM. 58
4.2 A CAM-RAM structure. 59
4.3 Structure of highly associative CAM-RAM cache. 60
4.4 Examples of false hits and false misses. 62
4.5 Mechanism for mitigating false hits. 64
4.6 Modified CAM tag for mitigating false misses. 65
4.7 Access algorithm of a STCAM cache. 66
4.8 Cache miss rate and close hit rate. 68
4.9 Distribution of the target addresses of accesses. 69
4.10 Tag encoding schemes. 70
4.11 Miss and close-hit rates with proposed tag encoding. 71
4.12 Performance of processor using STCAM. 73

5.1 Schematic of a SRAM cell. 78
5.2 Distribution of blocks at different defect rates. 82
5.3 Example of data swapping. 86
5.4 Encoding and decoding in data superimposition. 87
5.5 Example of data superimposition. 88
5.6 Example structure of a SEVA cache. 89
5.7 Yields of tag and data subarrays. 92
5.8 Normalized performance when defect rate is equal to 0.005. . . 96
5.9 Normalized performance as defect rate is varied. 96
5.10 Breakdown of accesses from L2 cache to memory per 1,000 instructions when λ is equal to 0.005. 97
5.11 Breakdown of accesses from L2 cache to memory per 1,000 instructions when λ is varied. 98
5.12 Number of written-back blocks to memory per 1,000 instructions. 98

List of Tables

3.1 Number of SECDED check bits for various data unit size . . . 34
3.2 Parameters of Simulated Architecture. 50
3.3 Area overheads of protection schemes. 52
3.4 Unrecoverable Soft Error Rate. 55

5.1 Variation of V_{th} over process generations. 77
5.2 Probability of failure cell versus σV_{th} in 45nm process. 79
5.3 Number of check bits required for different information-bit lengths 80
5.4 Yields and overheads of subarrays and cache 92
5.5 Parameters of Simulated Architecture 94
5.6 Uncorrectable error rate of L2 caches 99

Chapter 1

Introduction

Computer systems with very high dependability are required as the society becomes increasingly dependent on the computation and communication capability provided by them. *Dependable computing* has been emerging as an important and active research field, covering aspects including reliability, availability, safety, and security of computer systems [1].

Radiation-induced soft errors are expected to have serious impacts on the reliability of semiconductor devices, particularly as scaling trend continues. This thesis focuses on mitigation of soft errors to improve the reliability of memory caches and, as a result, to enhance the dependability of processors. In this chapter, we explain the problem of soft errors, and briefly review existing mitigation techniques in memory caches. We then describe the research contributions. At the end of the chapter, we describe the organization of this thesis.

1.1 Background

1.1.1 Soft Error Problem

Soft errors, also called Single Event Upsets (SEUs), in a semiconductor device refer to intermittent or transient failures caused by radiation events [2]. Soft errors cause loss of of data but do not bring about permanent damage to the device. The problem of soft errors are early recognized in 1950s [3]. Since then, extensive work has been done to investigate the mechanisms of soft errors, their impacts on the reliability of semiconductor devices, as well as to

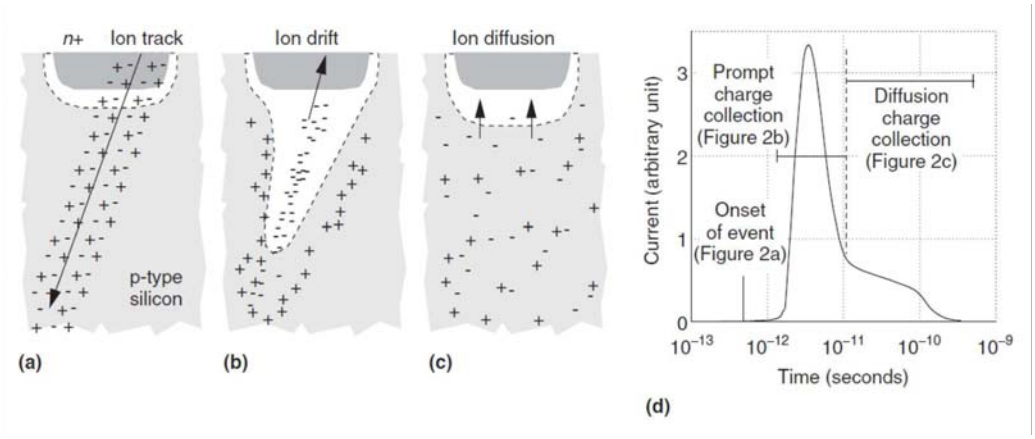


Figure 1.1: **Charge generation and collection in a reverse-biased junction.** Formation of a cylindrical track of electron-hole pairs in (a), funnel shape extending high field depletion region deeper into substrate in (b), diffusion beginning to dominate collection process in (c), and the resultant current pulse caused by the passage of a high-energy ion in (d) [4].

develop techniques to mitigate them.

Soft Error Mechanisms

The reverse-biased P/N junction node (i.e., drain node of a transistor) is the most charge-sensitive part of circuits, particularly if the junction is floating or weakly driven and consequently most susceptible to soft errors. As ionizing radiation passes through a node, electrons and holes are generated along the track of ionizing particles, as shown in Figure 1.1-a. Depending on the type and amount of energy of the incident particles, the generated charges are produced by either *direct ionization* by the incident particle, or *indirect ionization* of secondary particle produced by nuclear reaction between the incident particle and the target material [5]. When the resultant ionization track traverses or comes close to the depletion region, the electric field rapidly collects carriers, creating a current/voltage glitch at that node. Charge collection is greatly enhanced by *funneling effect* [4]. Funneling is caused by distortion of the potential into a funnel shape extending deeply into the substrate, as shown in Figure 1.1-b. Excess charges produced by a

radiation track inside this funneling region are collected very rapidly. This collection phase completes within tens of picoseconds. Then another phase follows in which diffusion begins to dominate the process. Figure 1.1-c shows the resultant current pulse.

Radiation Sources

There are several sources of high-energy particles that can cause soft errors [2]. Soft errors may be caused by alpha particles emitted from radioactive decay of contaminants in the metal, passivation layers, and package materials. Soft errors may also be caused by cosmic rays (mainly neutrons and protons) that are the by-products of interactions between high energy (tens of MeV/nucleus) galactic cosmic ray with oxygen and nitrogen in the atmosphere. The third source of soft errors comes from alpha particles released from interaction of low-energy cosmic neutrons with the isotope boron-10 (^{10}B) present in borophosphosilicate glass (BPSG), which is a common semiconductor dopant and dielectric component in IC materials [6]. In advanced processes that use highly purified chip and packaging materials with extremely low levels of uranium and thorium impurities, and where a dielectric material free of ^{10}B has replaced BPSG layers, high-energy cosmic neutrons are responsible for the majority of soft errors observed [4].

Soft errors can cause bit-upsets in dynamic and static memory elements (e.g., DRAMs, SRAMs, latches, flip-flops). Memory caches have been the main targets of soft error mitigation. The problem of soft errors in logic circuits has received attention in recent years.

Soft Error Rate and Impacts of Scaling

The amount of charges collected by a gate after a radiation event depends on many factors including the substrate structure, device doping and biasing of circuit nodes, the substrate structure, device doping, the type of ion, where the ion occurs within the devices, and the device's state [5]. The collection slop Q_s is a measure of the charge collection efficiency of a device measured in fC and it is heavily dependent on the process technology [7]. Previous work has showed that Q_s will become smaller as technology scales down [8].

The collection efficiency of a device, however, is not the only factor that determines if a soft error occurs. The sensitivity of the device to the excess charges must also be considered. Q_{crit} is the critical amount of charges that

has to be deposited to a circuit node for a soft error to occur. Q_{crit} depends on many factors including the nodal capacitance, operating voltage, circuit topology [4]. Circuit simulators can be used to calculate the critical charge of a circuit [9]. The scaling trend towards lower supply voltage and smaller device dimensions has decreased Q_{crit} [4].

For simple isolated junction (e.g., DRAM cells), a soft error will be induced when a radiation event occurs close enough to a sensitive node such that the collected charge represented by Q_s is greater than Q_{crit}. Conversely, if the event produces a Q_s less than Q_{crit}, the circuit will survive the event and no soft error will occur. In SRAM or other logic circuit having active feedback, there is an additional component of Q_{crit} related to the magnitude of the compensating current and the switching time of the device. Increasing the strength of the feedback circuit and the time to switch the device will in effect increase Q_{crit}.

A widely used empirical model relating the soft error rate (SER) to Q_s and Q_{crit} is indicated by Equation 1.1 [8]. F in the equation is the particle flux which is the number of particles per unit area per second (particles/$(cm^2 s)$). A is the area of the circuit that is sensitive to particle strikes (cm^2).

$$SER \propto F \times A \times e^{-\frac{Q_{crit}}{Q_S}} \tag{1.1}$$

DRAM's per-bit SER was high when manufacturers used planar capacitor cells that store charges in two-dimensional, large-area junctions. These cells were very efficient at collecting radiation-induced charges. DRAM manufacturer since then adopted three-dimensional capacitor designs that significantly increase Q_{crit} while greatly reducing junction collection efficiency by eliminating the large storage junctions. DRAM's per-bit SER has been shrinking about 4x to 5x per generation. The increased memory density (bits per system) almost as fast as the SER reduction that technology scaling provided. DRAM's system SER has remained essentially unchanged over generations [4].

A SRAM is more susceptible to soft errors than a DRAM. Designers have deliberately minimized the SRAM junction area to reduce capacitance, leakage, and cell area while aggressively scaling down SRAM's operating voltage to minimize power consumption. Which each successive SRAM generation, a reduction in operating voltage and node capacitance has cancelled out the reduction in cell collection efficiency caused by shrinking cell depletion volume. Initially, SRAM's per-bit SER was increasing with each successive generation.

More recently, as feature sizes have size have shrunk into the deep-submicron range, SRAM's per-bit SER has reached saturation and might even be decreasing. This saturation is primarily due to the saturation in voltage scaling, reduction in junction collection efficiency, and increased charge sharing caused by short-channel effects with neighboring nodes. SRAM's per-chip SER is expected to increase at most linearly with decreasing feature size [8].

A transient pulse caused by ionizing event can propagate through logic gates and finally be latched by a sequential element, resulting in an incorrect output. Whether an erroneous pulse in a logic circuit resulting in an incorrect output depends on three masking effects: logical masking, electrical masking and latch window masking. Logical masking happens when one of inputs of a gate is in controlling state (e.g., 0 for a NAND gate) so that then transient pulse at another input is blocked. Electrical masking happens when pulse is attenuated by subsequent logic gates because of the electrical property of the logic gate. Latch window masking means the arrival transient pulse is outside of the latching window for the sequential elements. The impacts of these masking effects on SER of logic circuit have been studied [10] [7] [11]. SER of logic circuits is expected to increase rapidly and contribute a significant fraction of system SER [7].

The commonly used unit of measure for the soft error rate (SER) and other hard-reliability mechanisms is the FIT (failure in time). A FIT is equivalent to one failure in 10^9 device hours. Soft errors have become a huge concern in advanced computer chips because, uncorrected, they produce a failure rate exceeding that of all other reliability mechanisms combined. For example, a typical failure rate of a hard-reliability mechanism (e.g., gate-oxide breakdown, metal electro-migration) is about 5 to 150 FITs. However, without mitigation, SER can easily exceed 50,000 FITs per chip [4].

Necessity of Soft Error Tolerance

Traditionally, soft errors were regarded as a major concern only for space and avionics electronics. Since the flux of radiation particles increases with high altitude, electronics working in high altitude environments are exposed to a much higher SER than those on ground. Specifically, the sea-level flux is several hundred times smaller than the fluxes seen at aircraft altitudes [12]. The necessity of employing measures against soft errors in space and avionics electronics also stems from the fact that these applications are extremely mission-critical such that even a small SER is unacceptable.

Until recently, the problem of soft errors was largely ignored in commodity electronics. Commodity electronics are highly cost-sensitive so that any measure against soft errors that imposes a relatively high cost was commonly considered unacceptable by users. Moreover, SER was low enough so that the costs and design efforts spent to implement the measures were unjustified. However, the situation has been increasingly changing. The problem of soft errors in commodity electronics cannot be negligible any more. Device scaling enables large scale integration. System-level SER therefore grows rapidly. Customers become more aware of, and demand stringent tolerance requirements for soft-error problem. These factors make soft error mitigation indispensable not only for space and avionics electronics, but also for commodity electronics.

Soft-error induced failures in commodity electronics have been reported. SUN servers crashed due to lack of soft error tolerance in the designs of UltraSparc II processor and its internal cache [13]. Cisco's 12000 line-card router reset after SEU failures, requiring two or three minutes to recover [14]. FPGA-based Q Cluster and System X, which were the second and third fastest supercomputers on the November 2003 Top 500 Supercomputer Sites list, experienced fatal failures caused by soft errors [15]. The Q cluster, for example, experienced 26.1 CPU failures a week [15]. Such failures had unavoidably led to customer's dissatisfaction, damaged reputation of companies. Advanced processor designs have put a great emphasis on soft error tolerance. Figure 1.2 shows the diagram of SPARC64 processor [16] [17]. Memory caches, data paths, and 83% of latches in the processor are protected from soft errors, either by parity or Error Correcting Code (ECC).

1.1.2 Soft-Error Tolerance in Memory Caches

Memory speeds are increasing at a much slower rate than processor speeds [18]. So even though processors are becoming faster, the overall performance declines because of the slower memory. The processor will spend increasing portion of execution time waiting for data to be brought from memory. To bridge the gap between memory speed and processor speed, an advanced processor usually devotes a majority of hardware resources for its memory caches. For instance, Itanimum 2 processor devotes 86% of the transistors for its L3 cache [19]. Moreover, memory caches are usually SRAMs that are highly vulnerable to SEUs. Having memory caches to be resilient against soft errors is indispensable for attaining high processor's reliability.

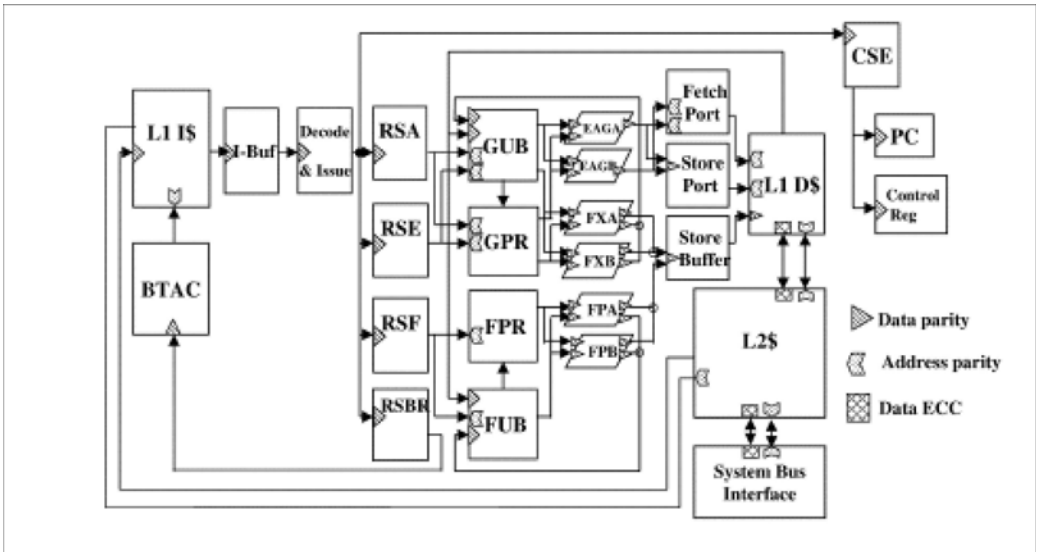


Figure 1.2: **Error check diagram in SPARC64 processor.** Caches and data paths are protected by either parity or ECC.

Soft errors in memory caches can be mitigated at multiple levels. At process and technology level, by using purified materials or employing shielding, soft errors caused by alpha particles can be mostly eliminated [20] [21]. Charge collection efficiency can be reduced in processes with extra doping layers [22] [23] [24] [25], or silicon-on-insulator (SOI) [26]. While techniques at this level can provide an error-hardened substrate for all circuits built on it, combinations of them with techniques at higher levels are usually required for achieving a low SER, particularly as neutron-induced soft errors are increasingly become a major concern.

Coding techniques (e.g., parity and ECC) are very effective in protecting memory caches, thanks to the high regularity of memory arrays. The overheads of coding techniques are reasonable and acceptable in most cases. Single-Error Correction Double-Error Detection Hamming code (SECDED) is the most widely used ECC in practice. As we point out later, SECDED can incur a high area overhead for some cases. Multi-bit errors can be tolerated by combining SECDED with *interleaving* [27] [28], or *scrubbing* [29] [30].

Circuitry of individual memory cells can be modified to make them more robust against soft errors [31] [32] [33]. However, since a hardened cell typ-

ically requires a significant larger area than a typical cell, these techniques are not suitable for cost-sensitive applications.

By exploiting architectural properties such as dirtiness or access frequency of data, error protection can be selectively performed on those data having high impacts on the cache's SER [34] [35] [36]. While such an approach requires less area overhead than the approach treating all data equally, its effectiveness is dependent on the amount of locality of data access in execution programs.

1.2 Research Contributions

This research focuses on mitigating soft errors in memory caches. We make the following contributions.

- We proposes **Zigzag-HVP**, a cost-effective technique to detect and correct soft errors for such caches. While ECC is widely used for SEU mitigation in VLSI caches, we show that ECC is expensive for being implemented in caches with word-based access. Zigzag-HVP utilizes the concept of horizontal-vertical parity (HVP). HVP maintains parity of a data array in two-dimensional directions: horizontal and vertical. While requiring fewer check bits than ECC, a basic HVP scheme can detect and correct only a single bit error (SBE), but not a multi-bit error (MBE). By dividing the data array into multiple HVP domains and interleaving the bits of different domains, a spatial MBE can be converted to multiple SBEs, each of which can be detected and corrected by the corresponding parity domain. Vertical parity update and error recovery in Zigzag-HVP can be performed efficiently by modifications to the cache data paths, write-buffer, and Built-In Self Test. Evaluation results indicate that the area and power overheads of Zigzag-HVP caches are lower than those of ECC-based ones.

- We proposes **STCAM**, a soft-error tolerant Content-Addressable Memory (CAM) architecture. Due to differences in circuit structure and access nature, mitigation of soft errors in a CAM is more difficult than in a RAM. STCAM can detect and correct false hits and false misses caused by soft errors in a CAM. The technique involves subdividing a CAM and providing backup checking for cases the input is partially matched in a CAM search. An original encoding scheme is proposed to

reduce the frequency of backup checking. Our evaluation results show that modifications to support STCAM do not increase access latency. Performance degradation incurred for false miss checking is very low.

- We propose **SEVA**, a cost-effective soft-error- and variation-aware cache architecture. As devices are scaled down, the number of variation-induced defective memory cells increases rapidly. The proposed SEVA combines SECDED with a redundancy technique to effectively tolerate such a high number of defects. While SECDED can repair a defective cell in a block, the block becomes vulnerable to soft errors. To remedy the problem, SEVA allows only clean data to be stored in the defective (but repairable) blocks. Such constraint is enforced through a mechanism called *assurance update*. An error occurring in a defective block can be detected and the correct data can be obtained from the lower level of the memory hierarchy. We also propose techniques to reduce the frequency of assurance update.

1.3 Thesis Organization

The remaining of this thesis is organized as follows. Chapter 2 provides a detail survey on existing soft-error mitigation techniques in memory caches. Chapter 3 presents **Zigzag-HVP**. Chapter 4 presents **STCAM**. Chapter 5 presents **SEVA**. Finally, Chapter 6 makes conclusion.

Chapter 2

Existing Work on Soft-Error Tolerance in Memory Caches

This chapter surveys existing techniques for mitigating soft errors in memory caches. The problem of soft errors can be addressed at multiple levels. The techniques presented here are classified based on at which level they are implemented. We then identify those areas that require improvements. Following chapters will propose techniques dealing with them.

2.1 Process- and Technology-Level Hardening Techniques

The most obvious way to eliminate soft errors is to remove the radiation sources that cause them. To mitigate the dominant SER cause by low-energy neutrons and ^{10}B, manufacturers have removed BPSG from virtually all advanced technology [4]. To reduce alpha particle emissions, semiconductor manufacturers use extremely high purity material and processes, production screening all materials having low background alpha emission measurements. Purified interconnect metal is used to reduce the natural alpha emission [20]. Coating the chip surface with a thick polyimide layer before packaging, can block alpha particles with energies up to approximately 9 MeV [21]. This would shield the active silicon from the particles being emitted from the ceramic packaging. Although large SER reductions are possible either by removing the sources of or shielding the ^{10}B reaction products and alpha particles, a large portion of the high-energy cosmic neutrons will always reach

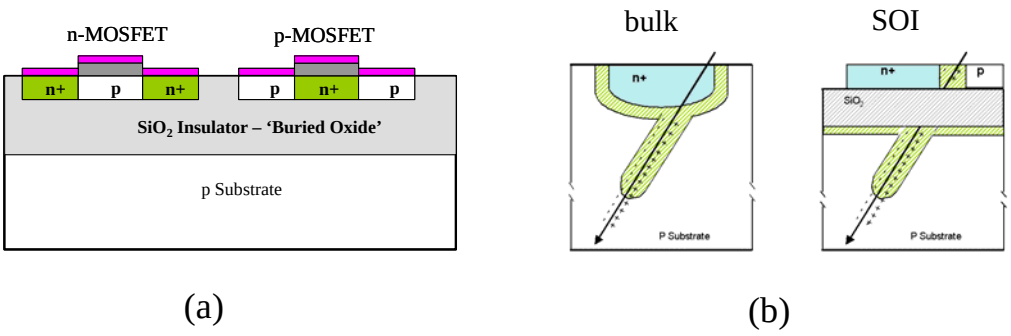


Figure 2.1: **Structure and ionized charge collection of SOI.** MOSFET is formed over a buried oxide layer in (a). The charge volume in SOI is significantly reduced thank to isolation by buried oxide layer, as compared with a bulk device in (b).

the devices and cause soft errors. Ultimately, high-energy cosmic neutron radiation defines the SER limit.

Modern methods for technology hardening of memories against SEU rely on reducing charge collection at sensitive nodes. Introducing extra doping layer as a novel p-well protection barrier can restrict substrate charge collection [22]. Triple-well [23] or quadruple-well [24] structures can improve resistivity against soft errors. Built-in junctions in these structure increase the recombination of charges far away from the active region. Epitaxial substrates reduce charge collection by funneling and provide some benefits in SEU reduction [25].

In addition to offering high-speed and low-power, silicon-on-insulator (SOI) is also renown for their resilience again SEUs [26]. Figure 2.1-a show the cross-section of SOI CMOS structure. MOSFET is formed on a thin SOI layer over a buried oxide layer, and the entire MOSFET is enclosed in a silicon oxide layer. The device is fabricated in a thin silicon layer that is isolated from the substrate by a buried oxide layer that acts as a dielectric. After an ionized strike, the charges deposited in the silicon substrate below the buried oxide cannot be collected at the drain because of the isolating dielectric between the drain and the substrate. The charge volume is significantly reduced in SOI, as compared with bulk device, as shown in 2.1-b. The charge collection depth thereby reduces from as much as a few microns in

bulk-Si devices to 100 to 300nm in SOI devices. Moreover, the buried oxide results in the absence of junction depletion region below the source and the drain. Thus, the P/N junction area is considerably reduced as compared to bulk CMOS processes, resulting in increased soft error immunity.

The abovementioned process- and technology-level hardening techniques can provide a significant improvement in SER reduction. Nevertheless, these techniques are not silver bullets that can provide a total solution to the soft-error problem. Combinations with hardening techniques at other levels are required in order to achieve a high degree of soft error tolerance.

2.2 Circuit- and System-Level Hardening Techniques

Parity and error-correcting code (ECC) are widely employed in practice to protect memory caches from soft errors. In a parity scheme, a parity generator computes the parity bit of the memory word to be written into a cache. The parity bit are also stored in the cache. If a particle strike inverts one bit of a memory word, the error can be discovered by checking the parity code when the memory word is read. The area overhead of a parity scheme is very low. However, the drawback is that the scheme can detect but *cannot* correct the error. Parity scheme is sufficient for a cache storing only clean data (e.g., an instruction cache or a write-through data cache). Since all the data in the cache can also be found in memory, error recovery is simply a matter of discarding corrupted data and fetching correct data from memory. However, parity scheme is not sufficient for a cache holding dirty data (e.g., a write-back cache). Corruption of dirty data in the cache can leave the system into unrecoverable state if the data are referenced later. Error correcting capability provided by ECC is required for such a cache.

While numerous error correcting codes exist, the incurred area and speed penalty for implementing them can be excessive, especially for the ones that enable correction of multiple errors. Single-Error Correction Double-Error Detection Hamming code (SECDED) is the most widely-used ECC in practice. As implied by its name, SECDED can detect and correct a single-bit error, or detect a double-bit error in a codeword. The overheads incurred by SECDED is moderate and acceptable in most cases. The encoding/decoding circuitry of SECDED can be constructed as XOR trees and is quite fast.

A multi-bit error (MBE) in a codeword may be left undetectable, or be detectable but uncorrectable by SECDED. An MBE may result from the error bits accumulated from multiple particle strikes over time, or the error bits caused by a single particle strike. We call the former case a *temporal* MBE, and the latter case a *spatial* MBE. The scaling trend toward smaller device dimensions and lower supply voltages has increased the probability that a strike will result in a spatial MBE [37] [38]. Interleaving different codewords in the same row can disperse the error bits of a spatial MBE into multiple codewords so that each error bit can be detected and corrected by SECDED [27] [28].

The probability of a temporal MBE can be negligible for a small, frequently-accessed memory cache [29]. However, for a very large cache or cache that may be idle for a long period while still holding data, the probability of a temporal MBE may not be insignificant. The probability of a temporal MBE can be reduced by *scrubbing* [30]. Scrubbing consists of periodically reading out the data from a memory cache, correcting any latent error, recomputing ECC, and writing the bits back. If scrubbing interval is short enough, the opportunity for a temporal MBE to arise is practically eliminated.

Nicolaidis et al. proposed the use of current sensors to protect memory from SEUs [39] [40]. A memory cell being in the steady state drives a very small current. But when its state is reversed due to the impact of an ionized particle, an abnormal current flows through the Vdd and Gnd lines of the cell. It is then possible to detect the occurrence of a SEU by implementing one current sensor (BICS) on each vertical power line of the cell array, as shown in Figure 2.2. Because BICSs monitor the vertical power lines of the memory, the detection of a SEU also indicates the position of the affected bit. Since memory cells on the same column can share a censor, the area overhead of BICS is lower than that of ECC. However, implementing BICS significantly increases the design complexity. Moreover, the efficacy of this technique relies on the capability of a sensor to capture a small current. Properly capturing the current will become more difficult due to larger variations in scaled process technologies.

Techniques to make the static storage cells to be radiation-hardened have been proposed. Those cells preserve their states even if the electrical states of some of their nodes is altered by an ionizing particle strike. Resistance elements can be added to the gate or drain terminals of transistors of the cross-coupled inverters in a SRAM cell [31] [32]. Those resistance elements will form RC low-pass filters filtering out the high-frequency components

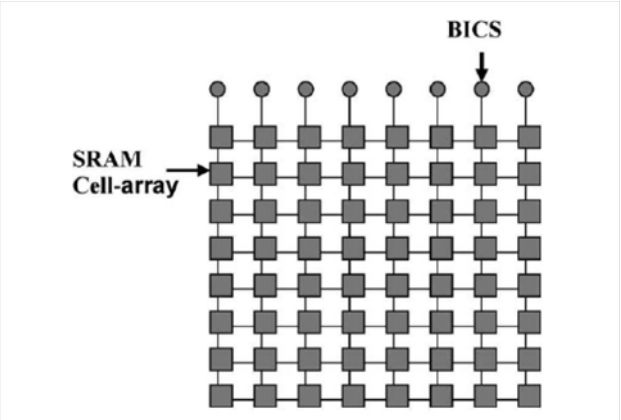


Figure 2.2: **Memory protection using BICS.** A current sensor (BICS) added to each column of a cell array can detect the abnormal current accompanied by ionized particle striking on that row.

of a SEU spike. The main disadvantages of these techniques are that they increase the cell's size, and possibly degrade the write time and sense time of a memory cell.

Dual interlocked storage cell (DICE) employs logic redundancy into a memory cell [33]. Figure 2.3 shows the principle of DICE. The two latches L1 and L2 store the same data, with the data in the uncorrupted latch providing state-preserving feedback to the corrupted latch. The differential outputs OA, OB of each latch section are connected to the differential feedback input IA, IB of the opposite, dual latch section. The basic drawback of DICE cells is their hardware cost, which is generally close to duplication.

2.3 Architectural Level Hardening Techniques

Conventional ECC is implemented in a *uniform* manner. That is every unit of data is protected by a check code of the chosen capability. Such an implementation treats data equally. However, cache lines stored in a memory cache can have different access frequency. Most frequently used (MFU) cache lines are most error-prone and errors in those blocks easily propagate unless checked. Kim et.al., proposed the concept of *parity caching* in which a small cache is allocated to store check bits of recently used data [34]. Parity caching

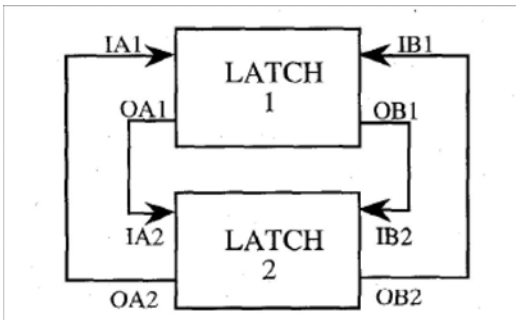


Figure 2.3: **Principle of logic design hardened storage cell.**

can obtain a high reliability with lower area overhead as compared with a uniformly-implemented ECC. They also proposed *shadow checking* in which the copies of MFU blocks are stored in a shadow cache [34]. The underlying idea is the same as parity caching, but the shadow cache performs error checking by means of comparison using the copies of data rather than check codes. The goal of shadow checking is to obtain a high reliability enhancement even in the presence of multi-bit errors with smaller chip area overhead. *Replication cache* enhances shadow checking concept by using a small fully-associative cache to store the replica of every write to the L1 cache [35]. *ICR cache* uses these cache lines that are predicted to be "dead" as replicas of "hot" dirty data [36]. While these techniques offer lower area overheads than a uniform ECC, a large portion of the dirty data still remains unprotected if the locality of the data is low. The level of dependability achieved by these techniques is obviously not acceptable for applications demanding a very high reliability. In addition, while shadow checking and replication cache may be applicable to small L1 caches, application of them to large L2 caches is impractical [35].

The use of *early writeback* to improve cache reliability has been proposed [41]. Conventionally, a cache line stays in a cache until a cache miss occurs and the cache line is chosen as an eviction candidate. If a dirty cache line in a cache has a long lifetime, the possibility that the cache line being corrupted by SEU and causing an unrecoverable error can be high. By early writing back dirty blocks to a lower-level cache, the lifetime of the dirty blocks can be reduced. Instead of using ECC that incurs a high area overhead, simple parity can be sufficient to protect a cache employing early writeback. When

a clean cache line is verified to be corrupted by parity, the correct data can be obtained from the lower-level cache. Interestingly, early writeback may also have a favorable side-effect on processor performance. If a program makes frequently data updates, the writeback buffer may be full requiring the processor to stall its pipeline for the buffer to write back its old entries. Early writeback can improve performance by reducing those stall cycles caused by a full writeback buffer [42]. Periodically refreshing a cache with data from the lower-level cache is proposed in [43]. While these techniques improve the reliability of a cache with low area overheads, they increase the access activity to a lower-level cache and consequently its power consumption.

When hardware ECC is not available, software-implemented ECC can be used to protect memory caches [44] [45]. Given the address and size of the data block that needs to be protected, ECC software requests the operating system to allocate another block. Then, it calculates the check bits and stores them in the allocated block. Any modification to the data block requires its check bits to be recomputed by the means of software. Software-based ECC may be applicable to a code section or a less-frequently updated data section. However, for a data section receiving frequent updates, software-based ECC is impractical since the performance overhead of recomputing check bits is excessive in this case. Given that hardware-based approach is transparent to user's software and ECC support has increasingly being the norm in modern memory designs, the benefit of software-based ECC has become less appealing.

2.4 Room for Improvements

2.4.1 Overhead Reduction

Error-correcting code (ECC) is an effective technique to protect data from soft errors. ECC may incur a moderate area cost that is acceptable for most cases. However, for some designs, the area penalty incurred by ECC may be undesirable. In order to keep area overhead low, a large codeword is preferred. However, in caches where data are accessed on a per-word basis, maintaining ECC on a per-word basis is preferred. Otherwise, if large codewords consisting of multiple data words is used for such caches, partial updates to the large codeword will frequently occur, incurring expensive *read-modify-write* operations that read the entire codeword, recompute the

check bits, and write back the codewords. However, the cost of maintaining ECC on a per-word basis is high. For instance, a 32-bit word requires seven SECDED check bits and incurs a 22% area overhead. The proposed Zigzag-HVP described in Chapter 3 will provide cost-effective technique mitigate soft errors in caches with word-based access.

2.4.2 Soft Error Mitigation in Content-Addressable Memory

Memories can be classified into two types based on their access mechanisms: Random Access Memories (RAMs) and Content Addressable Memories (CAMs). Due to the difference in access mechanism, RAM and CAM require different mechanisms for mitigating soft errors. Coding techniques such parity or ECC have proved to be very effective in mitigating soft errors in RAMs. However, those techniques are not immediately applicable to CAMs because they depend on processing the full contents of the memory word outside the array, which is not possible in a normal CAM access. Mitigation of soft errors in CAMs is more challenging than in RAMs and has received little attention so far. However, since several components in a processor are built as CAMs, having those components to be resilience against soft errors necessary to increase the coverage of error protection in a processor. The proposed STCAM in Chapter 4 can deal with soft-error problem in CAMs.

2.4.3 Pursuit of Soft-Error- and Defect Tolerance

Variation-induced defects become severe in scaled processes [46] [47]. Previous work has shown that ECC is effective in tolerating a high number of random defects in memory caches [48] [49]. It would be cost-effective if the same ECC resource can be used for both soft error tolerance for reliability and defect tolerance for yield. However, while a defective cell present in a block can be repaired by ECC, the block becomes vulnerable to soft errors. An error occurring in the block may be detectable but *uncorrectable*. This can get the processor system into an unrecoverable state, particularly when the corrupted data are dirty data that have no backup elsewhere in the memory hierarchy. The proposed SEVA in Chapter 5 can deal with this problem.

Chapter 3

Zigzag-HVP: Soft-Error Mitigation in Caches with Word-based Access

3.1 Introduction

Error Correction Code (ECC)—specifically Single Error Correcting and Double Error Detecting Hamming Code (SECDED)—is widely used to detect and correct soft errors in caches. For a SECDED codeword consisting of k information bits and c check bits, the relation between k and c is given by Equation 3.1.

$$2^c \geq k + c + 1 \tag{3.1}$$

Table 3.1 shows the number of check bits, and overhead of SECDED when the codeword size varies. The number of check bits just increases linearly as the codeword size increases exponential. Therefore, in order to keep the overhead low, a large codeword is preferred. However, this is not true for caches with word-based access where data are accessed on a per-word basis. Exemplified caches with word-based access are L1 data caches or L2 caches with write-through L1 caches. The later are commonly found in the memory hierarchy of multiprocessor systems because maintaining cache coherency at L2 cache is simpler if the accompanied L1 cache is write-through. For these caches with word-based access, maintaining ECC on a per-word basis is preferred [50] [51]. Otherwise, if large SECDED codewords consisting of

Table 3.1: **Number of SECDED check bits for various data unit sizes.** Doubling the size of data unit increases the number of check bits by one. Large data unit is preferred to keep the overhead low

Data unit size (bits)	16	32	64	128	256
Number of check bits	6	7	8	9	10
Overhead (%)	37.5	21.9	12.5	7.0	3.9

multiple words is used, partial updates to the large codeword will frequently occur, incurring expensive *read-modify-write* operations that read the entire codeword, recompute the check bits, and write back the codewords. However, the cost of maintaining SECDED on a per-word basis is high. For instance, a 32-bit word requires seven check bits and incurs a 22% area overhead.

This chapter describes **Zigzag-HVP**—a low-cost technique to detect and correct soft errors for these word-based accessed caches. The technique makes use of horizontal-vertical parity (HVP) [52]. HVP maintains the parity of the data array both horizontally and vertically. Basic HVP can detect and correct a single bit error (SBE), but not a multi-bit error (MBE). The probability that multiple errors will be accumulated from multiple particle strikes is very low in frequently accessed caches. Instead, MBE is mostly caused by a single particle strike that corrupts multiple bits at once [38]. Therefore, the corrupted bits are located close to one another. By dividing the data array into multiple HVP domains and interleaving bits of different domains, a spatial MBE is converted into multiple SBEs, each of which can be detected and corrected by the corresponding domain.

An error recovery routine is executed when an error is detected by horizontal parity. By sequentially reading the data words belonging to the parity domain, vertical parity can be recomputed to locate the error bit. The Built-In Self Test (BIST) [53] [54] is enhanced to include such a recovery function. We also modify the cache data path and write buffer to efficiently accommodate vertical parity updates. The evaluation results indicate that the area and power overheads of Zigzag-HVP caches are lower than those of the ECC-based ones.

The remainder of this chapter is organized as follows. Section 3.2 discusses related work. Section 3.3 explains the basic concept of HVP and its limitations. Section 3.4 describes Zigzag-HVP. Section 3.5 discusses the ap-

plication of Zigzag-HVP. Section 3.6 presents the evaluation results. Finally, Section 3.7 summarizes this chapter.

3.2 Related Work

The use of a small cache to store check bits or replicas of recently used data has been proposed [34]. Replication cache [35] enhances this concept by using a small fully associative cache to store the replica of every write to the L1 cache. ICR cache [36] uses these cache blocks that are predicted to be "dead" as replicas of "hot" dirty data. While these techniques offer lower overheads than ECC, a large portion of the dirty data still remains unprotected if the locality of the data is low. The portion of unprotected data for some benchmarks has been as high as 40% [34], 30% [36], or 5% [35]. The level of dependability achieved by these techniques is obviously not acceptable for applications demanding extremely high reliability. In addition, while a replication cache [35] may be suitable for small L1 caches, its application to large L2 caches is impractical. Zigzag-HVP can reduce the error rate by many orders of magnitude and can be applied to both L1 and L2 caches.

Cross-parity [55] can deal with MBEs by maintaining parity in the diagonal parity, in addition to the horizontal and vertical. However, an examination of the horizontal parity alone cannot expose the existence of some MBEs (e.g., MBEs where the number of error bits in the same row is even). Recomputation and checking of vertical or diagonal parity are required to detect such MBEs. However, these operations are expensive and it is impractical to execute them on every data access. By relying on interleaving to disperse the error bits, the existence of spatial MBEs in Zigzag-HVP can be detected by only examining the horizontal parity and the vertical parity only needs to be recomputed after the errors have been detected.

Nicolaidis et al. propose the use of current sensors that are built into the vertical power lines of a data array to detect the abnormal current dissipation accompanied with a particle strike [39] [40]. Memory cells on the same column can share a censor, resulting in a lower area overhead than ECC. However, the efficacy of this technique relies on the capability of the sensors to capture the small current; capturing the current more difficult due to larger variations in scaled process technologies. Zigzag-HVP is more scalable since it is not affected by process variations, and the degree of interleaving can be easily increased to deal with a higher probability of MBEs in scaled

process technologies.

The concept of interleaving has been used to combat burst errors (e.g., in data communications, compact disks, magnetic storages, hologram memories, and two-dimension barcodes). Existing interleaving schemes usually require intensive computation to be able to detect and correct the errors. However, on-chip caches are latency-critical, making these complex interleaving schemes unacceptable. Simple interleaving of ECC datawords in the same row to tolerate MBEs has been used in caches or memories [27] [28]. Zigzag-HVP interleaves the data in two dimensions and its practicality is explained in this chapter.

3.3 Horizontal-Vertical Parity and Its Limitations

Since our proposed Zizag-HVP is based on horizontal-vertical parity (HVP), this section first describes the concept of a basic HVP. We then explains the limitations of the basic HVP, namely HVP is unable to detecting and/or correcting multi-bit errors (MBEs). We then describes the characteristics of MBEs and the impacts of process technology having on the trends of MBEs. How such limitations of basic HVP are overcome by Zigzag-HVP is described in the next section.

3.3.1 HVP Concept

Figure 3.1 shows the concept of HVP [52]. The parity of the $m \times n$ cell array is maintained both horizontally and vertically. $d_{i|j}$ denotes a data bit in the i-th row and j-th column of the array. hp_i are vp_j are respectively the parity bits of the i-th row and j-th column. hvp is the sum parity of the vertical parity bits and this is also equal to the sum parity of the horizontal parity bits.

HVP can detect and correct single-bit errors (SBEs). Assume that the bit $d_{i|j}$ is corrupted. When row i is accessed, the parity of the row is recomputed and compared to hp_i. A mismatch indicates the existence of an error in the row. The vertical parity is recomputed by sequentially reading all rows of the array. The resultant vertical parity are compared to those previously stored in the array. A mismatch in column j of the vertical parity indicates the position of the error bit in the victim row.

$$
\begin{array}{cccc|c}
d_{1|1} & d_{1|2} & \ldots & d_{1|n} & hp_1 \\
d_{2|1} & d_{2|2} & \ldots & d_{2|n} & hp_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
d_{m|1} & d_{m|2} & \ldots & d_{m|n} & hp_m \\
\hline
vp_1 & vp_2 & \ldots & vp_n & hvp
\end{array}
$$

$$
\begin{aligned}
hp_i &= d_{i|1} \oplus d_{i|2} \oplus \ldots \oplus d_{i|n} \\
vp_j &= d_{1|j} \oplus d_{2|j} \oplus \ldots \oplus d_{m|j} \\
hvp &= vp_1 \oplus vp_2 \oplus \ldots \oplus vp_n \\
&= hp_1 \oplus hp_2 \oplus \ldots \oplus hp_m
\end{aligned}
$$

$\oplus$ denotes Exclusive OR

Figure 3.1: **Horizontal-vertical parity.** The parity is maintained both horizontally and vertically.

For an $m \times n$ data array, HVP requires $m+n$ check bits or an $(m+n)/(m \times n)$ area overhead. The overhead is small when m and n are sufficiently large.

3.3.2 Limitations with Basic HVP

While the basic HVP scheme previously described can detect and correct an single-bit error (SBE), it may be unable to detect and correct a multi-bit error (MBE). Figure 3.2 shows several cases of MBEs in a 4×4 cell array. In case A, since the corruption of both $d_{2|1}$ and $d_{2|2}$ leaves the parity of row 2 unchanged, horizontal parity is unable to detect the existence of the errors. Examining vertical parity can show the existence of the error. However, recomputation of vertical parity is an expensive operation which requires all the rows to be read from an array. Examining vertical parity on every cache access is therefore impractical. In case B, while the existence of errors in row 3 and row 4 can be detected by hp_3 and hp_4, the exact positions of the errors in the rows cannot be revealed by vertical parity since the vertical parity of column 1 remains unchanged. In case C, while horizontal parity reveals the existence of errors in rows 3 and 4 and vertical parity reveals the existence of errors in columns 3 and 4, HVP still cannot determine which one of the two pairs $(d_{3|4}, d_{4|3})$ or $(d_{3|3}, d_{4|4})$ has been corrupted.

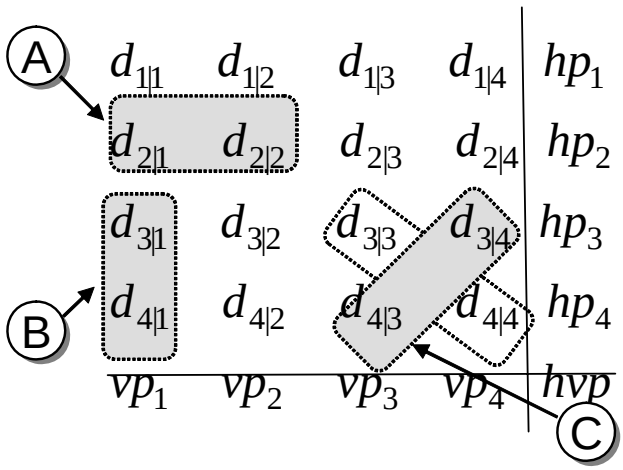


Figure 3.2: **Examples of multi-bit errors.** Multi-bit errors may be undetectable by horizontal parity (case A), or detectable but uncorrectable (case B and C).

3.3.3 Multi-Bit Errors and Characteristics

An MBE results from the error bits accumulated from multiple particle strikes, or the error bits caused by a single particle strike. We call the former case a *temporal* multi-bit error, and the latter case a *spatial* multi-bit error.

In a temporal MBE, a particle strike occurs and corrupts a single bit. Another strike occurs and corrupts a different bit before the erroneous data are accessed through which the error could have been detected and corrected. The error bits in a temporal MBE are *randomly* located in the data array. Temporal MBE can occur in very large memories where the data might not be accessed for a long time and the probability that undetected errors will accumulate cannot be ignored.

A spatial MBE is resulted from a single strike corrupting multiple bits at once. In contrast to a temporal MBE, the error bits in a spatial MBE are *closely* located. The scaling trend toward smaller device dimensions and lower supply voltages has increased the probability that a strike will result in a spatial MBE [37] [38]. Experiments have confirmed the existence of up to four-bit spatial MBEs for SRAM fabricated in a 90 nm process [38].

3.4 Zigzag-HVP

Since on-chip caches usually have a high access frequency, an error generated by a strike is likely be detected and corrected before the next strike occurs. The results in Section 3.6 confirm that probability for a temporal MBE to occur in an on-chip cache is very low. We therefore focus on measures against spatial MBEs.

Zigzag-HVP enhances the basic HVP to effectively deal with spatial MBEs. Zigzag-HVP exploits the property that the error bits in a spatial MBE are closely located. Zigzag-HVP groups the data bits of a cache into multiple *parity domains*. Each parity domain consists of several data words and is protected by HVP. By interleaving different parity domains, the spatial error bits in a spatial MBE are converted to multiple SBEs. Each SBE belongs to a parity domain that can be successfully detected and corrected. The bit interleaving scheme, parity update mechanism, error detection, and recovery mechanism are described in this section.

Let us define some terminology. A cache can be expressed as a data array with N_R rows. Each row contains N_L cache lines, each line contains N_W words, and each word contains N_B bits. Cache size is $N_R \times N_L \times N_W \times N_B$ bits. The cache size, N_W, and N_B are usually specified in advance in a cache design. Given the predetermined architectural parameters (e.g., cache size, N_W, N_B, cache associativity), tools like Cacti [56] can automatically calculate N_R, N_L that yield good trade-offs between access time and power efficiency.

3.4.1 Bit Interleaving Schemes

A spatial MBE can have multiple error bits in the same row, or the same column [38]. We will now describe the interleaving schemes to deal with these patterns of error bits.

Dealing with Horizontal MBEs

Interleaving layout of words converts adjacent error bits on the same row into SBEs in different words. ECC-protected caches have used this technique to effectively reduce MBEs on the same ECC unit [27] [28]. We use the same technique in Zigzag-HVP to deal with horizontal MBE. The words interleaved in a row belong to different parity domains. Figure 3.3-a illustrates a simple

example; two words are interleaved in each row and up to two bit errors can be tolerated (b_i^j indicates the j-th bit of the i-th word in a row).

To tolerate up to d bit errors, we must interleave at least d different words. Upon data access, only one bit is multiplexed for every d consecutive bits. We have two possible options in choosing how to interleave words horizontally: 1)interleaving words from different cache lines or 2) interleaving words from the same cache line. If the the number of cache lines in each row N_L is larger than or equal to d, we should interleave words from different cache lines. In this case, the words belonging to a cache line can be accessed at once, which is preferred for line-based cache operations such as line replacements or cache refills. If N_L is smaller than d (possible in small caches), the interleaved word could be selected from the same line. In this case, reading a whole cache line requires several cache accesses since an access can only read a subset of words of the cache line.

The number of horizontal parity bits is equal to the number of cache words ($N_R \times N_L \times N_W$) and is independent of how the words are interleaved.

Dealing with Vertical MBEs

Vertical MBE can also be dealt with by using the same interleaving concept as in the case of horizontal MBE. Consecutive bits in the same column are protected by different vertical parity bits. In Figure 3.3-b, bits in the even and odd rows are protected by a different vertical parity and up to two bit errors can be tolerated in this example. In order to tolerate up to d bit errors, this scheme requires d vertical parity bits in each column, or $d \times (N_L \times N_W \times N_B)$ bits in total. Since the vertical parity bits are frequently updated, they should be implemented as flipflops or latches which consume larger areas than normal SRAM cells. The overheads for vertical parity increase with a large d and may offset the benefit of HVP.

We instead propose an original scheme for encoding vertical parity. The HVP domain is constructed so that a vertical bit is calculated from bits located in a *zig-zag* path, rather than from bits in the same column. The physical locations of any two bits in the same zig-zag path are separated by a sufficient distance so that both cannot be corrupted by a particle strike. The number of vertical parity bits in this scheme is equal to the number of columns and is *independent* of d.

To formalize the scheme, the expression w_{ijk} ($0 \leq i < N_W$, $0 \leq j < N_L$, $0 \leq k < N_R$) is used to refer to the i-th word of the j-th cache line of the

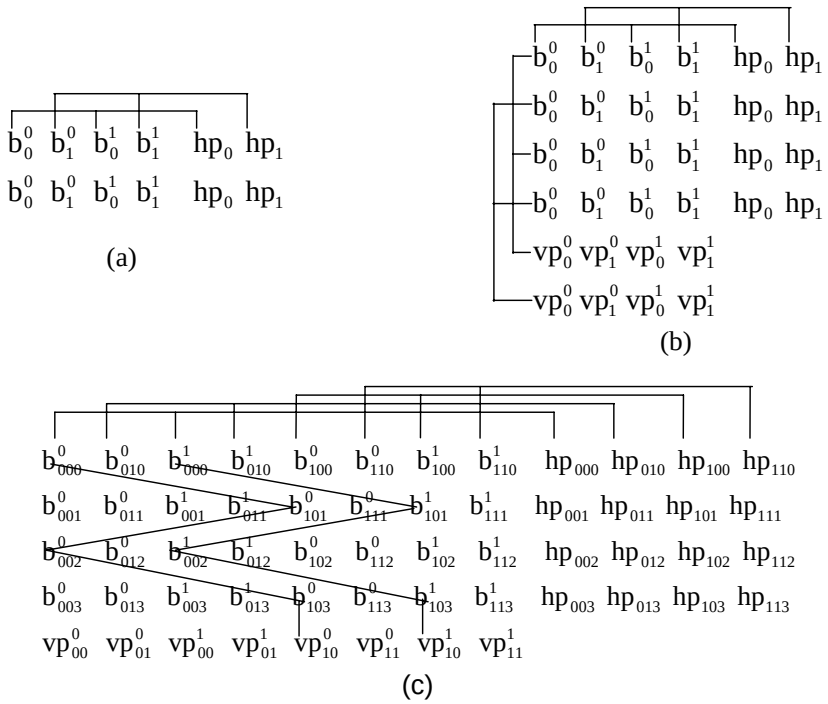


Figure 3.3: **Interleaving schemes to deal with MBEs.** Bits are interleaved to tolerate horizontal MBEs in (a). Scheme in (b) tolerates horizontal & vertical MBEs but requires many vertical parity bits. Scheme in (c) can tolerate horizontal & vertical MBEs with fewer vertical parity bits.

k-th row. A parity domain $\mathcal{PD}_{mn}$ ($0 \leq m < N_W$, $0 \leq n < N_L$) is a set of words that can be expressed by Equation 3.2.

$$\mathcal{PD}_{mn} = \{w_{inj} | \ (i - j) \equiv m \ (mod \ N_W)\} \tag{3.2}$$

Figure 3.3-c illustrates a simple example. The data array has four rows, each row contains two cache lines, and each cache line has two two-bit words. b^l_{ijk} indicates the l-th bit of word w_{ijk}. The two cache lines in each row are interleaved to tolerate horizontal MBEs. Two zigzag paths which consist of bits belonging to the same parity domain ($\mathcal{PD}_{00}$) are shown in the figure. The parity domain consists of four words: w_{000}, w_{101}, w_{002}, and w_{103}. MBEs having up to two bits can be tolerated in this example.

The proposed scheme can tolerate up to an N_W-bit vertical MBE. Given that a cache line typically consists of 8∼32 words and an SS-MBE contains no more than four bits in current processes [38], our zigzag scheme can effectively deal with vertical MBEs.

Other MBEs

If the interleaving scheme in Section 3.4.1 tolerates up to d_1 consecutive error bits in a row and the interleaving scheme in Section 3.4.1 tolerates up to d_2 consecutive error bits in a column, then any MBE in which the error bits are confined to a $d_2 \times d_1$ array can be successfully detected and corrected.

3.4.2 Parity Update Mechanism

Any update of a word (e.g., in processor writes or cache line replacements) requires the parity of the corresponding domain to be updated. From Equation 3.2, the parity domain that a word belongs to can easily be determined from the values of the bits used to index the row, and the values of the bits used to index the word inside the cache line.

Updating of horizontal parity is simple: The parity bit is newly calculated from the updated word value and stored together with the word. Updating of the vertical parity follows Equation 3.3.

$$\mathcal{VP}_{new} = \mathcal{VP}_{old} \oplus w_{old} \oplus w_{new} \tag{3.3}$$

The new vertical parity of the domain ($\mathcal{VP}_{new}$) is the exclusive-OR of the old vertical parity ($\mathcal{VP}_{old}$), old and new values of the data word (w_{old} and w_{new}). The horizontal parity bits are included in the w_{old} and w_{new}.

The update the vertical parity requires the old value of the word. Section 3.5 discusses how caches can be modified to effectively supply the old words.

A Vertical parity update can be done in parallel with writing the data word into data array. Therefore, the access latency of a Zigzag-HVP cache is comparable to that of a cache protected by a simple parity.

3.4.3 Error Recovery

When a word is read, the horizontal parity bit is recomputed and compared with the pre-stored value. A mismatch indicates the existence of a bit error in the word. A dedicated error recovery routine is then triggered. Since there is a possibility that bit error(s) may also be present in other word(s), the routine examines not only the parity domain containing the identified erroneous word but also all the parity domains. For each domain, the routine sequentially reads all the words belonging to that domain. The horizontal parity of each word and the vertical parity of the domain are recomputed, and compared with the old ones. If an SBE is present in the domain, its location can be determined.

Modern caches are typically equipped with a Built-In Self Test (BIST) [53]. The BIST accesses data in particular access patterns, also called *marching patterns*, to locate potential manufacturing defects. Modern BISTs are programmable and support various marching patterns [54]. The capability of sequentially reading the words belonging to a parity domain can be supported by extending the existing BIST hardware. The address patterns of those data words belonging to the same parity domain are derived from Equation 3.2. By accommodating such patterns into the BIST, an error recovery routine can be achieved at modest hardware cost.

Recovering from errors requires the cache to be fully scanned. Nevertheless, the overhead for error recovery is small since soft errors occurs very infrequently, For instance, let us consider a 512KB cache. Assuming a per-word access throughput of 512M-word/sec, a full scan of all parity domains in the cache requires 269 μsec. Such an overhead is incurred once every 17 years and is therefore negligible (refer to Section 3.6.5 for the cache error rate).

3.5 Applications of Zigzag-HVP

Two possible candidates for Zigzag-HVP are 1) L1 write-back caches and 2) L2 caches with write-through L1 caches. These two candidates share unique properties: They receive frequent word-based updates from processors, and they may hold dirty data so that not only detection but also correction capabilities are required to be able to fully tolerate soft errors. This section focuses on how the data paths of these caches can be modified so that the vertical parity update of Zigzag-HVP can be efficiently executed.

3.5.1 L1 Write-back Caches

Zigzag-HVP requires the old value of the modified word for vertical parity update. Before writing a word, the L1 cache needs to probe its tag to determine whether it is holding the word or not. The L1 cache is adapted so that data access occurs in parallel the tag probing. If the line is found in the L1 cache, the old value of the data word is accessible after the tag probe phase. The cache then proceeds to write the new value while the old value is passed to the vertical parity update unit. Such a modification is easily accomplished since, in practice, L1 caches already perform a tag probe and data access in parallel to achieve minimum latency.

Write-back caches usually employ a *write-allocate* policy [57]: A write miss fetches the missed line from the lower level cache and allocates the location for storing the line. In the case of a write miss, the vertical parity update unit will receive the old value of the modified word when the line is retrieved from the lower level cache.

3.5.2 L2 Caches with Write-through L1 Caches

Many processors adopt a cache hierarchy in which the L1 data cache is write-through and is backed by a large L2 write-back cache [58] [50] [59]. Making the L1 data cache a write-through cache simplifies the task of maintaining cache coherency in multiprocessor systems [60]. Maintaining simple parity for the detection of errors is sufficient for write-through L1 caches since correct data can be obtained from the L2 caches. The application of Zigzag-HVP to L2 caches is discussed hereafter.

Write Buffer and Its Implication:

A processor with a write-through L1 cache usually includes a write buffer. The write buffer has two essential functions: 1)absorbing processor writes at a rate faster than the L2 cache could, thereby preventing processor stalls and 2) coalescing writes to the same cache block, thereby reducing the traffic to the L2 cache. The reduction of word-based updates to the L2 cache by the write buffer has an implication in the implementation of ECC in the L2 cache. One could maintain ECC in large data units (e.g., double-word (64-bit) [61], or per quad-word (128-bit) [62]) to reduce the hardware overheads, and rely on the write buffer to merge updates to consecutive words into a single update of a large ECC word so that the read-modify-write operations could be reduced.

To evaluate the capability of the write buffers to coalesce modified words into a large ECC data unit, we model a superscalar processor with a write buffer of eight entries, where each entry is of 32 byte (equal to the size of L1 cache line). The L1 data cache is a four-way, 16KB, non-write-allocate, write-through cache. The details of the configurations of the processor are indicated in Table 3.6.1. The write buffer attempts to retire the oldest entry whenever more than six entries are occupied (*retire-at-6* policy [63]). Two ECC unit sizes are evaluated: double-word and quad-word. For each ECC unit size, we measure the amounts of partially-modified and fully-modified ECC units retired from the write buffer to the L2 cache for SPEC2000 benchmarks. The results are shown in Figure 3.4. A majority of ECC codewords are partially modified. These partially-modified ECC codewords require expensive read-modify-write operations and unavoidably incur significant overheads.

Per-word ECC incurs a large hardware cost, while a larger ECC unit incurs frequent read-modify-write operations even with the presence of the write buffer. Zigzag-HVP can deal with such shortcomings with ECC schemes.

Support for Efficient Vertical Parity Updates

The old values of the modified words, which are required for vertical parity update in Zigzag-HVP, can be supplied directly by the L2 cache. However, since the L2 cache is large, reading the old values from L2 caches incurs a large power overhead. Moreover, while tag access and data access are performed in parallel in L1 caches, the tag access and data access in L2 cache are done sequentially for low-power consumption. Therefore, the latency of reading

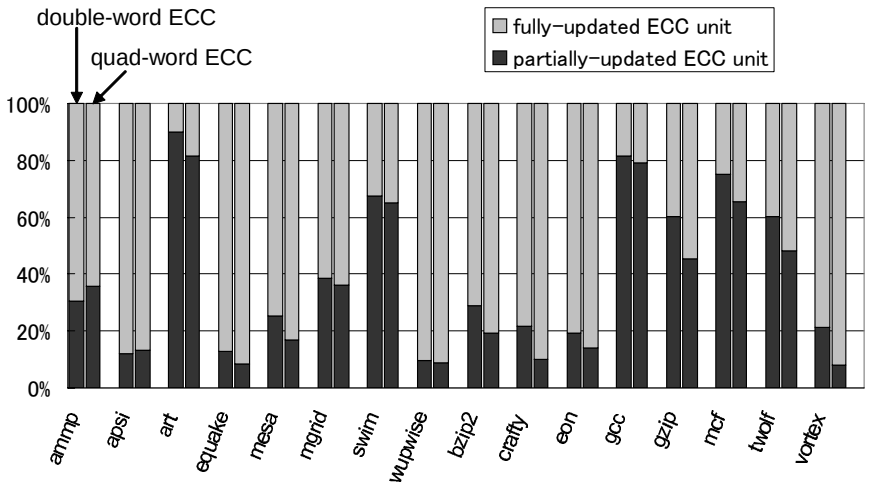


Figure 3.4: **Distribution of partially- and fully-modified ECC codewords.** Applications such as *art*, *swim*, *gcc*, *gzip*, *mcf*, *twolf* have large fractions of partially-modified codewords that require expensive read-after-write operations.

the old values from the L2 cache cannot be hidden.

The old words can be supplied by the L1 cache instead of being supplied from the L2 cache. Before writing a word, the L1 cache must probe its tag to determine whether it is holding the word or not. The L1 cache is adapted so that data access occurs in parallel with tag probing. If the write hits in the L1 cache, the old value of the data word is accessible and passed to the vertical parity update unit of the L2 cache. This scheme does not incur additional latency since tag probing usually takes more time than data access [56]. Reading the data word from a small L1 cache consumes less power than reading from a large L2 cache. The modified data path is shown in Figure 3.5.

Let us consider the case in which a write miss occurs in the L1 cache. If the L1 cache adopts a *write-allocate* policy [57], L1 cache will fetch the missed cache line from the L2 cache. The old value of the data word can be obtained from the cache line retrieved from the L2 cache. However, while write-back caches may adopt *write-allocate* policy, write-through caches usually employ a *no-write-allocate* policy: The missed line is not allocated in the L1 cache.

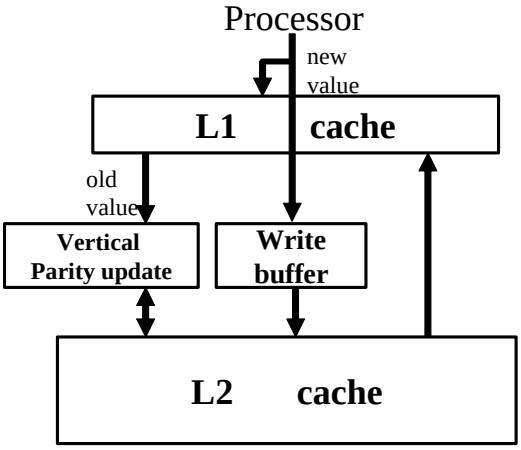


Figure 3.5: **Modified data path of L2 cache with write-through L1 cache.** The L1 cache provides the old value of the being-updated word to the vertical parity update unit of the L2 cache.

When a write miss occurs in the no-write-allocate L1 cache, the old value of the updated word is explicitly supplied by the L2 cache to the vertical parity update unit.

Figure 3.6 shows the write miss rate for a 16-KB, 4-way set associative, non-write-allocate L1 data cache (please consult Table 3.6.1 for the details of the configurations of the processor). Since the majority of writes hit in the L1 cache, the frequency of accesses to the L2 cache to obtain the old values is low.

The write buffer is also modified. Figure 3.7 shows an entry of the modified write buffer. The v bit, if set, signifies that the entry is valid and it is holding the words addressed by the *addr* field. The v_i bit indicates whether the i-th word is valid or not. A new bit h bit is added to each word. The h_i bit, if set, indicates that the i-th word did hit in the L1 cache and that the vertical parity update unit has already been updated with the old value of the word. When an entry retires, the valid words are written back to the L2 cache. For those valid words for which h bits are not set, the old values must be explicitly read from L2 and passed to the vertical parity update unit. When an error is detected in the L2 cache, the write buffer must write back all valid entries to the L2 cache before the error recovery can take place.

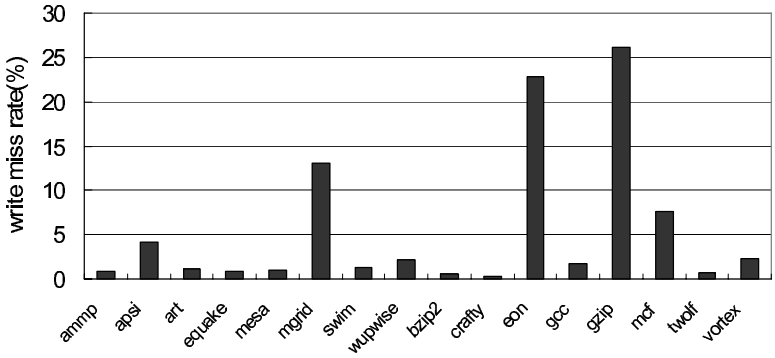


Figure 3.6: **Write miss rate of L1 data cache.** Since most writes hit in the L1 cache, the the frequency of accesses to the L2 cache to obtain the old values of updated words is low.

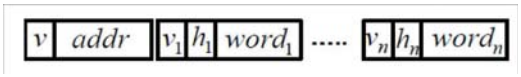


Figure 3.7: **An entry of write buffer.** An h-bit is added to each word of a cache line to remember whether the word has been hit in L1 cache or not.

3.6 Evaluation

This section evaluates the application of Zigzag-HVP to L2 caches with write-through L1 caches. Compared to L1 caches, L2 caches are typically much larger and the costs of implementing protection against soft errors are also higher; therefore, the reduction of such costs is preferred. L2 caches also present other interesting considerations due to the presence of write buffers.

3.6.1 Evaluation Methodology

The L2 cache used in the evaluation is a unified, 512 KB, 64 B-line, four-way set associative cache. The L2 cache is accompanied by a 16 KB, write-through L1 data cache, and an eight-entry write buffer. The write buffer attempts to retire the oldest entry whenever more than six entries are occupied (*retire-at-6* policy [63]). The detailed configuration of the evaluated architecture is listed in Table 3.6.1.

While 64-bit processors are increasingly gaining popularity, there are still many 32-bit processors and software built on 32-bit architectures in use in practice. Even in the 64-bit processors, support for efficient execution of 32-bit software is essential. For instance, the L2 caches in Itanium processors maintain ECC per 32-bit data to accommodate frequent 32-bit data updates encountered when executing 32-bit software [50]. A 32-bit word size is assumed in the evaluation.

Four caches with different error protection schemes are assumed in the evaluation:

- **NOPRT:** L2 cache without any soft error protection.
- **ECCSW:** L2 cache in which both the tag and data portions are protected by SECDED per single-word. Data words in the same row are interleaved to tolerate horizontal MBE.
- **ECCQW:** L2 cache in which both the tag and data portions are protected SECDED per quad-word. Data words on the same row are interleaved to tolerate horizontal MBE.
- **ZHVP:** L2 cache in which the tag and data portions are protected by Zigzag-HVP.

Table 3.2: Parameters of Simulated Architecture.

Processor Parameters	
Frequency	1 GHz
Functional Units	4 integer ALUs, 4 FP ALUs
	1 integer multiplier/divider
	1 FP multiplier/divider
LSQ size	16 instructions
RUU size	32 instructions
Issue Width	4 instructions/cycle
Memory Hierarchy Parameters	
L1 i-cache	16 KB, direct-map, 32 B block
	1 cycle latency
L1 d-cache	16 KB, 4-way, 32 B block
	1 cycle latency
	write-through & no-write-allocate
Write buffer	eight 32 B entries
	retired-at-6
L2	512 KB, unified, 4-way
	64 B block
	6 cycle latency
	write-back
Memory	100 cycle latency

We evaluate the unrecoverable error rate, number of check bits, and power consumption. Cache access activity, which is required for calculating the error rate and power consumption, is collected from cycle-accurate processor simulation using SimpleScalar toolset [64]. SPEC2000 benchmarks are used in the evaluation. Each benchmark is run for four billion instructions. Cacti tool [56] is used to determine the physical configurations of the L2 caches.

3.6.2 Physical Configurations of L2 caches

Soft error experiments with SRAM fabricated in 130 nm and 90 nm processes confirmed the existence of up to four bit MBEs [38]. Implementation of the protection schemes to tolerate spatial MBEs of up to four-bit is considered in the evaluation.

The L2 cache contains 2048 sets. Each set has four 64-B cache lines. Each tag entry has 19 bits (15 tag bits and four status bits). Cacti tool suggests that the tag portion should be divided into 256 rows, each row containing 32 tags from eight sets. Interleaving the tags of different sets in the same row together tolerates horizontal MBE while allowing four tags of the same set to be read simultaneously in a cache access. Each tag is an ECC unit in ECCSW and ECCQW. For ZHVP, the tag portion consists of 32 parity domains and each domain is a 256×19 bit array.

Cacti similarly suggests the data portion to be an array of 2048 rows; each row holds four cache lines of the same set. The tag portion and the data portion of a L2 cache are accessed sequentially to attain low power consumption. Thus, while four tags of the same set are required to be read simultaneously, only the data for the hitting line are read from the data portion. Therefore, four lines of the same set can be interleaved to tolerate MBEs without compromising the latency of the line-based accesses. In ECCSW, each word in the data portion is an ECC unit. The ECC unit in ECCQW is comprised of four consecutive words of the same line. For ZHVP, the data portion consists of 64 parity domains; each domain is a 2048×32 bit array.

3.6.3 Area Overhead

Table 3.3 lists the overhead in terms of the number of check bits required to implement the protection schemes. For ECCSW, each tag requires six check bits and each data word requires seven check bits, resulting in 944 Kb of check bits in total, or an 22.22% overhead. For ECCQW, each tag requires

Table 3.3: **Area overheads of protection schemes.** ZHVP incurs lower overhead than ECC-based schemes.

Protection Scheme	**Number of check bits (Kb)**	**Overhead (%)**
NOPRT	0	0
ECCSW	944	22.2
ECCQW	336	7.9
ZHVP	138.6	3.3

six check bits and nine check bits are required for every four words, resulting in 336 Kb of check bits in total, or an 7.91% overhead.

In ZHVP, the tag and data portions respectively require 8.6 and 130 Kb of check bits or an 3.26% overhead in total. The overhead of ZHVP is definitively smaller than those for ECCSW and ECCWQ.

We implemented a BIST similar to the one described in [54]. It is synthesized using the Hitachi 0.18 μm process. The original BIST occupies 0.28% the area of the L2 cache. The BIST is then extended to support the error recovery function. The modified BIST occupies 0.35% the area of the cache. Therefore, the cost of implementing error recovery is very small.

3.6.4 Power Overhead

When implementing Zigzag-HVP to the L2 cache, the L1 data cache needs to supply the old values to the vertical parity update unit of the L2 cache. This increases the power consumption of the L1 cache. Power consumed by both the L1 data cache and the L2 cache are taken into account in the evaluation.

The power consumption of individual accesses to the L1 and L2 caches are computed with Cacti. We modified Cacti to allow the power consumption to be computed based on the granularity of the accessed data. We implemented 32-bit SECDED, 128-bit SECDED, and horizontal vertical parity calculation circuits in 0.18 μm process and then used Synopsis NanoSim to calculate their power consumption.

Figure 3.8 breaks down the power consumed in the L1 data cache and L2 cache for the benchmarks. ECCSW increases the power consumption in the L2 cache by 28% on average, mainly due to the power consumed by

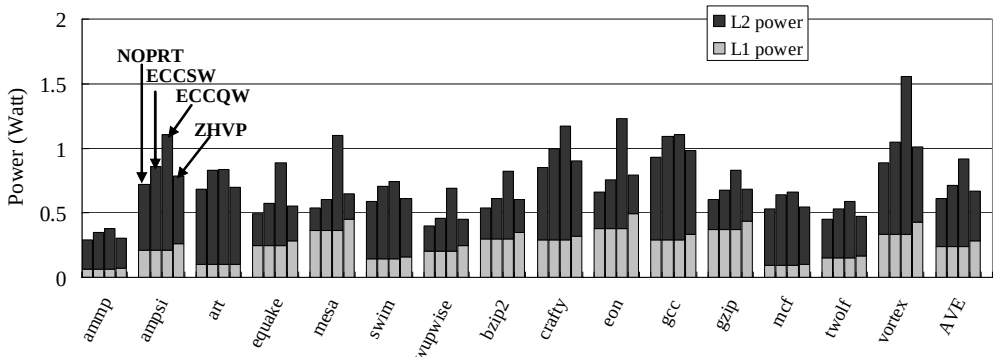


Figure 3.8: **Breakdown of power consumption.** ECCQW incurs a large power overhead due to read-modify-write operations. ZHVP consumes less power than ECCSW on average.

accessing the check bits in the tag and data portions. ECCQW increases the power consumption in the L2 cache by 80%, of which 54% is consumed by reading partially-modified quad-words and the remaining 26% is consumed by accessing and computing the check bits. The power consumption of the L1 cache remains unchanged for ECCSW and ECCQW. For ZHVP, reading the old values the updated data words increases the power consumption of the L1 cache by 18%. We also confirmed that 94.4% of writes hit in L1 cache. Additional access to the L2 cache to obtain data words missed in the L1 cache and parity calculation together increase the power consumption of the L2 cache by 4%.

When both the L1 cache and L2 cache are taken into account, ECCSW, ECCQW, and ZHVP respectively increase the total power by 17%, 49%, and 10%. ZHVP therefore consumes less power than ECC-based schemes.

3.6.5 Unrecoverable Soft Error Rate

We assume that the soft error rate (*SER*) of an unprotected SRAM equal to 1.6 KFIT[1] per megabit [4] and soft errors follow uniform distribution. We will now describe the mechanism for calculating the unrecoverable soft error rates (*URSER*) of the L2 caches. The unrecoverable errors include those

[1]One FIT (Failure In Time) corresponds to one failure per 10^9 hours

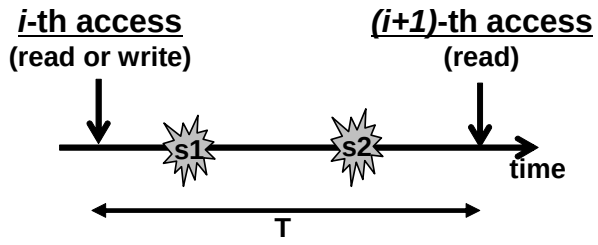


Figure 3.9: **Illustration of unrecoverable error.** Two strikes occurring between consequent accesses to the same data unit results in unrecoverable error.

detected but unrecoverable errors and those undetected errors.

Any error in NOPRT results is unrecoverable; thus the *URSER* in this case is equal to $SER \times cachesize$. Unrecoverable errors in caches other than NOPRT result from errors accumulated from multiple strikes. Figure 3.9 shows two successive accesses to the same data unit. The second access is a read access, following the first after a delay of T. Two strikes occurring in the interval between the two accesses result in unrecoverable errors in the data unit, and such a possibility is equal to $P_1(T) \times P_2(T)$, where $P_1(T)$, $P_2(T)$ are the probabilities of the two strikes occurring in a time interval T. For ECCSW, the data unit in consideration is a data word, and $P_1(T)$, and $P_2(T)$ are the probabilities of strikes occurring in the same data word ($P_1(T) = P_2(T) = SER \times T \times wordsize$). Similarly, the data unit is a quad-word, and $P_1(T)$ and $P_2(T)$ are equal to $SER \times T \times quadwordsize$ for ECCQW. For ZHVP, the data unit is a single word. Unrecoverable errors result if one strike occurs in the data word ($P_1(T) = SER \times T \times wordsize$), and the other strike occurs in the same parity domain with the data word in consideration ($P_2(T) = SER \times T \times paritydomainsize$ where $paritydomainsize$ is the number of bits in a parity domain).

Cache access activity for each data unit collected from the cycle-accurate processor simulation allows us to compute the *URSER* of each data unit. The $USRER$ of the entire cache is the sum of the *URSER* of all individual data units.

Table 3.4 lists the *URSER* of the L2 caches, averaged for all benchmarks. NOPRT's *URSER* is 6.63 KFIT, or equivalent to roughly one error in 17 years. Such an *URSER* would be unacceptable since a parallel system con-

Table 3.4: **Unrecoverable Soft Error Rate.**

Protection Scheme	*URSER*(KFIT)
NOPRT	6.63
ECCSW	$1.72 * 10^{-16}$
ECCQW	$5.85 * 10^{-16}$
ZHVP	$2.37 * 10^{-13}$

sisting of 1024 processor nodes must fail for every week. Protection schemes other than NOPRT achieved great reductions in *URSER*. While ECCSW and ECCQW have lower *URSER*, the level achieved by ZHVP is clearly sufficient for all practical purpose. More specifically, ZHVP's *URSER* is equivalent to one failure in one million products in about every 500 million years. [2].

3.7 Summary

VLSI caches must employ measures against soft errors to provide high reliability. Maintaining ECC on a per-word basis, which is preferred for word-based accessed caches, is expensive. This chapter presents Zigzag-HVP, an alternative technique to SECDED ECC to detect and correct the soft errors in such caches. The technique utilizes the concept of horizontal-vertical parity. Two-dimension interleaving of the data words converts a spatial MBE to multiple SBEs, each of which can be successfully detected and corrected within the capability provided by horizontal-vertical parity. Modifications to the cache data paths, write buffer, and BIST allow the parity update and error recovery to be performed efficiently. Implementation of Zigzag-HVP in a 512 KB L2 cache indicates that the overheads in terms of area and power consumption are respectively 3.3% and 10%, and are smaller than those of the ECC-based ones. While Zigzag-HVP is vulnerable to MBEs accumulated from multiple particle strikes, our results show that such a probability is extremely small.

[2]For reference, IBM has targeted undetected error corrupting rate of 114 FIT (one error per 1000 years) for high reliable systems [65]

Chapter 4

STCAM: Soft-Error Tolerant Content-Addressable Memory

4.1 Random Access Memory and Content Addressable Memory

Memories can be classified into two types based on their access mechanisms: Random Access Memory (RAM) and Content Addressable Memory (CAM). In a RAM access, user supplies a memory address and the RAM returns the data stored at that address. On the other hand, in a CAM access, user supplies the data and the CAM searches its entire memory to see if that data are stored any where in it. If the data were found, the search is recognized as a *hit*. In this case, the CAM returns the address where the data were found. Otherwise, if the data were not found, the CAM reports a search *miss*.

Figure 4.1-a shows the basic circuit structure of a RAM. Given an input address, the decoder decodes the address and asserts a horizontal word line. The contents of the cells on the asserted row are read through pairs of vertical bit lines. Since the data are *explicitly* read from the data array and available for examination in a RAM access, mitigating soft errors in a RAM is fairly easy. When a RAM is going to store data, it generates check bits from the data. The check bits are then stored in RAM along with the data. When the data are accessed, the check bits are also read. New check bits are then generated from the data. The new check bits are compared with the check bits previously stored in RAM. If the data have been corrupted by a soft error, the presence of the error will be recognized through a mismatch between the

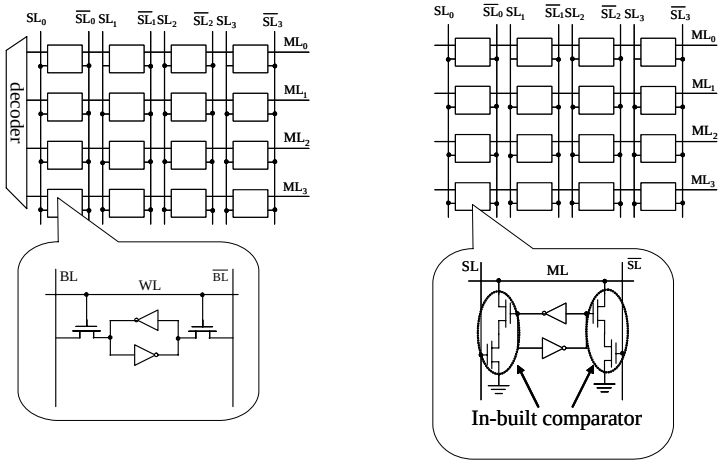


Figure 4.1: **Structures of RAM and CAM.** Structure of a RAM in (a) and structure of a CAM in (b). Each CAM cell is equipped with an in-built comparator.

two kinds of check bits.

Figure 4.1-b shows the basic circuit structure of a CAM. At the beginning of a CAM search, all the horizontal matchlines are precharged to a high voltage. The input data are then fed through the pairs of vertical search lines. Each CAM cell, equipped with an in-built comparator, compares its content with the input bit. Any mismatch between an input bit and the data bit stored in a cell on a row discharges the matchline of that row from the high voltage to zero voltage. After the search, any matchline remaining at high voltage indicates a search hit. Otherwise, if all matchlines are discharged, there is no row matching with the input; the search results in a miss. CAM effectively performs a table lookup operation, which speeds up variety of lookup-intensive applications. Contrary to a RAM access, the data stored in the CAM cells are not explicitly read out of the array in a CAM access; only hit/miss information is available after the search. This very difference in the access mechanisms leads to difference in soft error mitigation in RAMs and CAMs. The techniques used to mitigate soft errors in RAM cannot be directly applicable to CAM. Mitigating soft errors in CAMs is more challenging and has usually been ignored so far.

In practice, CAMs are usually not used stand-alone but instead a CAM is associated with a RAM, forming a CAM-RAM structure, as shown in Figure 4.2. Pairs of address and data are usually stored in a CAM-RAM structure: the addresses are stored in CAM and data are stored in RAM. Given an input address, associative search is performed in CAM. In case of a search hit, the hit match line is used to drive the corresponding word line in the RAM, from which the data are read. There are several components in a microprocessor having such a CAM-RAM structure: instruction and data caches, Translation Look-aside Buffer (TLB), load-store queue (LSQ). These components play important roles in the functioning of a processor. Making these components being tolerable against soft errors is important for achieving high processor's reliability.

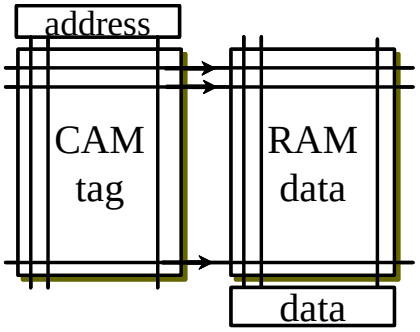


Figure 4.2: **A CAM-RAM structure.** Associative search is performed in CAM. Data for matched entry are read from RAM.

Our research concentrates on mitigating soft errors in CAMs used as tag portions of instruction and data caches in a processor. CAMs in instruction and data caches are bigger and hold more entries than those CAMs found in other components (e.g., TLB, load-store queue), so they are more vulnerable to data integrity problems caused by soft errors. However, the proposed technique is also applicable to other CAM-RAM components in a processor.

4.2 CAM-RAM Caches

In typical implementation of a memory cache, data portion is implemented using a RAM while tag portion can be implemented using either a RAM or

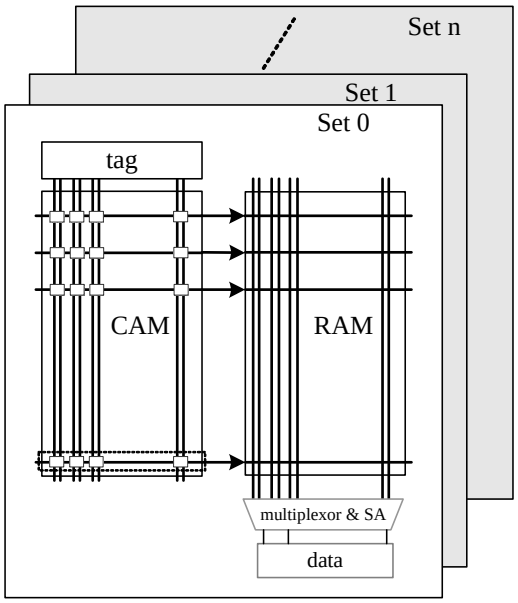


Figure 4.3: **Structure of highly associative CAM-RAM cache.** Cache lines belonging to the same set forms a CAM-RAM macro. Only one macro is activated in an access.

a CAM. We refer to a cache with a RAM tag as a RAM-RAM cache, and a cache with a CAM tag as a CAM-RAM cache.

Figure 4.3 shows the basic structure of a CAM-RAM cache. The cache consists of several CAM-RAM macros. Each CAM-RAM macro corresponds to a cache set. Each row in a CAM-RAM macro holds the tag and data of a cache line. The number of rows in a CAM-RAM macro therefore represent the number of cache lines in a set (i.e., the cache associativity). Upon a cache access, some low order bits of the input address are decoded to select a CAM-RAM macro. The remaining bits are used as input tag. Associative search for the input tag is performed on the selected CAM-RAM macro. If the search results in a hit, the data of the hit cache line is read from the RAM. Such a CAM-RAM structure allows the realization of highly associative caches. 32- or 64-way set associative CAM-RAM caches can be found in practice [66] [66].

There are several advantages with such highly associative CAM-RAM caches. First, a highly associative cache reduces conflict misses and improves processor performance. Second, fine granularity of cache *lock-down* is pos-

sible [67]. In cache lock-down, a portion of cache is reserved exclusively for storing instructions or data of critical routines. By cache lock-down, the non-determinism of processor performance caused by unpredictable cache misses can be avoided when the routines are executed. Cache lock-down is an useful feature for systems with high real-time requirement. With a 4-way set-associative cache, for example, it is practical to lock down a quarter of cache (1-way), half of the cache (2-way), or three-quarters of the cache (3-way). This is a coarse granularity and is inefficient if, for example, all that need be lock down is memory to hold data for a small interrupt handler. With a 32-way CAM-RAM cache, for instance, cache lock-down can be done in units of 1/32 of the cache, which is a much finer granularity. Third, since only one CAM-RAM macro is activated for each access and other macros are clock gated, CAM-RAM cache can be power-efficient. Previous work claimed that a CAM-RAM cache consumes less power than a RAM-RAM cache [68] [69] [70] [71]. These advantages make CAM-RAM cache a valuable design choice.

4.3 False Hits and False Misses

Data corruption by soft errors in the tag portion of a cache raises the possibilities of *false hits* and *false misses*. This section explains how false hits and false misses happen as well as the data integrity problems caused by them.

A false hit refers to a search hit that would have been a miss if soft errors had not occurred. Figure 4.4 illustrates an example of a false hit. The tag portion initially held two tag entries $\langle 1010 \rangle$ and $\langle 1000 \rangle$. A particle then struck and corrupted a bit in the first entry, making its value change from $\langle 1010 \rangle$ to $\langle 1000 \rangle$. Under that situation, if the tag $\langle 1000 \rangle$ is inputted for searching, then the search will result in a hit in the corrupted entry. Such a hit is a false hit. A false hit in the tag portion leads the processor to load data from or store data to the incorrectly matched location. Soft errors may also result in a *multi-hit*—a search hits in multiple entries.

A false miss refers to a miss that would have been a hit if the soft error had not occurred. Referring back to Figure 4.4, if the tag $\langle 1010 \rangle$ is inputted for searching, the search would cause a mismatch for the first entry which value was originally $\langle 1010 \rangle$ but now is $\langle 1000 \rangle$ due to the soft error. A false miss causes data integrity problems if the would-have-hit entry holds dirty data. The false miss in this case will trigger fetch and use of the stale data

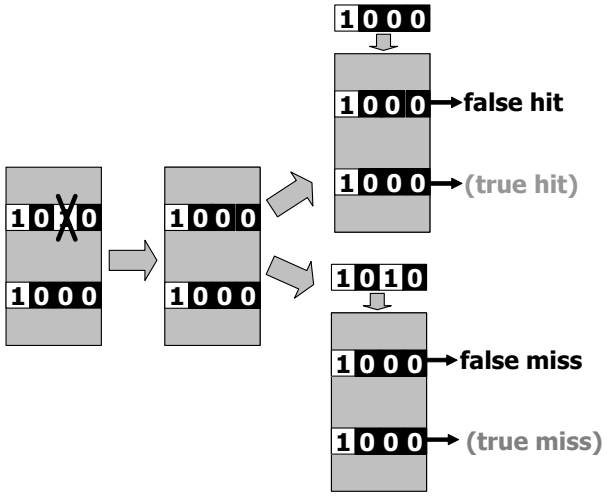


Figure 4.4: **Examples of false hits and false misses.**

from lower level caches. Since write-back caches can hold dirty data, such caches are susceptible to the data integrity problem caused by false misses.

There are several "*naive*" methods to deal with false misses and false hits in CAM tag of a CAM-RAM cache. First, false misses/false hits can be detected by replicating the CAM tag. Tag search is performed in both CAMs. The search results including the hit/miss information as well as hit addresses are compared. A mismatch in the search results indicates the presence of a soft error in one of the CAMs. While the concept is simple, tag replication incurs large area overhead. A CAM cell typically has much larger size than a RAM cell. For instance, a CAM cell occupies an area approximately four times larger than that of a RAM cell [69]. Moreover, while tag replication can detect errors, a coding technique must be employed to determine which one of the two CAMs is not corrupted.

Second, false hits and false misses can be reduced through cache scrubbing [29]. In cache scrubbing, the content of the CAM tag is periodically read and check for data integrity. Errors can be detected/corrected before showing up as false hits or false misses later. Data integrity problem is still possible with cache scrubbing if the data are corrupted and accessed in a scrubbing interval. Such a probability can be high for frequently accessed L1 caches [29]. Performing scrubbing more frequently while helps reduce the probability of false hits and false misses but increases scrubbing overheads.

Third, false misses in CAM tag can be made harmless by making the cache a write-through cache. With a write-through cache, the lower level cache is always updated with the up-to-date data. A false miss will the fetch the up-to-date data from lower level cache. However, compared with a writeback cache, write-through cache increases the traffic to lower level cache and therefore has impacts on performance and power consumption. Moreover, while false misses can be eliminated with write-through cache, we still need to deal with false hits.

The naive methods described above have their limitations. To our knowledge, there has been no cost-effective technique to deal with false hit and false miss problems in the CAM tag of a CAM-RAM cache so far. Our proposed STCAM architecture allows the CAM tag to be tolerable against soft errors.

4.4 STCAM Architecture

STCAM provides tolerability against soft errors in the CAM portion of a CAM-RAM cache. In this section, we first explain the mechanisms to deal with the false hits and false misses. We then describe the access algorithm of a STCAM cache.

4.4.1 Mitigation of False Hits

STCAM deals with false hits by storing the check bits for each CAM tag entry into the corresponding data entry in the RAM, which is shown in Figure 4.5. When a search for a given input tag results in a hit, the corresponding check bits of the hit CAM tag are read from RAM along with the data. The check bits are compared with those check bits directly computed from the hit input tag. Mismatch in the check bits signifies that the hit tag has been corrupted and the hit is indeed a false hit. Otherwise, it is a true hit.

Using parity as check bits for tags can detect the false hits. In order to be fully recoverable from false hits, error correcting code must be used.

4.4.2 Mitigation of False Misses

STCAM divides the CAM in each CAM-RAM macro into two sub-CAMs, as shown in Figure 4.6. A tag is divided into two subtags and each subtag

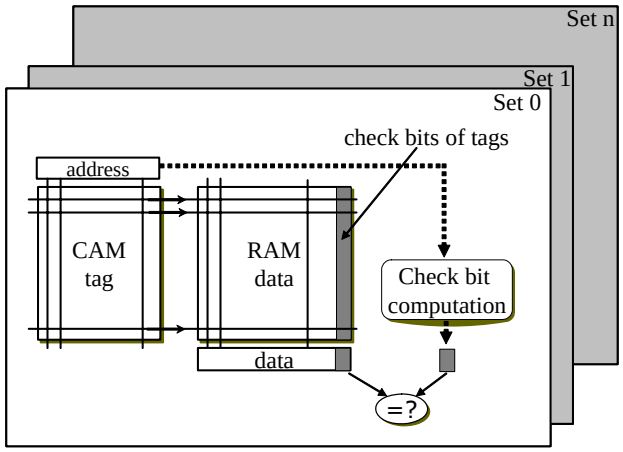


Figure 4.5: **Mechanism for mitigating false hits.** Check bits of CAM tags are stored in RAM. Tag is checked on a cache hit.

is stored in a sub-CAM. The matchlines of the sub-CAMs are called *local match lines*. Pair of local match lines belonging to the same tag are ANDed to create a *global match line*. The OR logic of the two local match lines, called *close-hit line*, is also provided.

The global match lines have the same function as the match line of a normal CAM. A global match line staying high after a search indicates a hit. If the global match line of a row is discharged while the close-hit line of the same row stays high, then the case is called a *close hit*. The close hit indicates that the tag stored in that row is partially matched with the input tag. Since soft errors are infrequent events and a soft error effects only one or a very few bits located closely, the separation of the sub-CAMs makes the probability that both subtags of the same tag be corrupted very low. A false miss therefore will show up as a close hit in STCAM that can be recognized by an asserted close-hit line.

When a close hit is encountered, the tag with the close hit line asserted is explicitly read from the CAM and its check bits are read from RAM. Verification are performed to determine whether the tag has been corrupted or not. The corrupted tag is corrected and compared with the input tag to determine whether the close hit is a false miss or a true miss.

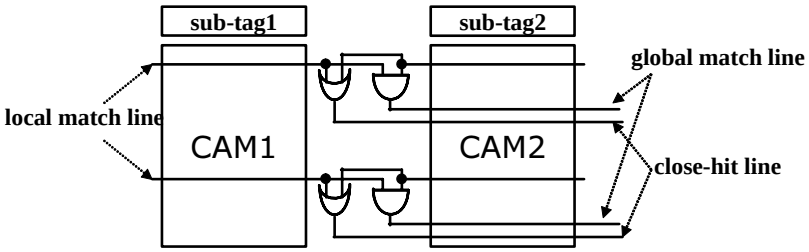


Figure 4.6: **Modified CAM tag for mitigating false misses.** CAM tag is divided in horizontal direction into two sub-CAM. A back-up checking is required for a partially-matching case which is signified by an asserted close-hit line.

4.4.3 Cache Access Algorithm

Figure 4.7 indicates the access algorithm of STCAM. Given an input address, the set index portion and the tag portion are derived. The set index portion is used to select a CAM-RAM macro. The tag portion of the address is divided into two subtags. The subtags are fed into the sub-CAMs of the selected macro. The sub-CAMs then perform associative search in parallel. A global match line staying asserted after the search indicates a cache hit. In this case, the check bits of the hit tag are also read from RAM together with the data word. Check is performed to verify whether the hit is really a true hit or a false hit.

In the case of a cache miss (e.g., there is no global match line staying high after the search), if there is no close-hit line asserted, then the miss is a true miss. Otherwise, if there is a close hit, the close hit tag are explicitly read from CAM and its check bits are read from RAM. The check bits recomputed from the tag read from CAM are compared with the check bits previously stored in RAM. The close hit is a false miss if the tag is found out to be corrupted and the corrected tag is identical to the input tag. Otherwise, the close hit is considered a true miss.

It is possible that a search results in hits or close hits of multiple tag entries. In this case, the checking routine is repeated for each of those entries.

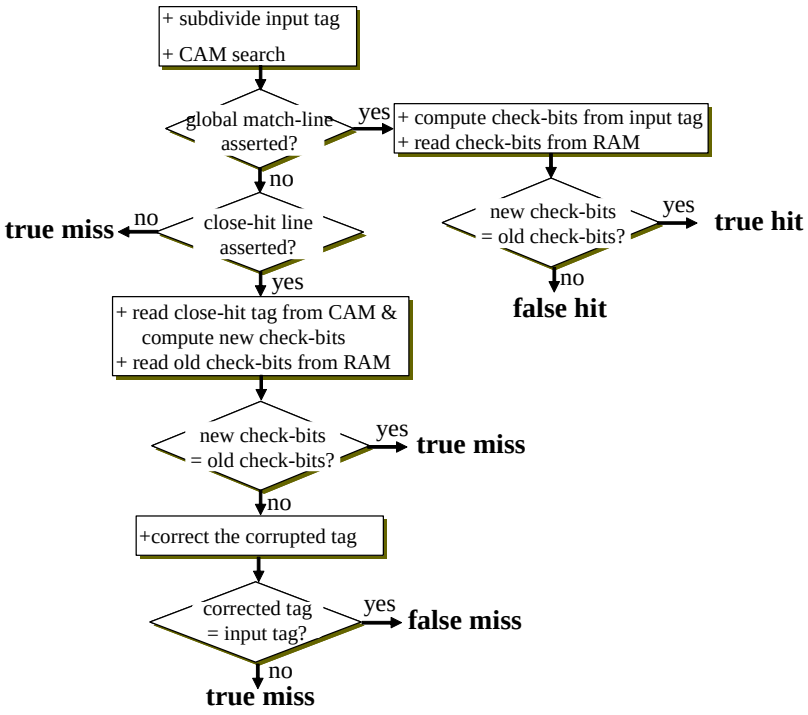


Figure 4.7: **Access algorithm of a STCAM cache.**

4.5 Close Hit Rate and Tag Encoding Scheme

There are two causes of close hits: soft errors and accidental matching of subtags. A soft error occurs and corrupts a tag so that an access that would have been a true hit if the soft error had not happened becomes a close hit. A close hit also results if an input subtag is accidentally matched with a subtag stored in the CAM. Since soft errors are very infrequent events, accidental matching of tags is the main cause of close hits. Since verifying if the close hit is a false miss or a true miss requires the tag to be explicitly read from the sub-CAMs, overheads in term of access latency and power consumption incur. This section first discusses how often the close hits may occur due to accidental matching of subtags. An encoding scheme for reducing the frequency of close hits is then proposed.

4.5.1 Close Hit Rate

If we assume that the values of input tags as well as the tags stored in the tag portion are *random*, the rate of close hits caused by accidental matching of subtags, R_{ch}, can be roughly estimated by Equation 4.1.

$$R_{ch} = R_m * \frac{1}{2^{T/2}} * W * 2 \tag{4.1}$$

Here, R_m is the cache miss rate, W is the cache associativity, and T is the length of a tag in bit.

Close hits are considered only on cache misses (no global match line asserted), hence the multiplication of R_m in the right hand side (RHS) of the equation. The second term in RHS is the probability that a two randomly chosen $\frac{T}{2}$-bit subtags are coincident. A input subtag needs to be compared with W subtags belonging to the same set, hence the multiplication of W. Finally, close hit can be triggered by the matching of either of the two input subtags, hence the last term in the RHS.

Let us consider a concrete example. Here we assume a 32KB, 32-way, 32B line cache which is similar to the data cache found in Intel XScale processor [72]. With 32-bit address, five bits are used for indexing bytes inside a cache line, another five bits are used for set indexing (the cache have 32 sets), leaving the remaining 22 bits used for tag. Substituting these values (W=32, T=22) into Equation 4.1, we have $R_{ch} = R_m/32$. That means that a close hit

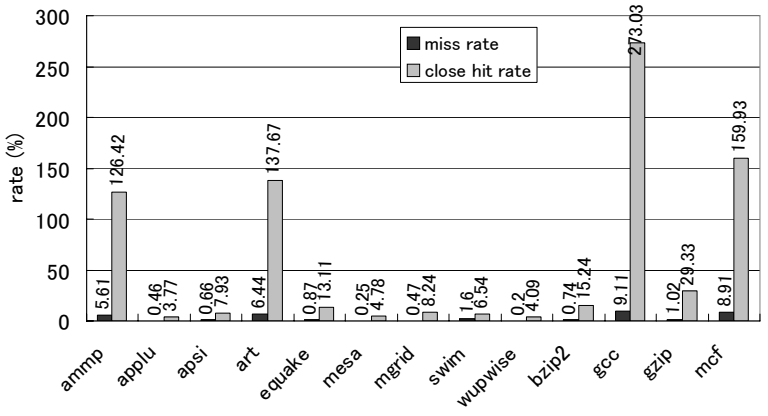


Figure 4.8: **Cache miss rate and close hit rate.** Close hit rate of the data cache is many times higher than the miss rate.

is estimated to occur once for every 32 cache misses on average. With such low close hit rate, the overhead of checking for a false miss can be tolerable.

We perform simulation to verity the estimated close hit rate with real-world close hit rate. Figure 4.8 shows the miss rate, close hit rate obtained when SPEC2000 benchmarks are executed with the above-mentioned 32KB data cache. The data cache is virtually-indexed, virtually-tagged. The cache simulation is carried out using SimpleScalar toolset [64]. The ARM binaries generated using a GCC compiler (version 2.95) are used in the evaluation. Each benchmark is run for 20 billion instructions. A tag is divided based on high and low order bits.

Contradict to our estimation ($R_{ch} = R_m/32$), the real close hit rate is prohibitively high, particularly for *ammp, art, gcc*, and *mcf*. The close hit rates many time higher than the miss rates are possible since multiple close hits may occur in a single miss. We next investigate the cause of such excessively high close hit rate.

4.5.2 Distribution of Access Addresses

The cause of high close hit rate exhibited in real applications can be understood by considering the distribution of the target addresses of the accesses to the data cache, shown in Figure 4.9. Here, the 32-bit (i.e., 4GB) address

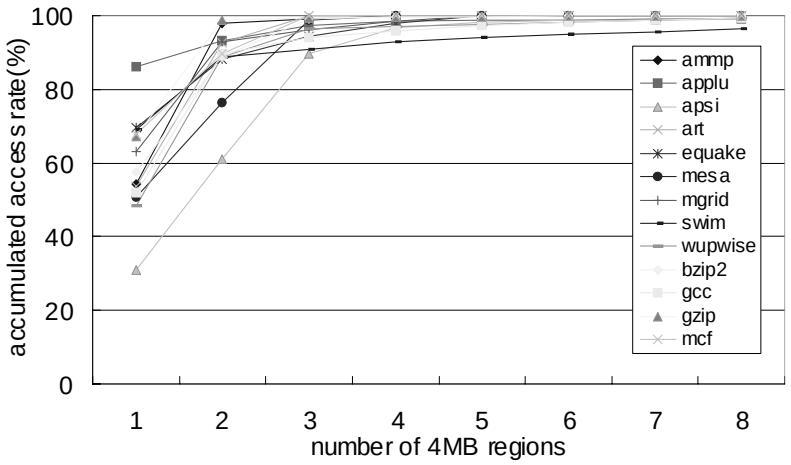


Figure 4.9: **Distribution of the target addresses of accesses.** Accesses to the data cache cluster in only a few region.

space is divided into regions of 4MB. The X-axis shows the number of regions have been most accessed, and the Y-axis shows the accumulated access rate of the selected number of regions.

Accesses cluster in a very few "hot" regions in the address space for all benchmarks. The cluster of accesses results in low diversity of the high order bits of the target addresses. If we simply divide the tags to subtags containing respectively the high and low order bits, as shown in Figure 4.10-a, many close hits will result from the coincidences of the subtags containing the high order bits.

4.5.3 Tag Encoding Scheme

We propose an original tag encoding scheme, shown in Figure 4.10-b. A tag is first divided into two parts containing respectively the high and low order bits. The part containing the low order bits is used as first subtag. The second subtag is produced by XORing the two parts together. Since the diversity of the parts containing the high order bits of the tags is low, by XORing the two parts together, we essentially impose the diversity of the low order bits to the high order bits. The probability that the parts containing the high order bits being accidentally matched is reduced, leading

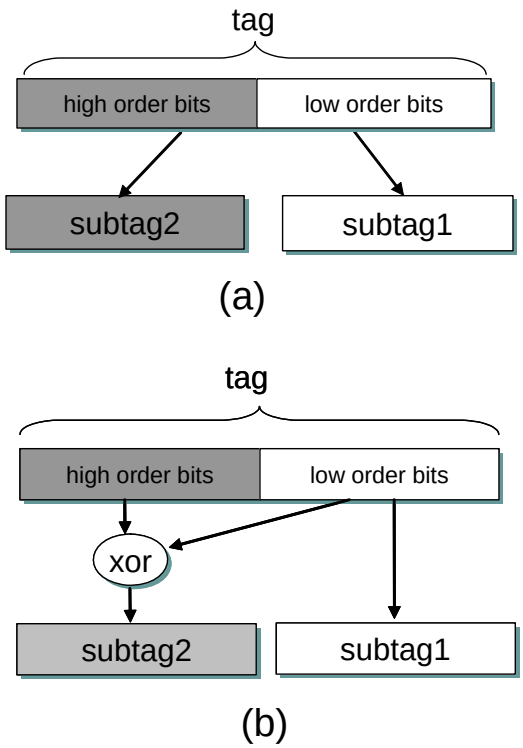


Figure 4.10: **Tag encoding schemes.** Scheme in (a) causes frequent close hits. The proposed scheme in (b) effectively reduces the close hit rate.

to reduction in the frequency of close hits. XOR operation in the proposed tag encoding scheme preserves data information. The original high order bits of a tag can be reproduced by XORing the two encoded subtags.

Figure 4.11 shows the close hit rates of the data cache employing the proposed tag encoding scheme. Cache configuration and simulation method are identical to those described in Section 4.5.1. Compared to the results shown in Figure 4.8, the close hit rates are reduced greatly for all benchmarks. The proposed tag encoding scheme is therefore very effective in reducing close hits.

Noteworthy, with the proposed tag encoding scheme, the detection (or correction) of soft errors should be based on the encoded subtags. Specifically, check bits should be directly computed from the encoded tag (e.g., consisting

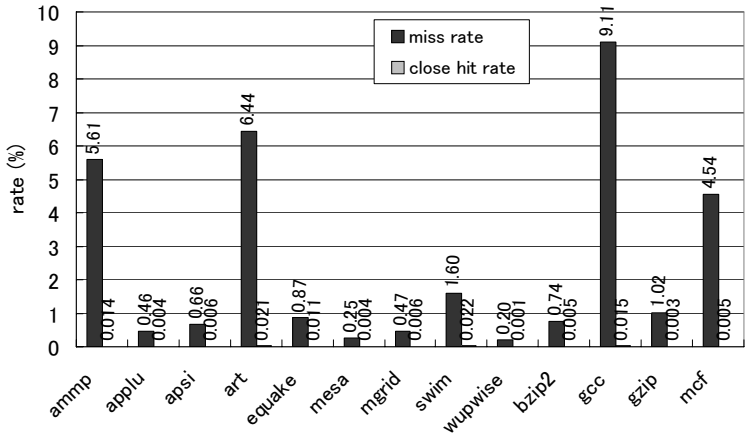


Figure 4.11: **Miss and close-hit rates with proposed tag encoding.** The close-hit rate is reduced greatly.

of *subtag1* and *subtag2* in the Figure 4.10). If the check bits are computed from the original tag otherwise, a bit corruption in *subtag1* stored in CAM would generate a another bit corruption when restoring the high order bits (by XORing *subtag1* and *subtag2*). Since SECDED ECC are usually used, the double-bit error may exceed the error detection/correction capability provided by the ECC.

4.6 Overheads of STCAM

Implementing STCAM requires modifications to the circuitry of a CAM-RAM cache. This section discusses how the modifications may impact the cache access latency. Furthermore, when a close hit occurs, additional cycles must be spent to verify whether the close hit is a true miss or a false miss. The impact of such an overhead on processor performance is also considered.

4.6.1 Cache Access Latency

The access time of the 32KB, 32-way, 32B line CAM-RAM cache is evaluated using Cacti tool [56]. Given cache's architectural parameters such as cache size, cache line size, and associativity as well as a scaling parameter repre-

senting the process technology being used, Cacti performs analytical analysis to determine the optimal cache structure that yields the best trade-offs in access latency, area, and power consumption. Since the original Cacti is only able to model a monolithic fully-associative CAM-RAM, we enhance Cacti to allow it to be able to model an CAM-RAM cache consisting of multiple CAM-RAM macros, taking into account the time to route and select a macro.

The access times of the CAM-RAM cache, before and after modification, are listed respectively in Equation 4.2 and Equation 4.3. Here, we assume 0.18 um process technology. The access time is the sum of 1) the time to route and select a CAM-RAM macro (*macro_select*), 2) time to search the CAM (*cam_search*), and 3) time to read data from RAM (*ram_read*).

$$\begin{aligned} access\_time &= macro\_select(0.21ns) \\ &\quad + cam\_search(0.66ns) \\ &\quad + ram\_read(0.51ns) \\ &= 1.38ns \end{aligned} \tag{4.2}$$

$$\begin{aligned} access\_time &= macro\_select(0.21ns) \\ &\quad + cam\_search(0.29ns) \\ &\quad + ram\_read(0.51ns) \\ &= 1.01ns \end{aligned} \tag{4.3}$$

CAM search time dominates the access time in the original cache. Since matchlines in a CAM typically have high capacitances, driving such matchlines consumes much time. Subdivision of a CAM into two sub-CAMs in the modified cache reduces the effective lengths of the matchlines and consequently reduces the time required to charge/discharge the matchlines. This results in considerable improvement in CAM search time in the STCAM(from 0.66ns to 0.29ns).

We evaluated that the time required for tag encoding is 0.067ns. Since tag encoding can perform in parallel with macro selecting and tag encoding requires less time than macro select, the latency of tag encoding is completely hidden.

The latency of computing or verifying check bits is not included in Equation 4.2 and 4.3 for several reasons. First, since a reliable cache must protect the data portion from soft errors, the overhead of computing and verifying check bits for the data already present, regardless of whether soft error

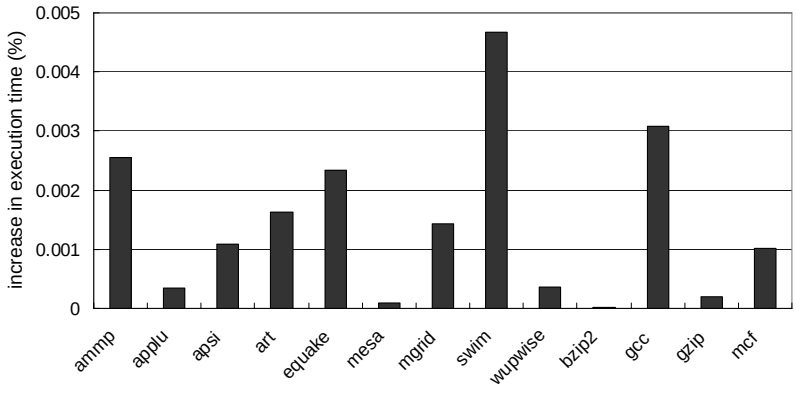


Figure 4.12: **Performance of processor using STCAM.** When STCAM is employed in processor's data cache, performance degradation incurred by close-hit checking is very small.

measures for the tag portion are employed or not. Second, computing and verifying the check bits can be removed from the critical path of the cache access. The processor can immediately use the data obtained without waiting until the checking for false hit/miss completes. If a false hit is detected later, the processor simply discards the intermediate results produced from the corrupted data.

Modifications to the cache to support our technique therefore do not increase the cache access time. For cases of close hits, additional execution cycles for checking the potential false miss are explicitly allocated and such an overhead is evaluated next.

4.6.2 Processor Performance

Our modified cache requires backup checking on a close hit to verify whether the close hit is a true miss or a false miss. Such checking needs additional execution cycles and degrades processor performance. We used an execution-driven, cycle-accurate simulator from SimpleScalar toolset [64] to measure the performance degradation of a processor using the modified data cache. We modeled a processor similar to Intel XScale processor [72]. The processor is a 5-stage, in-order scalar processor, having 32KB, 32B line, 32-way set associative instruction and data caches. Cache hit latency is 1 clock cycle. The

memory is idealized (i.e., alway hit) with access latency of 32 cycles. Each benchmark are run for 10 billion instructions. We assumed that verifying each close hit requires an additional cycle. Figure 4.12 shows the evaluation results. Execution time for all benchmarks increases no more than 0.005%. Such very small performance degradation results from very low close hit rate brought by our tag encoding scheme.

4.7 Summary

The circuit structure and access nature of content addressable memories (CAMs) make it difficult to mitigate false hits and false misses caused by soft errors. This chapter proposed STCAM, a soft-error tolerant CAM architecture. In STCAM, the check bits of the CAM portion are stored in the associated RAM portion of a CAM-RAM structure. False hits in the CAM tag can be detected by examining the check bits of the hit tags. Mitigation of false misses involves subdividing a CAM and providing backup checking for cases the input is partially matched in the CAM. An original encoding scheme is proposed to reduce the frequency of back-up checking. Evaluation results indicated that the circuit modifications to support the technique do not increase cache access latency. The performance degradation imposed by backup checking was extremely low.

Chapter 5

SEVA: Soft-Error- and Variation-Aware Cache Architecture

5.1 Introduction

SRAM designs are confronted with two serious problems: soft errors and variation-induced defects. Soft errors refer to radiation-induced transient errors [6]. Soft error rate (SER) per bit of SRAM is expected to stay steady over process generations [73]. However, device scaling allows the number of bits integrated on a chip to increase exponentially, rising the total SER rapidly. Tolerance of soft errors is highly required. Error Correcting Code (ECC)–particularly, Single Error Correction Double Error Detection Hamming code (SECDED)– is widely employed to detect and correct soft errors in SRAMs.

Process variation causes spreads in the electrical characteristics of scaled devices. The effect is pronounced in SRAMs where minimum-geometry transistors are used. The number of variation-induced defective memory cells becomes high with device scaling [46] [47], leading the conventional redundancy techniques (e.g., using redundancy rows/columns) to become impractical. Combination of a redundancy technique with ECC can tolerate a high degree of random defects [48] [49]. A block containing a single defective cell can be repaired by ECC. At a much lower probability, a block may contain multiple defective cells that exceed the error detection and/or correction ca-

pability provided by ECC. Only such a block is replaced by a redundancy element. With such a combined approach, a small number of redundancy elements can be sufficient.

It is cost-effective if the same ECC resource can be used to tolerate both defects and soft errors. However, while a defective cell in a block can be tolerated by SECDED, the block becomes vulnerable to soft errors. Soft errors occurring in the block could be left undetectable and/or uncorrectable. While the error detection and correction capability of ECC can be enhanced by using codes that are more powerful than SECDED, these codes incur significant overheads and are impractical for being implemented in high-speed SRAMs. Previous work therefore improves defect tolerance at the expense of degraded soft error tolerance [48] [49].

This chapter proposes **SEVA**, a Soft Error and Variation Aware cache architecture. SEVA exploits SECDED to tolerate variation-induced defects while preserving high resilience against soft errors. SEVA allows only the clean data to be stored in the defective (but still usable) blocks. Such a constraint is enforced through a selective write-through mechanism called *assurance update*. Soft errors cannot cause integrity problem in these blocks because SECDED is still able to detect the errors. When soft errors are detected, correct data can always be obtained from the lower levels of the cache hierarchy. Assurance updates can be effectively executed by maintaining information about *defectiveness* and *dirtiness* for each SECDED block. We also propose data swapping between blocks of the same cache line to reduce number of assurance updates. SEVA improves yield and reliability with modest costs.

The rest of this chapter is organized as follows. Section 5.2 respectively discusses the existing techniques for defects in SRAMs. Section 5.3 discusses the limitation of these existing tolerance techniques. Section 5.4 describes SEVA architecture. Section 5.5 presents defect and yield analysis for an SEVA cache. Section 5.6 presents the performance and reliability evaluations. Finally, Section 5.7 summarizes this chapter.

Table 5.1: **Variation of V_{th} over process generations**. Variation becomes severe as process scales down.

Technology (nm)	65	57	50	45	36
Vth (V)	0.18	0.17	0.16	0.15	0.14
σV_{th} (mV)	19	21	22	24	27

5.2 Variation-induced Defects and Defect Tolerance

5.2.1 Variation-induced Defects

As process scales down, precise control of various device parameters such as gate width (W), gate length (L), number and locations of dopant atoms in manufacturing becomes increasingly difficult. Variations of such parameters result in considerable variations in electrical properties of a device. Threshold voltage (V_{th}) is an important parameter that characterizes the electrical property of a transistor. Table 5.1 shows variation of V_{th} over process generations, predicted by ITRS [73]. Particularly, fluctuations in number and locations of dopant atoms in the channel region of a transistor are known as the dominant source of V_{th} variation in scaled devices [74] [75] [76]. Such variation is intrinsic and cannot be solved by lithography improvements.

Effect of variation is pronounced in SRAMs where minimum-geometry transistors are used. Figure 5.1 shows the schematic of an SRAM cell. The cell consists of two cross-coupled CMOS inverters (comprised by transistor $T3 \sim T6$) and two N-type access transistors ($T1, T2$). Large mismatches in the strengths of these transistors in an SRAM cell can make the cell fail to function. Three types of failures can occur in an SRAM cell: *read*, *write*, and *access time* failures. We now provide brief descriptions about the mechanisms of these failures.

- **Read Failure.** Read failure is defined as the flipping of cell data while reading an SRAM cell. The voltage of node B (the node storing 0 in Figure 5.1), V_B, is raised from zero to V_{read} due to a voltage divider between BL (precharged at VDD) and GND through $T1$ and $T3$. If V_{read} is larger than the tripling voltage V_{trip} of $\{T4,T6\}$ inverter, the

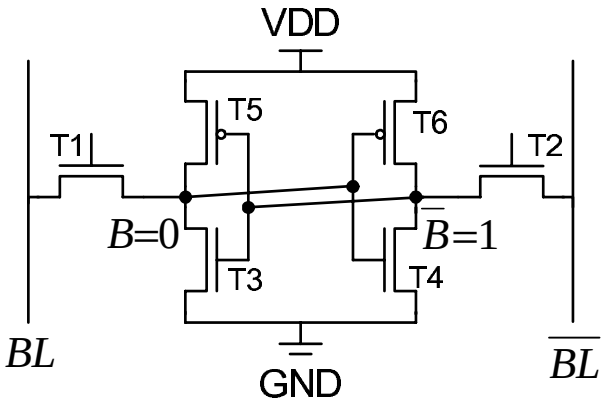


Figure 5.1: **Schematic of a SRAM cell.** The cell is consisted of six transistors. Variations in threshold voltages of these transistors can lead to a cell failure.

cell flips resulting in a read failure. Variations in V_{th} of $T1$ and $T3$ (or $T6$ and $T4$) lead to large variation in V_{div} (or V_{trip}).

- **Write Failure.** Write failure is defined as the inability to successfully write to a cell. When writing 0 to node $\overline{B}$ which originally stores 1, $V_{\overline{B}}$ develops to V_{write} due to a voltage divider between $\overline{BL}$ at GND and VDD through $T2$ and $T6$. If V_{write} is larger than V_{trip} of $\{T3,T5\}$ inverter, the write will fail. Since $T2$ and $T6$ (also $T1$ and $T5$) are typically the smallest transistors in the cell, V_{th} variations in these transistors causes large variation in V_{write}, resulting in a high probability of write failure [46].

- **Access Time Failure.** The access time (T_{access}) is defined as the time required to develop a predefined voltage difference between BL and $\overline{BL}$. When node B stores 0, BL will discharge through $T1$ and $T3$ in a read operation. The discharge speed depends on the strengths of $T1$ and $T3$. V_{th} variations in these transistors cause a spread in T_{access}. An access fails if T_{access} is larger than the maximum tolerable limit T_{limit}.

Table 5.2 shows the failure probability of a cell at 45nm process with different amount of V_{th} variations [47]. P_{RF}, P_{WF}, and P_{AF} are respectively

Table 5.2: **Probability of failure cell versus σV_{th} in 45nm process [47].**

$\sigma_{V_{th}}$(mV)	P_{RF}	P_{WF}	P_{AF}	P_{Fault}
20	$1x10^{-4}$	$1x10^{-4}$	$1x10^{-4}$	$1x10^{-4}$
30	$2x10^{-4}$	$2x10^{-4}$	$5x10^{-4}$	$8x10^{-4}$
40	$1x10^{-3}$	$6x10^{-4}$	$3x10^{-3}$	$4x10^{-3}$

the probability of read, write, and access failures. P_{Fault} is the total failure probability. The failure probability is quite high and highly sensitive to V_{th} variation.

5.2.2 Defect Tolerance Techniques

Redundancy techniques have been used to tolerate manufacturing defects in SRAMs to improve yield. After fabrication, SRAMs go through a test process. A test algorithm applies input data in particular patterns to the SRAMs, and diagnoses the receiving output data to locate the defects. Common defects in SRAMs are malfunction cells, stuck-at faults, and bridging faults. When the defects have been identified, a repair algorithm is executed to replace the defective locations with redundancy elements. Traditionally, external equipments are extensively used to test the chips, to localize the defects, and to drive a laser beam to perform repair, or to blow fuses or anti-fuses. However, external test & repair method is costly. Modern SRAMs typically includes Built-In Self Test (BIST) and Built-in Self Repair (BISR) to maintain reasonable test and repair cost [77].

Conventional redundancy techniques usually use subarrays [19], cache lines [78], rows or columns [61] as the redundancy elements. However, high defect densities in scaled processes make such coarse-grain redundancy techniques become impractical. Fine-grain redundancy techniques have also been proposed. In [47], a row can contain several cache lines and an individual cache line, rather than an entire row, can be replaced through modifications to the column multiplexers. Maintaining redundancy at a word level (e.g., 32-bit) allows more efficient utilization of redundant resources [79] [80]. Nevertheless, such fine-grain redundancy techniques are still impractical for high defect densities. For instance, assuming that the defect probability per cell is 0.001, then a 512-KB cache will have roughly 4K defective words. The BISR

Table 5.3: **Number of check bits required for different information-bit lengths.** DEC requires about two times more check bits than SECDED.

Information-bit length	32	64	128
SECDED	7	8	9
BCH DEC	12	14	16

and decoder circuitry supporting the replacement of such a high number of defective words is very complex. Particularly, the Content-Addressable Memory (CAM), which is usually used in BISR to store the addresses of the defective words that need to be rerouted to other non-defective words, becomes excessively large and degrades the access latency of the cache.

There have been proposals that combine a redundancy technique with ECC to tolerate a high number of random defects [48] [49]. ECC, usually SECDED, is maintained for each data block. In the case a defective cell is present in the block, the defective cell can be corrected by SECDED. Only in the case where more than one defective cells present in the same block, the block is replaced by a redundancy element. The probability of the later case is much lower than the former case. With such a combination approach, the number of redundancy elements can be kept at low and the BISR can be keep simple.

5.3 Limitation of Existing Techniques

Defect tolerance and soft-error tolerance are the two topics that have usually been treated separately. ECC has proved to be effective for tolerating either high defect densities or soft errors. It would be cost-effective if the same ECC resource can be used for both of these purposes. However, while a defective cell present in a block can be tolerated by SECDED, the block becomes vulnerable to soft errors. An error occurring in the block is detectable but *uncorrectable*. This can get the processor system into an unrecoverable state, particularly when the corrupted data are dirty data that have no backup elsewhere in the cache hierarchy.

Error detection/correction capability of ECC can be improved by using codes that are more powerful than SECDED. For instance, if Double Error Correction Code (DEC) is used, a defective cell present in a block can be

tolerated while a single-bit error occurring in the block can be successfully detected and corrected. Some conventional DEC codes described in the literature are Bose-Chaudhury-Hocquenghem (BCH), Reed-Solomon [81]. However, their overheads are considerably larger than those of SECDED. Table 5.3 shows the number of check bits required to implement SECDED and BCH DEC at various information bit lengths. DEC increases number of check bits significantly. Moreover, while encoding/decoding circuitry of SECDED can be constructed as XOR trees and is quite fast, the powerful codes including BCH DEC are typically cyclic codes employing multi-bit Linear Feedback Shift Registers (LFSR) which introduce inordinate delay and are unsuitable for being implemented in SRAMs requiring fast access. Even in an optimized implementation of a Reed-Solomon code, it requires tens of nanoseconds to encoder/decoder the code [82].

Therefore, it is important to use SECDED as an ECC of choice. Existing work [48] [49] that have advocated for SECDED/redundancy-combined approach have limitation that they improve defect tolerance at the expense of degraded tolerance against soft errors. SEVA can overcome such a limitation. SEVA utilizes SECDED to tolerate variation-induced defects while preserving high resilience against soft errors.

5.4 SEVA Architecture

5.4.1 Cache Structure and Block Classification

A SEVA cache consists of several subarrays. A subarray has several rows. Each row is comprised of several blocks. Each subarray has its own decoder, BIST, BISR, and some redundancy rows. BIST detects the defective cells in the subarray. The blocks are classified based on the number of defective cells they contain. A block that does not have any defective cell is called a *good* block (g-block). A block that has a defective cell is called a *tolerable* block (t-block). A block that has more than one defective cells is called a *bad* block (b-block). The row containing at least one b-block is a bad row. BISR replaces the bad rows with redundancy rows.

SEVA associates each block with a *g-bit*. The g-bit of a block is set (or reset) if the block is a g-block (or t-block). Defect analyzing and setting of the g-bits are performed by BIST every time system is booted. Alternatively, such an overhead can be reduced by storing the values of the g-bits into a

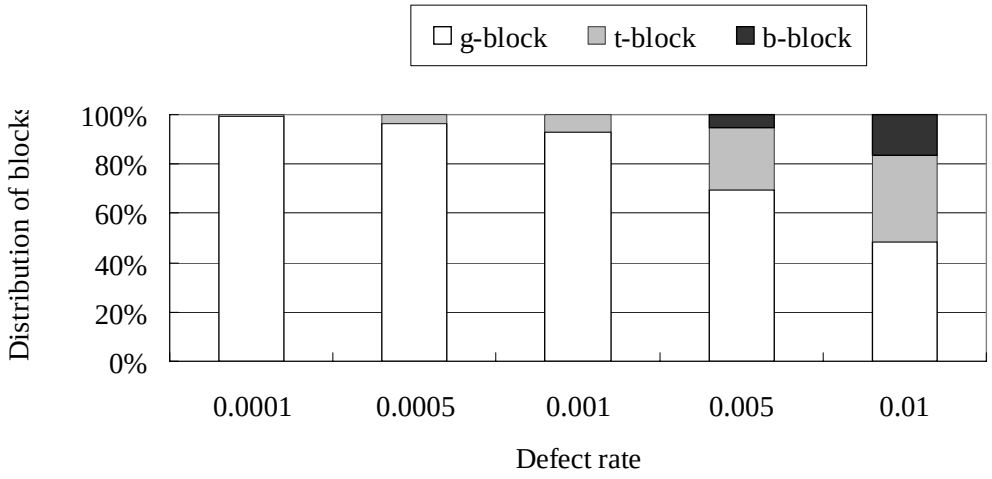


Figure 5.2: **Distribution of blocks at different defect rates.** As defect rate becomes high, there is a significant proportion of t-blocks which are susceptible to unrecoverable soft errors.

non-volatile storage and reloading these values into the g-bits at rebooting.

SEVA interleaves multiple blocks in the same row to disperse the error bits of a spatial MBE. We assume that a block contains as much as one error bit and focus on how to deal with a single-bit error in a block. If the block is a g-block, SECDED can be exclusively used for soft error tolerance. Therefore, the single-bit error is detectable and correctable. However, if the block is a t-block, a part of SECDED capability is used for repairing the defective cell, leaving the reduced capability for soft error tolerance. The error in this case is detectable, but *uncorrectable*. If the block holds the only instance of the data, soft error occurring in the block can get system into an unrecoverable state.

Figure 5.2 shows the distribution of three types of blocks at different defect rates. Section 5.5 will describe how to calculate such a distribution. The block size is 72 bits (64 information bits and 8 check bits). With a high defect rate, after the b-blocks are replaced by redundancy elements, there is a significant proportion of t-blocks and these blocks are vulnerable to data integrity problem caused by soft errors.

5.4.2 Inclusion Properties in Multi-level Cache Hierarchies

There are two factors that determine whether data corruption in a t-block can lead to an unrecoverable state or not. The first relates to the dirtiness of data—whether the corrupted data are dirty or clean. The second relates to whether or not the cache hierarchy maintains inclusion property [83].

In some processors (e.g., Intel Pentium as well as most RISCs), the data in a cache may also be in its lower level cache. In other words, data stored in a cache constitute a superset of the data stored in its lower level cache. These caches are called inclusive. Other processors (e.g., AMD Athlon) have exclusive caches; the data are guaranteed to be in at most one of a cache and its lower level cache. The major advantage of inclusive caches is that they simplify the maintenance of cache coherency in a multiprocessor system. To process a coherency request from other processors, a processor must determine whether or not it currently holds the requested cache line. In inclusive caches, examining the lowest level cache is sufficient. In an exclusive cache hierarchy, however, the upper level caches must be checked as well. Another advantage of inclusive caches is that the lower level cache can use larger cache lines, which increases the spatial locality and reduces the size of the cache tags. Exclusive caches have an advantage that they store more data. However, as processors include more and more on-chip caches and the relative difference in sizes of a cache and its lower level cache becomes larger, the merit of storage saving provided by exclusive caches becomes less significant.

There is only one instance for each piece of data stored in an exclusive cache hierarchy. If the data stored in a t-block are corrupted due to a soft error, no matter whether the data are clean or dirty, there is no way to recover the data. If such data are referenced by the processor, the system can get into an unrecoverable state. On the other hands, in an inclusive cache hierarchy, the clean data in a cache always have a copy in its lower level cache. When clean data in a t-block of a cache are corrupted due to a soft error, the correct data can be obtained from the lower level cache. Only dirty data in a t-block being corrupted can raise the possibility of an unrecoverable error.

In this research, we assume that multi-level cache hierarchy maintains inclusive property. The unrecoverable data integrity problem due to corruption of dirty data stored in t-blocks is dealt with by the assurance update mechanism described in the next section.

5.4.3 Assurance Update

The data integrity problem caused by soft errors can be prevented by allowing only clean data to be stored into t-blocks. Whenever a t-block of a cache receives an update from an upper level cache (or processor), the updated data are also sent to the lower level cache. Such a selective update is referred to as an *assurance* update. An assurance update is also necessary if the update goes to a cache line which the block storing its tag is a t-block. It is because if the tag is corrupted by a soft error, we cannot reclaim the correct address to which the cache line originally belonged.

Assurance updates increase the number of accesses to lower level cache. Conventionally, there is a writeback buffer sitting between a cache and its lower level cache. Data updated to lower level cache are temporarily stored in writeback buffer and will be written back later when the bus is free. Thanks to writeback buffer, an assurance update can be mostly removed from the critical path of a cache access. However, if writeback buffer is full, an update will be stalled for writeback buffer to retire its entries to make room for newly coming data. Stalling caused by assurance updates will degrade processor performance. Assurance updates also increase power consumption. As previously indicated in Figure 5.2, with a high defect rate the proportion of t-blocks and consequently the frequency of assurance update can be significant. The next section will describe techniques that efficiently perform assurance updates to limit their impacts on performance and power consumption.

5.4.4 Assurance Update Reduction

In this section, we are going to describe three techniques that can help reduce the frequency of assurance update.

A. Maintaining dirtiness at block level

In conventional caches, the dirtiness of data is maintained at cache line level. Since typical cache line size is 64B∼256B while SECDED block size is 32b∼256b, a cache line usually consists of multiple SECDED blocks. When a dirty cache line is written back, all the constituent blocks are written back, no matter whether the blocks have been modified or not. It is wasteful since clean blocks in a cache already have their copies in the lower level cache. By

maintaining the dirtiness of data at the block level rather than at cache line level, unnecessary writebacks of clean blocks can be omitted.

Each block is associated with a *d-bit*. The d-bit of a block is set (or reset) if the block stores dirty (or clean) data. When a cache line is written back, only those blocks having their d-bits set are sent to its lower level cache. The d-bits are then reset to indicate that the data in the blocks are now clean. By reducing the number of blocks updated to lower level cache, the probability of assurance update triggered by lower level cache can also be reduced.

B. Data swapping

The clean data stored in a g-block are "over-protected" by SECDED since the capability of detecting single-bit error is sufficient in this case. On the other hand, the dirty data stored in a t-block are "under-protected" since an error occurring in the block drives the data into unrecoverable state. For other two combinations–clean data stored in a t-block or dirty data stored in a g-block–the data are just sufficiently protected. We propose the scheme in which the under-protected and over-protected data can be swapped to improve the protectiveness of the data as a whole. Swapping is carried out between data blocks belong to the same cache line. An assurance update can be avoided if the number of dirty blocks in a cache line is no more than the number of t-blocks in the same cache line.

Figure 5.3 shows an example of data swapping. In Figure 5.3-a, since dirty data are stored into a t-block (the third block in the figure), an assurance update is required. By swapping data between the third and fourth blocks, the assurance update can be avoided, as shown in Figure 5.3-b.

In data swapping, overheads in terms of power and latency incur for clean data to be read from g-blocks and then stored back into t-blocks. However, we believe that performing a data swapping in a cache consumes less power than making an update request to its lower level cache which is typically many times bigger and slower. This is particularly true if the lower level cache is an off-chip memory.

Swapping is executed based on the information of t-bits and d-bits. There could be several ways to implement the swapping function. Instead of focusing on the detail implementation of swapping function, this research concentrates on investigating the potential of data swapping in reducing the number of assurance updates.

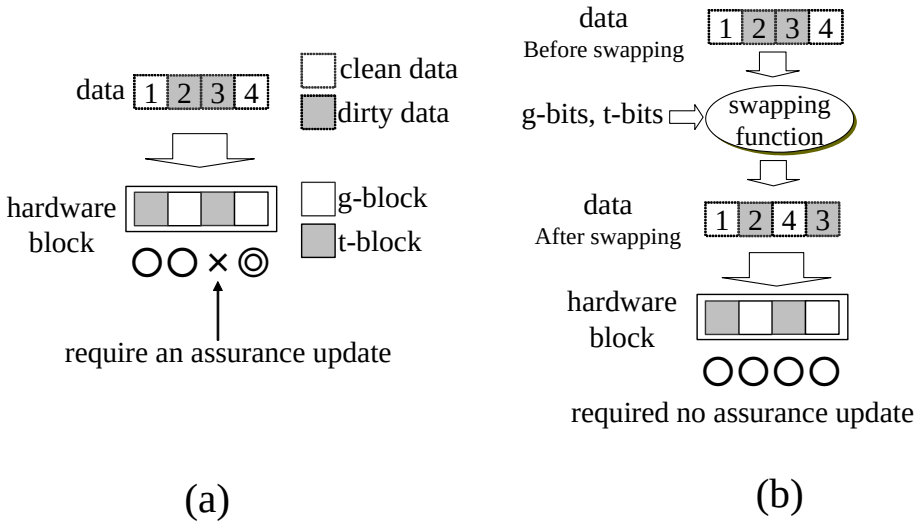


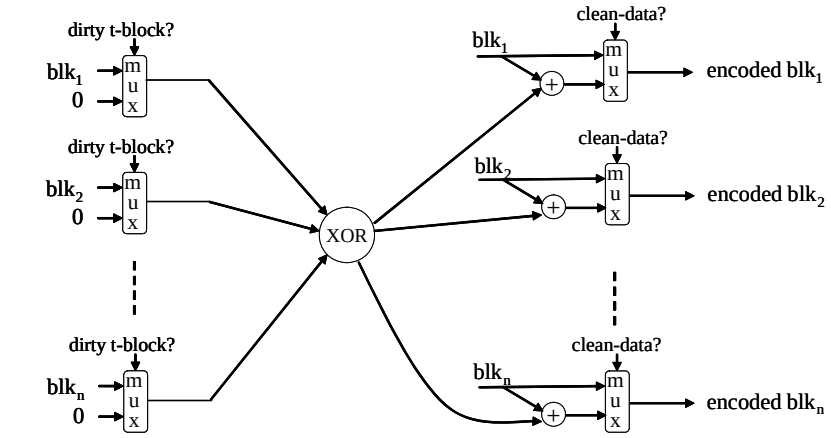
Figure 5.3: **Example of data swapping.** An assurance update is required since dirty data are stored in a t-block in (a). By swapping data between block the third and fourth blocks, assurance can be avoided in (b).

C. Data superimposition

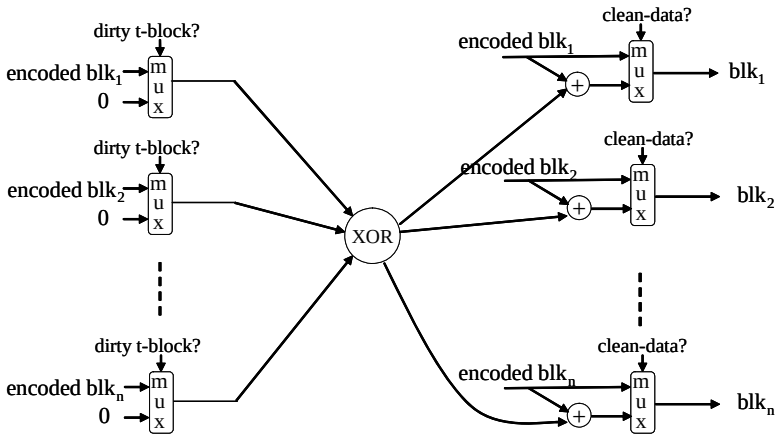
Dirty data stored in a t-block are under-protected by SECDED since a soft error occurring in the block causes uncorrectable data corruption. In data superimposition, we propose that in addition to being stored in a t-block, the under-protected data are also encoded and "cached" into those block(s) storing clean data. Figure 5.4-a shows the encoding process in data superimposition. Only clean data blocks are modified by the encoding process; those dirty data blocks stored either in t-blocks or g-blocks remain unchanged. Each clean data block is exclusive-ORed with all under-protected dirty data blocks belonging to the same cache line to produce encoded data. The encoded data in this case are also considered to be clean. Encoded data blocks are stored in a cache.

When a cache line are read, the encoded clean data need to be decoded. If soft error has not occurred, clean data block can be restored by exclusive-ORing the encoded data with those dirty data stored in other t-blocks, as shown in Figure 5.4-b.

We consider two cases in which a soft error in a t-block causes data



(a) Data encoding



(b) Data decoding

Figure 5.4: **Encoding and decoding in data superimposition.** In data encoding in (a), the clean data are XORed with dirty data stored in other t-block(s) belong to the same cache line to produce encoded clean data. In data decoding in (b), the clean data are restored by XORing the encoded clean data with dirty data stored in t-blocks.

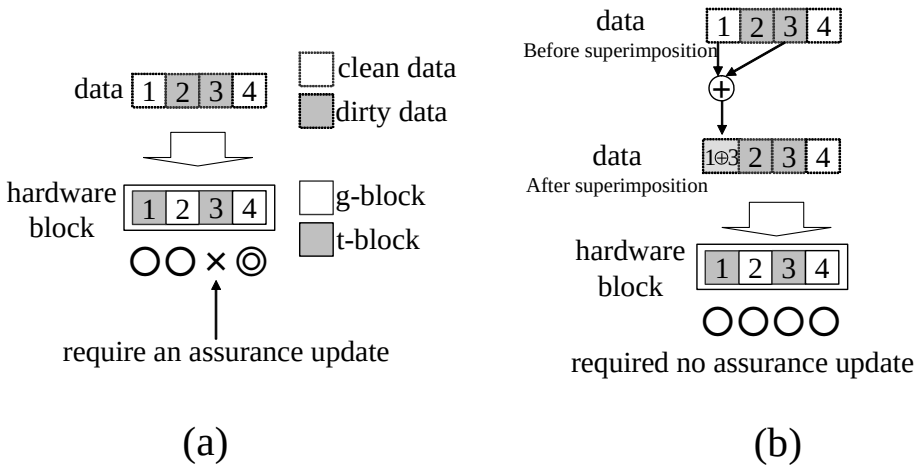


Figure 5.5: **Example of data superimposition.** In (a), an assurance update is required since dirty data are stored in a t-block (the third block). In (b), by superimposing data in the third block into the clean blocks (the first and fourth blocks), assurance update can be avoided.

corruption that cannot be corrected by SECDED. In the first case, a soft error has occurred in a t-block storing encoded clean data. Since clean data always have its copy in the lower level cache, when the error is detected, the correct data can be obtained from lower level cache.

In the second case, a soft error has occurred in a t-block storing dirty data. We assume that in a SEVA cache different cache lines are interleaved in the same row to disperse the error bits of a spatial multi-bit error. The victim t-block therefore has been corrupted by a single-bit error and it is the only block having been corrupted in the cache line. The corrupted dirty data can be restored from an encoded clean data block and other uncorrupted dirty t-blocks. First, the original clean data are fetched from the low level cache. The corrupted data are then the exclusive-OR summary of 1) the original clean data fetched from lower level cache, 2) the encoded clean data, and 3) the exclusive-OR summary of uncorrupted dirty data blocks stored in other t-blocks belonging to the same cache line.

Figure 5.5 shows an example of data superimposition. In Figure 5.5-a, since dirty data are stored into a t-block (the third block in the figure), an assurance update is required. By superimposing the dirty data in the third

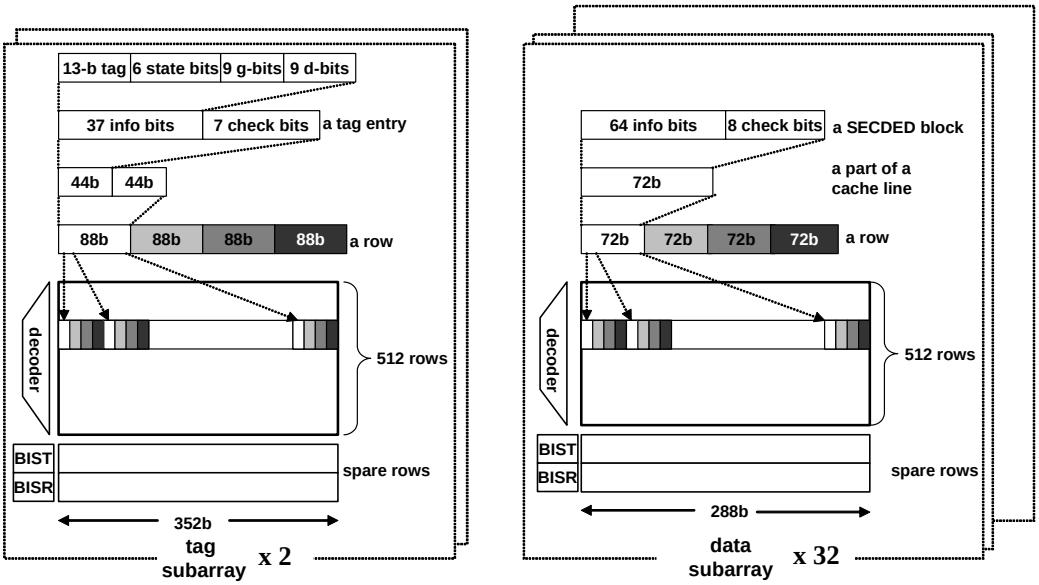


Figure 5.6: **Example structure of a SEVA cache.** The cache is a 512KB, four-way set-associative cache. The cache is consisted of 34 subarrays (two for tag and 32 for data). The g-bits and d-bits of data blocks of a cache line are stored in its tag.

block into the clean data in the first block, the assurance update can be avoided, as shown in Figure 5.5-b.

An assurance update can be avoided by data superimposition as long as there is at least one clean data block in a cache line. This condition is less strict than the condition required in data swapping. In data swapping, for an assurance update to be avoided, the number of dirty blocks in a cache line must be no more than the number of t-blocks in the same cache line. We therefore expect that more assurance updates be reduced in data swapping than in data superimposition.

5.4.5 Example SEVA Cache

Figure 5.6 shows a simplified structure of a 512KB, four-way set-associative SEVA cache. The size of a cache line is 64B. The cache is constructed from two tag subarray and 32 32-KB data subarrays. Each data subarray has 512

rows, each row is comprised of eight 72b blocks. The tag subarray has 256 rows, each row is comprised of sixteen 45b blocks. A tag entry consists of a tag address, some state bits (for LRU replacement, cache coherency), and g-bits and d-bits of all blocks belonging to the corresponding cache line.

In a typical cache, tag access usually proceeds data access. By storing g-bits and d-bits of those data blocks belong to the same cache line into the corresponding tag block in a SEVA cache, the decisions of 1) whether an assurance write is needed or not upon a cache write, and 2) which blocks are dirty that are needed to be written back upon a line eviction or an assurance update, can be determined at the end of the tag access. Accesses to unmodified blocks in a cache line can be skipped, thereby saving power consumption.

5.5 Defect and Yield Analysis

In this section, we perform defect and yield analysis of a SEVA cache. Our analysis focuses on variation-induced defects, which are the dominant type of defects in scaled devices. Other types of defects (e.g., stuck-at faults at wordlines, decoders) can also be dealt with using redundancy rows, but they are not considered in this analysis. We assume the defects are randomly distributed and λ is the probability that a cell is defective. Since a cache consists of multiple subarrays, we start with defect and yield analysis of a generalized subarray. The subarray has N_{row} rows. Each row contains N_{blk} blocks. Each block has BS bits, including information bits and check bits. The subarray has N_{rdrow} redundancy rows.

The probabilities that a block is a g-block, t-block, and b-block are respectively P_{g-blk}, P_{t-blk}, and P_{b-blk}, and are given by Equation 5.1, 5.2, and 5.3. Approximations in these equations are based on the assumption that λ is sufficiently small.

$$P_{g-blk} = (1-\lambda)^{BS} \approx 1 - BS \cdot \lambda + \frac{BS(BS-1)\lambda^2}{2} \tag{5.1}$$

$$P_{t-blk} = BS(1-\lambda)^{BS-1}\lambda \approx BS \cdot \lambda - BS(BS-1)\lambda^2 \tag{5.2}$$

$$P_{b-blk} = 1 - P_{g-blk} - P_{t-blk} \approx \frac{BS(BS-1)\lambda^2}{2} \tag{5.3}$$

A row is a bad row if it holds at least a b-block. The probability that a row is a bad row, P_{b-row}, is given by Equation 5.4.

$$P_{b-row} = 1 - (1 - P_{b-blk})^{N_{blk}} \tag{5.4}$$

An array is *passable* if the number of bad rows, N_{b-row}, is at most equal to the number of redundancy rows that are not bad rows, $N_{nb-rdrow}$. The yield of a array is given by Equation 5.5.

$$Yield_{array} = Prob(N_{b-row} \leq N_{nb-rdrow}) \tag{5.5}$$

We can simulate the yield of the subarray by assuming that N_{b-row} and $N_{nb-rdrow}$ follow Poisson distribution which means are given by Equation 5.6 and 5.7.

$$\overline{N_{b-row}} = N_{row} P_{b-row} \tag{5.6}$$

$$\overline{N_{nb-rdrow}} = N_{rdrow}(1 - P_{b-row}) \tag{5.7}$$

The yield of a cache is the product of the yields of its constituent subarrays, expressed in Equation 5.8.

$$Yield_{cache} = \prod^{all\ subarrays} Yield_{subarray} \tag{5.8}$$

5.5.1 Results

We perform yield simulation for the SEVA cache shown in Figure 5.6. The yields of the tag and data subarrays are calculated separately. The configuration parameters $\{N_{row}, N_{blk}, BS\}$ of the tag and data subarrays are respectively $\{512, 8, 44\}$ and $\{512, 4, 72\}$. We vary the number of redundancy rows to investigate its impact on the yield of the subarray. Figure 5.7 shows the yields of the tag and data subarray as the defect probability is varied from 0.0001 to 0.005. When defect probability is low, the subarrays can archieve high yield with a very few rows. As the defect probability becomes high, the number of redundancy rows required to achieve adequate yields increases appreciably.

Table 5.4 shows the yield and number of redundancy rows required in the tag and data subarrays in order to attain roughly 90% cache yield. Hardware

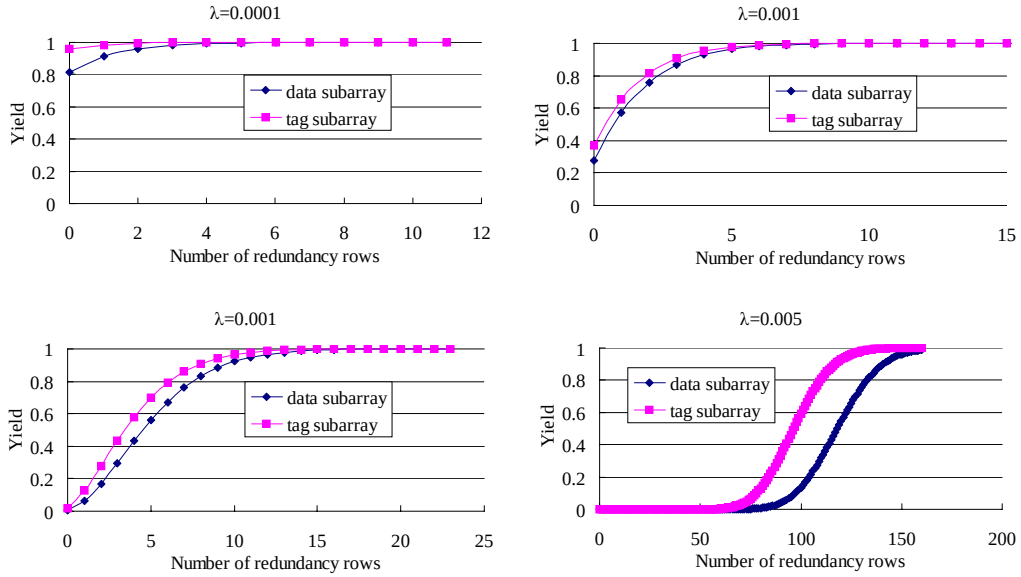


Figure 5.7: **Yields of tag and data subarrays.** The yields of a tag and data subarray are calculated as defect rate is varied from 0.0005 to 0.005.

Table 5.4: **Yields and overheads of subarrays and cache**

Defect rate	Tag subarray (yield/N_{rdrow})	Data subarray (yield/N_{rdrow})	Cache yield	Overhead (%)
0.0001	0.998/3	0.997/5	0.905	13.9
0.0005	0.997/8	0.997/9	0.903	14.7
0.001	0.995/14	0.997/17	0.899	16.4
0.005	0.997/142	0.997/165	0.903	46.4

overhead of SEVA is also shown in the table. The overhead is simply the total number of SECDED check bits, g-bits, d-bits, and the bits consumed by redundancy rows, divided by the number of memory bits in the original cache. The overhead of SECDED check bits is fixed and the overhead of redundancy rows grows up as defect probability increases. If defect probability is 0.001, SEVA can achieve 90% yield with 16.4% overhead. The overhead of redundancy rows grows substantially when defect probability becomes as high as 0.005.

5.5.2 Discussion

When defect probability becomes high enough (0.005 or larger), the approach in which all bad elements (i.e., those cannot be repaired by SECDED) must be replaced by good ones may require a large number of redundancy rows and complex repairing circuitry. One could consider another approach in which bad cache lines are marked and the processor simply does not use them. A set having bad cache line(s) is treated as a set with reduced associativity. This approach can be easily realized by providing a flag bit to each cache line to indicate whether the corresponding line is bad or not.

No matter which one of the two approaches is chosen, we need to deal with the problem of soft errors occurring in t-blocks. Assurance update mechanism therefore is applicable to both approaches.

5.6 Performance and Reliability Evaluation

The applications of SEVA caches on a processor's cache hierarchy are considered in this section. The performance overhead and reliability improvement are evaluated.

5.6.1 Evaluation Methodology

The simulated system is an out-of-order superscalar processor. Table 5.5 lists the configuration parameters. The processor has 16KB instruction and data caches, and a 512KB unified L2 cache. The data cache and L2 cache are writeback caches, each equipped with a four-entry writeback buffer.

SECDED is maintained for every 64 information bits in the L1 caches and L2 cache. We assume that all bad rows in the caches are replaced by

Table 5.5: **Parameters of Simulated Architecture**

Processor Parameters	
Frequency	1 GHz
Functional Units	4 integer ALUs, 4 FP ALUs
	1 integer multiplier/divider
	1 FP multiplier/divider
LSQ size	32 instructions
RUU size	64 instructions
Issue Width	4 instructions/cycle
Cache Hierarchy Parameters	
L1 i-cache	16KB, direct-map, 32B line, 1 cycle latency
	72b SECDED block
L1 d-cache	16KB, 4-way, 32B line, 1 cycle latency, writeback
	72b SECDED block, 4-entry writeback buffer
L2 cache	512KB, unified, 4-way, 64B line, 6 cycle latency
	72b SECDED block, 4-entry writeback buffer
Memory	100 cycle latency

redundancy rows. The probability that a block is a t-block or g-block is derived from Equation 5.1 and 5.2. The g-bits are randomly initialized at the beginning of the simulation based on such a probability. We consider defect tolerance only in SRAM caches and assume the memory to be fault-free.

The following target systems are evaluated:

- **Baseline**: The data cache and L2 cache do not perform assurance update
- **Seva-line**: The caches perform assurance updates and maintain dirtiness at cache line level.
- **Seva-blk**: The caches perform assurance updates and maintain dirtiness at block level.
- **Seva-blkswap**: The caches perform assurance updates at block level and data swapping.
- **Seva-blksuper**: The caches perform assurance updates at block level and data superimposition.

Simulation is performed using SimpleScalar [64]. SPEC2000 benchmarks are used in the simulation. For each benchmark, we skip the first one billion instructions and simulate the next four billion instructions.

We assume that the soft error rate of an unprotected SRAM equal to 1.6 KFIT per megabit [4], and soft errors follow an uniform distribution. The timestamps of accesses to the cache lines and blocks of the caches are recorded. Such information allows us to calculate the error rates.

5.6.2 Evaluation Results

Figure 5.8 shows the normalized performance when the defect rate (λ) is equal to 0.005. **Seva-line** degrades performance significantly for some applications (e.g., *lucas*, *gcc*). However, the performance degradation is reduced significantly for **Seva-line** and becomes extremely small for **Seva-blkswap** and **Seva-blksuper**. **Seva-blksuper** slightly outperforms **Seva-blkswap**. The same trend is also observed in Figure 5.9 which shows the performance degradation averaged for all benchmarks as λ is varied.

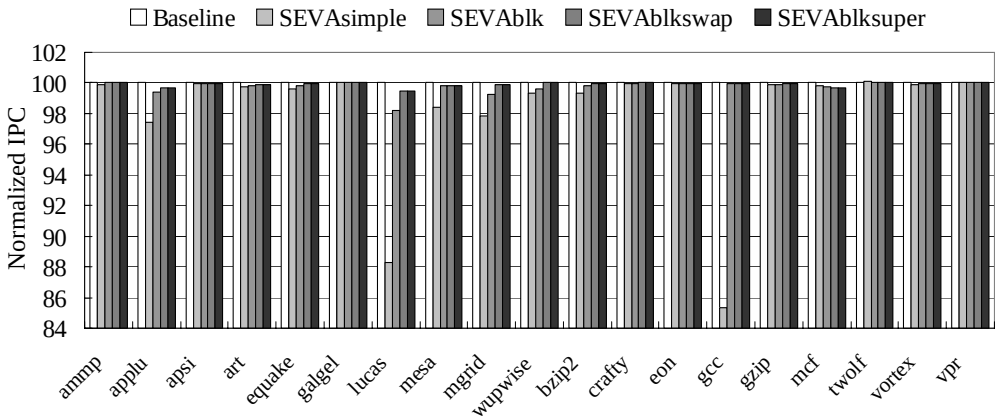


Figure 5.8: **Normalized performance when defect rate is equal to 0.005.**

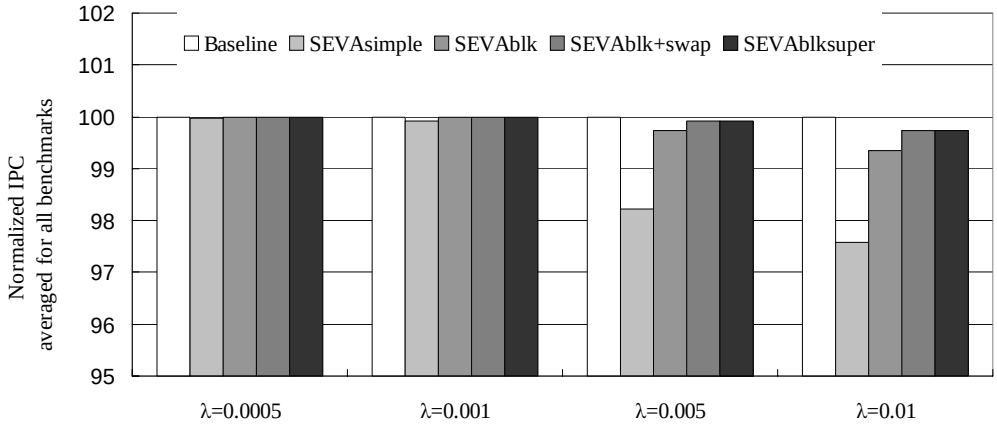


Figure 5.9: **Normalized performance as defect rate is varied.**

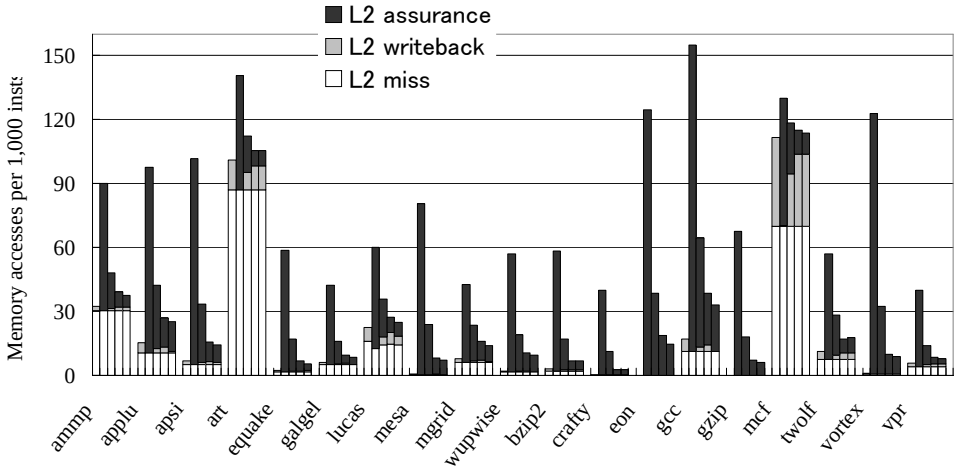


Figure 5.10: **Breakdown of accesses from L2 cache to memory per 1,000 instructions when λ is equal to 0.005.** Five bars in each group from left to right correspond to the five target caches.

Figure 5.10 shows number of accesses from L2 cache to memory per one thousand instructions with λ equal to 0.005. The breakdowns of the accesses are also shown in the figure. The number of accesses to memory for **Seva-line** increases considerably due to assurance updates for most benchmarks. Since assurance updates can have the effect of early writing back the dirty cache lines, some normal writebacks (upon cache replacements) are converted to assurance updates which can be observed through the reduction in the number of writebacks for **Seva-line** as compared with **baseline**. The number of assurance updates and writebacks are reduced greatly for **Seva-blk**. Data swapping and data superimposition allow further significant reductions in the number of assurance updates. The same trend can also be observed when λ is varied, as shown in Figure 5.11. The reduction in the number of assurance updates reduces the stalled cycles caused by a full writeback buffer and results in very small performance degradation that we noticed earlier in Figure 5.8.

Figure 5.12 shows the number of written back blocks to memory per one thousand instruction averaged for all benchmarks as λ is varied. The figure clearly confirms the effectiveness of maintaining data dirtiness per block for eliminating unnecessary block updates and improving power efficiency.

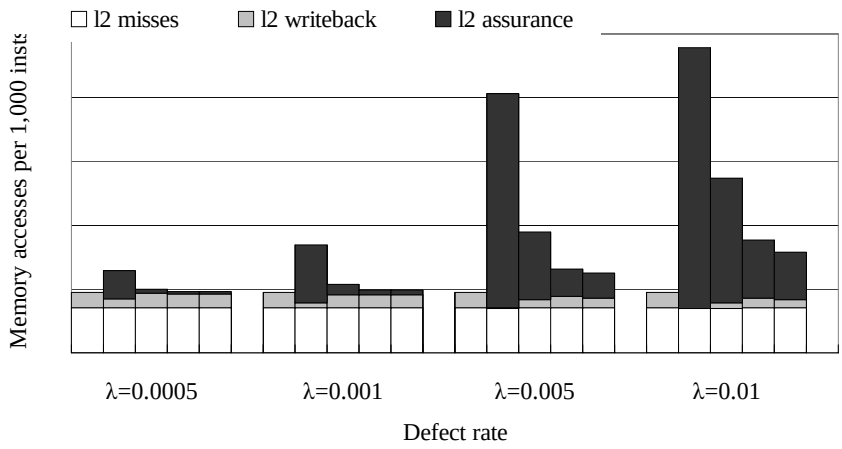


Figure 5.11: **Breakdown of accesses from L2 cache to memory per 1,000 instructions when λ is varied.** Five bars in each group from left to right correspond to the five target caches.

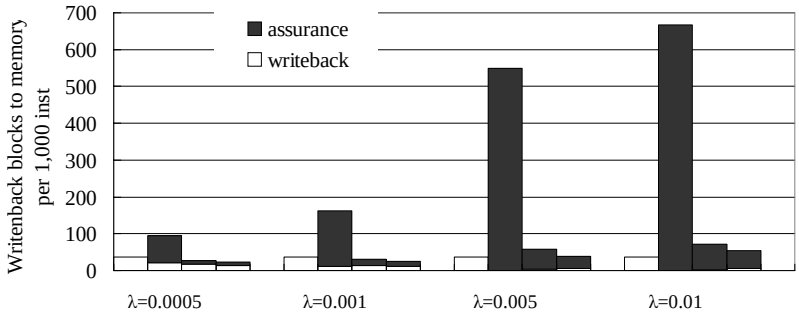


Figure 5.12: **Number of written-back blocks to memory per 1,000 instructions.** Five bars in each group from left to right correspond to the five target caches.

Table 5.6: **Uncorrectable error rate of L2 caches**

Cache	Uncorrectable Error Rate (FIT)
Baseline	631
Assurance update	3.84e-14

Table 5.6 shows the uncorrectable error rate of the baseline L2 cache and the L2 cache performing assurance update. For the baseline cache, an uncorrectable error occurs if a strike occurs on a t-block of a dirty cache line and the cache line is read later. For the cache performing assurance update, an uncorrectable error occurs if two strikes occur on the same t-block between two consecutive accesses to the block. The error rate of the cache performing assurance update is many orders of magnitude lower than that of the baseline cache.

5.7 Summary

Tolerating soft errors for high reliability and tolerating defects for yield improvement are highly required in advanced SRAM designs. The proposed SEVA cache architecture can satisfy both the requirements. By combining SECDED with a redundancy technique, SEVA can effectively tolerate a high number of variation-induced defects. To prevent unrecoverable erroneous states caused soft errors occurring in defective blocks, SEVA allows only the clean data to be stored in those blocks. This constraint is enforced through assurance update mechanism. We proposed three techniques to effectively reduce the number of assurance updates in a SEVA cache: maintaining dirtiness at SECDED's block-level, data swapping between blocks, and data superimposition. Yield analysis and performance evaluation verified the effectiveness of SEVA caches. The performance overheads caused by assurance updates in SEVA caches are very low even with high defect probability (as high as 0.01).

Chapter 6

Conclusions

6.1 Conclusions

This thesis focused on mitigating soft errors in memory caches. We have addressed the issues of existing soft-error mitigation techniques. We then proposed techniques that allow soft-error mitigation to be implemented with low costs in caches with word-based access, or in memory caches made use of Content-Addressable Memory (CAM). We also propose a cache architecture that achieves both soft-error- and defect tolerance with low cost. The followings are conclusions through this thesis.

Chapter 3: we proposed **Zigzag-HVP**, a cost-effective technique to detect and correct soft errors in caches with word-based access. Zigzag-HVP enhanced horizontal-vertical parity (HVP) to make it be able to detect and correct spatial multi-bit errors, which are expected to occur frequently in scaled processes. By dividing the data array into multiple HVP domains and interleaving the bits of different domains, a spatial MBE can be converted to multiple SBEs, each of which can be detected and corrected by the corresponding parity domain. Vertical parity update and error recovery in Zigzag-HVP can be performed efficiently by modifications to the cache data paths, write-buffer, and Built-In Self Test. Implementation of Zigzag-HVP in a 512 KB L2 cache indicated that the overheads in terms of area and power consumption were respectively 3.3% and 10%, which were smaller than those of the ECC-based ones. While Zigzag-HVP is vulnerable to MBEs accumulated from multiple particle strikes, our results show that such a probability is extremely small.

Chapter 4: we proposed **STCAM**, a soft-error tolerant CAM architecture. The unavailability of data outside a memory array in a CAM access make mitigation of soft errors in a CAM more challenging. Since CAMs are used in several components in a processor, making those CAMs being resilient against soft errors is indispensable to attain high processor's reliability. Check bits of the CAM portion in the RAM portion of a CAM-RAM structure. False hits in a CAM tag can be detected by examining the check bits of the hit tags. Mitigation of false misses involves subdividing a CAM and providing backup checking for cases the input is partially matched in the CAM. An original encoding scheme is proposed to reduce the frequency of back-up checking. Performance degradation incurred by additional cycles for false miss checking is very low (less than 0.01%).

Chapter 5: we proposed **SEVA**, a soft-error- and variation-aware cache architecture. Combination of SECDED with a redundancy technique can effectively tolerate a high number of variation-induced defects. While SECDED can repair a defective cell in a block, the block becomes vulnerable to soft errors. SEVA exploits SECDED to tolerate variation-induced defects while preserving high resilience against soft errors. SEVA allows only the clean data to be stored in the defective (but still usable) blocks. An error occurring in a defective block can be detected and the correct data can be obtained from the lower level of the memory hierarchy. We proposed three techniques to effectively reduce the number of assurance updates in a SEVA cache: maintaining dirtiness at SECDED's block-level, data swapping between blocks, and data superimpositioning. Yield analysis and performance evaluation verified the effectiveness of SEVA caches. Overheads incurred by assurance updates in SEVA caches were low even when defect probability was as high as 0.01.

Having memory caches to be resilient against soft errors is essential for attaining high processor's reliability. Incurring low area and power overheads, **Zigzag-HVP** makes support for soft-error tolerance to be more affordable and therefore pervasive. **STCAM** increases in the coverage of soft error protection in a processor making use of content-addressable memories. Finally, **SEVA** shows that soft-error tolerance for reliability and defect tolerance for yield can be archievable with reasonable costs, paving the way for successful SRAM designs in future processes.

6.2 Suggestions for Future Work

Mitigation of soft errors in memory caches is highly required, given the amount of resources devoted for them in a processor. However, as SERs of memory caches can be greatly reduced, the relative portion of SER attributed to soft errors in logic circuits will grow up. Due to the irregularity of the circuitry, mitigating soft errors in logic is more difficult than in a memory array. Methods to attack soft errors in logic employing spatial redundancy [84] [85] [86] or time redundancy [87] [88] [89] have been proposed. However, those methods incur high hardware overheads, or assume special microprocessor architectures. Novel techniques that can mitigate soft errors in logic with relatively low hardware overheads and broad applicability are still highly required to be developed.

The second suggestion is the development of tools that allow SERs to be estimated at high-level designs. Such tools are very helpful for designers to 1) verify the effectiveness of mitigation techniques, 2) explore the design spaces of soft error tolerance design, and 3) make proper trade-offs between the degree of SER reduction and the incurred hardware costs at early design stages. Early tools tended to model soft errors at device level with a lot of physics involved [90] [91]. While achieving high accuracy, they require intensive computation and are therefore inapplicable to large circuits. We have seen in very recent years several attempts to model SER at circuit- or system-levels [92] [93] [94]. While encouraging performance improvement have been reported, those tools are still at the early stage of development that require further improvements and, particularly, throughout experiments to verify their accuracy.

Publications

Journal

[1] Niko Demus Barli, Luong D. Hung, Hideyuki Miura, Chitaka Iwama, Daisuke Tashiro, Shuichi Sakai, and Hidehiko Tanaka, "Cache Coherence Strategies for Speculative Multithreding CMPs: Characterization and Performance Study". In *IPSJ Transactions on Advanced Computing Systems (ACS 7)*, Vol.45, No.SIG11, pp.119-132, Oct, 2004.

[2] Naoya Hatta, Niko Demus Barli, Chitaka Iwama, Luong D. Hung, Daisuke Tashiro, Shuichi Sakai, and Hidehiko Tanaka, "Bus Serialization for Reducing Power Consumption". In *IPSJ Transactions on Advanced Computing Systems (ACS 13)*, Vol.47, No.SIG3, Mar, 2006.

[3] Yi Ge, Takao Sakurai, Luong Dinh Hung, Koki Abe, and Shuichi Sakai, "Evaluation of Hardware Implementations for Interleaved Modular Multiplication". In *IEICE Transactions*, Vol.J88-A, No.12, pp.1497-1505, Dec, 2005.

[4] Luong D. Hung and Shuichi Sakai, "Dynamic Estimation of Task Level Parallelism with Operating System Support". In *IPSJ Transactions on Advanced Computing Systems (ACS14)*, Vol.47, No.SIG7, pp.43-51, May, 2006.

[5] Luong D. Hung, Masahiro Goshima, and Shuichi Sakai, "Zigzag-HVP: A Cost-effective Technique to Mitigate Soft Errors in Caches with Word-based Access". In *IPSJ Transactions on Advanced Computing Systems (ACS16)*, Vol.47, No.SIG7, pp.44-54, Oct, 2006.

International Conference

[6] Yoshimitsu Yanagawa, Luong Dinh Hung, Chitaka Iwama, Niko Demus Barli, Shuichi Sakai, and Hidehiko Tanaka, "Complexity Analysis of A Cache Controller for Speculative Multithreading Chip Multiprocessors". In *Proc. of the 10th International Conference on High Performance Computing (HiPC) 2003, LNCS 2913*, pages 393–404, 2003.

[7] Hideyuki Miura, Luong Dinh Hung, Chitaka Iwama, Niko Demus Barli, Shuichi Sakai, and Hidehiko Tanaka, "Compiler-assisted Thread Level Control Speculation". In *Proc. of the International Conference on Parallel and Distributed Computing Europar 2003, LNCS 2790*, pages 603–608, 2003.

[8] Niko Demus Barli, Luong Dinh Hung, Hideyuki Miura, Shuichi Sakai, and Hidehiko Tanaka, "A Dual-Length Path-Based Predictor for Thread Prediction". Oral presentation at International Workshop on Innovative Architectures 2003, 2003.

[9] Luong Dinh Hung, Chitaka Iwama, Niko Demus Barli, Naoya Hattori, Shuichi Sakai, and Hidehiko Tanaka, "Dynamic Cache Way Allocation for Static and Dynamic Power Reduction". In *International Symposium on Low-Power and High-Speed Chips (COOL Chips VII)*, at Yokohama, Japan, Vol.1, pp.268-277, Apr, 2004.

[10] Luong Dinh Hung, Masahiro Goshima, and Shuichi Sakai, "Mitigating Soft Errors in Highly Associative Cache with CAM-based Tag". In *IEEE International Conference on Computer Design (ICCD 2005)*, at San Jose, California, USA, Vol.2005, pp.342-347, Oct, 2005.

[11] Luong Dinh Hung and Shuichi Sakai, "Estimation of Task Level Parallelism with Operating System Support". In *IEEE/ACM International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN 2005)*, at Las Vegas, Nevada, USA, Vol. 2005, pp.358-363, Dec, 2005.

[12] Luong Dinh Hung, Masahiro Goshima, and Shuichi Sakai, "SEVA: A Soft-Error- and Variation-Aware Cache Architecture". In *IEEE International Symposium on Pacific Rim Dependable Computing (PRDC 2006)*, at Riverside, California, USA, pp.47-54, Dec, 2006.

[13] Luong Dinh Hung, Hidetsugu Irie, Masahiro Goshima, and Shuichi Sakai, "Utilization of SECDED for Soft Error and Variation-induced Defect Tolerance". To appear in *IEEE/ACM Design Automation and Test in Europe (DATE 2007)*, at Acropolis, Nice, France, April, 2007.

Domestic Conference

[14] Luong Dinh Hung, Hideyuki Miura, Chitaka Iwama, Daisuke Tashiro, Niko Demus Barli, Shuichi Sakai, and Hidehiko Tanaka, "A Hardware/Software Approach for Thread Level Control Speculation". In *IPSJ SIG Technical Reports 2002-ARC-149*, Vol.2002, No.12, pp.67-72, August 2002.

[15] Luong D. Hung, Masanori Takada, Yi Ge and Shuichi Sakai, "A Cost-effective Technique to Mitigate Soft Errors in Logic Circuits". *IEICE Technical Report VLD2004-49*, at the Kitakyushu International Conference Center, Vol.104, No.477, pp.31-36, Dec, 2004.

[16] Yuya Ueno, Luong D. Hung, Masanori Takada, Daisuke Tashiro and Shuichi Sakai, "Improvement of Signature-based Phase Detection and its Application to Power Reduction in Caches". *IEICE Technical Report RECONF2005-1*, at Kyoto University, Vol.105, No.42, pp.25-30, May, 2005.

[17] Taihei Momma, Luong D. Hung, Daisuke Tashiro and Shuichi Sakai, "Verification of Cache Coherency Protocol for Speculative Multithreading". *IPSJ SIG Technical Report 2005-ARC-164*, at Takeo City Bunka Kaikan, Vol.2005, No.80, pp.103-108, Aug, 2005.

[18] Ryota Shioya, Luong Dinh Hung, Hidetsugu Irie, Masahiro Goshima and Shuichi Sakai, "LRU-based Global Replacement Algorithm for Non-Uniform Shared Cache of Multi-Core Processors". *Symposium on Advanced Computing Systems and Infrastructures 2006 (SACSIS2006)*, at Osaka International Convention Center (Grand Cube Osaka), Vol.2006, No.5, pp. 23-31, May, 2006

Bibliography

[1] Jean-Claude Laprie, *Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems*, Springer Verlag, 1991.

[2] J. F. Ziegler, "Terrestrial commis rays," *IBM Journal of Research and Development*, vol. 40, no. 1, 1996.

[3] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, et al., "IBM experiments in soft fails in computer electronics," *IBM Journal of Research and Development*, vol. 40, no. 1, 1996.

[4] Robert Baumann, "Soft Errors in Advanced Computer System," *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[5] Kanad Chakraborty and Pinaki Mazumder, *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories*, Prentice Hall, 2002.

[6] Robert C. Baumann, "Soft Errors in Advanced Semiconductor Devices–Part I: Three Radiation Sources," *IEEE Transactions on Device and Materials Reliability*, vol. 1, no. 1, pp. 17–22, 2001.

[7] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," in *Proc. International Conference on Dependable Systems and Networks*, 2002, pp. 389–398.

[8] Peter Hazucha and Chirster Svensson, "Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2586–2594, 2000.

[9] Tino Heijmen, "Analytical Semi-Empiritical Model for SER Sensitivity Estimation of Deep-Submicron CMOS Circuits," in *Proc. IEEE International On-Line Testing Symposium*, 2005, pp. 3–8.

[10] K. Karnik, P. Hazucha, and J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Process," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[11] K. Mohanram and N. Touba, "Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits," in *Proc. International Symposium on Defect and Fault Tolerance*, 2003, pp. 433–440.

[12] E. Normand, "Single Event Effects in Avionics," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, 1996.

[13] Forbes, "Sun Screen" http://www.forbes.com/global/2000/1113/0323026a.html, 2000.

[14] "Cisco 12000 Single Event Upset Failures Overview and Work Around Summary," http://www.cisco.com, 2003.

[15] Heather Quinn and Paul Graham, "Terrestrial-Based Radiation Upsets: A Cautionary Tale," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005, pp. 193–202.

[16] 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, "A 1.3GHz Fifth Generation SPARC64 Microprocessor," *Journal on Solid State Circuits*, vol. 38, no. 4, pp. 636–642, 2003.

[17] Cameron McNairy and Rohit Bhatia, "Montecito: A Dual-Core Dual-Thread Itanium Processor," *IEEE Micro*, vol. 25, no. 2, pp. 10–20, 2005.

[18] Wm. A. Wulf and Sally A. McKee, "Hitting the Memory Wall: Implications of the Obvious," vol. 23, no. 1, pp. 20–24, 1995.

[19] Jonathan Chang, Stefan Rusu, Jonathan Shoemaker, Simon Tam, Ming Huang, Mizan Haque, et al., "A 130-nm Trimple-V_t 9-MB Third-Level On-Die Cache for the 1.7-GHz Itanium 2 Processor ," *Journal on Solid State Circuits*, vol. 40, no. 1, pp. 195–203, 2006.

[20] Jr. Tasch and L. H. Parker, "Memory Cell and Technology Issues for 64- and 256-Mb One-Transistor Cell MOS DRAM," *IEEE Transactions Components, Hybrids Manufacturing Technology*, vol. 77, no. 3, pp. 374–388, 1989.

[21] T. C. May, "Soft Errors in VLSI: Present and Future," *IEEE Transactions Components, Hybrids Manufacturing Technology*, vol. 2, no. 4, pp. 377–387, 1979.

[22] Amr M. Mohsen Sai-Wai Fu and Tim C. May, "Alpha-particle-induced Charge Collection Measurements and the Effectiveness of a Novel P-well Protection Barrier on VLSI Memories," *IEEE Transactions on Electron Devices*, vol. 32, no. 1, pp. 49–52, 1985.

[23] David Burnett, Craig Lage, and Al Bormann, "Soft-Error-Rate Improvement in Advanced BiCMOS SRAMs," in *Proc. IEEE International Reliability Physics Symposium*, 1993, pp. 156–160.

[24] J. D. Hayden et al., "A quadruple well, quadruple polysilicon BiCMOS process for fast 16Mb SRAM's," *IEEE Transactions on Electron Devices*, vol. 41, no. 12, pp. 2318–2325, 1994.

[25] P. E. Dodd et al., "Three-dimensional Simulation of Charge Collection and Multiple-bit Upset in Si Devices ," *IEEE Transactions on Nuclear Science*, vol. 41, no. 6, pp. 2005–2017, 1994.

[26] P. E. Dodd et al., "Single-event Upset and Snapback in Silicon-on-insulator Devices and Integrated Circuits ," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2165–2174, 2000.

[27] Kenichi Osada, Yoshikazu Saitoh, and Koichiro Ishibashi, "16.7-fA/Cell Tunnel-Leakage-Suppressed 16-Mb SRAM for Handling Cosmic-Ray-Induced Multierrors," *IEEE Journal on Solid State Circuits*, vol. 38, no. 11, pp. 1952–1957, 2003.

[28] Toshikazu Suzuki, Yoshinobu Yamagami, Akinori Shibayama, Hironori Akamatsu, and Hiroyuki Yamauchi, "A Sub-0.5-V Operating Embedded SRAM Featuring a Multi-Bit-Error-Immune Hidden-ECC Scheme," *IEEE Journal on Solid State Circuits*, vol. 41, no. 1, pp. 152–160, 2006.

[29] Shubhendu S. Mukherjee, Joel Emer, Tryggve Fossum, and Steven K. Reinhardt, "Cache Scrubbing in Microprocessors: Myth or Necessity," in *Proc. IEEE Pacific Rim International Symposium on Dependable Computing (PRDC2004)*, 2004, pp. 37–42.

[30] S. Hellebrand, H. J. Wunderlich, A. Ivaniuk, Y. Klimets, and V. N. Yarmolik, "Error detecting refreshment for embedded DRAMs," in *Proc. VLSI Test Symposium*, 1999, pp. 384–390.

[31] H.T. Weaver, "An SEU Tolerant Memory Cell Deriveed from Fundamental Studies of SEU mechanisms in SRAM," *IEEE Transactions on Nuclear Science*, vol. 34, no. 6, pp. 1281, 1987.

[32] L. R. Rockett, "Simulated SEU Hardened Scaled CMOS SRAM Cell Design Using Gated Resistors," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2534–2543, 1998.

[33] T. Carlin, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, 1996.

[34] Seongwoo Kim and A. K. Somani, "Area Efficient Architectures for Information Integrity in Cache Memories," in *Proc. 26th International Symposium on Computer Architecture*, 1999, pp. 246–255.

[35] Wei Zhang, "Replication cache: a small fully associative cache to improve data cache reliability," *IEEE Transactions on Computers*, vol. 54, no. 12, pp. 1547–1555, 2005.

[36] Wei Zhang, Sudhanva Gurumurthi, Mahmut Kandemir, and Anand Sivasubramaniam, "ICR: In-Cache Replication for Enhancing Data Cache Reliability," in *Proc. International Conference on Dependable Systems and Networks (DSN-2003)*, 2003, pp. 291–300.

[37] K. Johansson, M. Ohlsson, N. Olsson, J. Blomgren, and P-U. Renberg, "Neutron Induced Single-word Multiple-bit Upset in SRAM," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1427–1433, 1999.

[38] Jose Maiz, Scott Hareland, Kevin Zhang, and Patrick Armstrong, "Characterization of Multi-bit Soft Error events in advanced SRAMs," in *Proc. IEEE International Electron Devices Meeting*, 2003, pp. 519–522.

[39] T. Calin, F. L. Vargas, and M. Nicolaidis, "Upset-tolerant CMOS SRAM using Current Monitoring," in *Proc. IEEE International Test Conference*, 1995, pp. 45–53.

[40] Balkaran Gill, Michael Nicolaidis, Francis Wolf, Chris Papachristou, and Steven Garverick, "An Efficient BICS Design for SEUs Detection and Correction in Semiconductor Memories," in *Proc. Design, Automation and Test in Europe Conference (DATE05)*, 2005, pp. 592–597.

[41] Lin Li, Vijay Degalahal, N Vijykrishnan, Mahmut Kandemir, and Mary Jane Irwin, "Soft Error and Energy Consumption Interactions:

A Data Cache Perspective," in *Proc. International Symposium on Low Power Electronics and Design*, 2004, pp. 132–137.

[42] Hsien-Hsin S. Lee, Gary S. Tyson, and Matthew K. Farrens, "Eager writeback – a technique for improving bandwidth utilization," in *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2000, pp. 11–21.

[43] Hossein Asadi, Mehdi Tahoori, and David Kaeli, "Balancing Performance and Reliability in the Memory Hierarchy," in *Proc. International Symposium on Performance Analysis of Systems and Software*, 2005, pp. 269–279.

[44] R. H. Paschburg, "Software implementation of error-correcting codes," University of Illinois, Urbana AD-786 542, 1974.

[45] Philip P. Shirvani, Nirmal R. Saxena, and Edward J. McCluskey, "Software-Implemented EDAC Protection Against SEUs," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 273–248, 2000.

[46] Azeez Bhavnagarwala, Stephen Kosonocky, Carl Radens, Kevin Stawiasz, Randy Mann, Qiuyi Ye, and Ken Chin, "Fluctuation Limits & Scaling Oppoturnities for CMOS SRAM Cells," in *Proc. IEEE International Electron Devices Meeting*, 2005, pp. 659–662.

[47] Amit Agarwal, Bipul C. Paul, Saibal Mukhopadhyay, and Kaushik Roy, "Process Variation in Embedded Memories: Failure Analysis and Variation Aware Architecture," *IEEE Journal on Solid State Circuits*, vol. 40, no. 9, pp. 1804–1814, 2005.

[48] Charles H. Stapper and Hsing-San Lee, "Synergistic Fault-Tolerance for Memory Chips," *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1078–1088, 1992.

[49] Chin-Lung Su and Yi-Ting Yeh, "An Integrated ECC and Redundancy Repair Scheme for Memory Reliability Enhancement," in *Proc. IEEE International Test Conference*, 2005, pp. 81–89.

[50] Tom Grutkowski and Reid Riedlinger, "The High Bandwidth, 256KB 2nd Level Cache on an Itanium Microprocessor," in *Proc. IEEE International Symposium on Solid-State Circuits Conference*, 2002, pp. 418–419.

[51] Patrick J. Meaney, Scott B. Swaney, Pia N. Sanda, and Lisa Spainhower, "IBM z99 Soft Error Detection and Recovery," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 419–427, 2005.

[52] Parag K. Lala, *Self-checking and Fault-tolerant Digital Design*, Morgan Kaufman Publishers, 2001.

[53] R. Dean Adams, *High Performance Memory Testing*, Kluwer Academic Publishers, 2003.

[54] Jeff Brauch and Jey Fleischman, "Design of Cache Test Hardware on the HP PA8500," *IEEE Design and Test of Computers*, vol. 15, no. 3, pp. 58–63, 1998.

[55] M. Pflanz, K. Walther, C. Galke, and H. T. Vierhaus, "On-Line Error Detection and Correction in Storage Elements with Cross-Parity Check," in *Proc. 8th International On-Line Testing Workshop*, 2002, pp. 69–73.

[56] Premkishore Shivakumar and Norman P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," Tech. Rep. WRL-2001-2 HP Labs Technical Reports, 2001.

[57] Norman P. Jouppi, "Cache Write Policies and Performance," in *Proc. 20th International Symposium on Computer Architecture*, 1993, pp. 191–201.

[58] "PowerPC G5 White Paper," http://www.apple.com/g5processor, 2004.

[59] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel, "The Microarchitecture of Pentium 4 Processor," *Intel Technology Journal*, vol. 5, no. 1, 2001.

[60] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta, *Parallel Computer Architecture: A hardware/software approach*, Morgan Kaufman Publishers, Inc., 1999.

[61] Jinuk Luke Shin, Bruce Petrick, Mandeep Shingh, and Ana Sonia Leon, "Design and Implementation of an Embedded 512-KB Level-2 Cache Subsystem," *IEEE Journal on Solid State Circuits*, vol. 40, no. 9, pp. 1815–1820, 2005.

[62] D. Wendell, J. Lin, P. Kaushik, S. Seshadri, A. Wang, V. Sundaraman, P. Wang, H. McIntyre, S. Kim, et al., "A 4MB On-Chip L2 Cache for a 90nm 1.6GHz 64b SPARC Microprocessor," in *Proc. IEEE International Symposium on Solid-State Circuits Conference*, 2004, p. 66.

[63] Kevin Skadron and Douglas W. Clark, "Design Issues and Tradeoffs for Write Buffers," in *Proc. IEEE International Symposium on High-Performance Computer Architecture*, 1997, pp. 144–155.

[64] D. Burger and T. Austin, "The SimpleScalar Tool Set," Technical Report CS-TR-1997-1342, University of Wisconsin-Madison, 1997.

[65] Doug Bossen, "CMOS Soft Errors and Server Design," in *2002 International Reliability Physics Symposium Tutorial Notes-Reliability Fundamentals*, 2002.

[66] R. Witek and J. Montanaro, "Strong ARM: a high-performance ARM processor," in *Proc. Technologies for the Information Superhighway (Compcon96)*, 1996, pp. 188–191.

[67] Steve Furber, *ARM System-on-Chip Architecture*, Addison-Wesley, 2000.

[68] Thomas D. Burd and Robert W. Brodersen, *Energy Efficient Microprocessor Design*, Kluwer Academic Publishers, 2002.

[69] Jonathan R. Haigh, Michael W. Wilkerson, Jay B. Miller, Timothy S. Beatty, Stephen J. Strazdus, and Lawrance T. Clark, "A Low-Power 2.5-GHz 90-nm Level 1 Cache and Memory Management Unit," *Journal on Solid State Circuits*, vol. 40, no. 5, pp. 1190–1199, 2005.

[70] Alex Veidenbaum and Dan Nicolaescu, "Low Energy, Highly-Associative Cache Design for Embedded Processors," in *Proc. IEEE International Conference on Computer Design (ICCD2004)*, 2004, pp. 332–335.

[71] Michael Zhang and Krste Asanovic, "Highly-Associative Caches for Low-Power Processors," in *Proc. Kool Chips Workshop, in conjunction with MICRO33*, 2000.

[72] "Intel XScale Technology," http://www.intel.com/design/intelxscale.

[73] "International Technology Roadmap for Semiconductor," http://public.itrs.net, 2005.

[74] David J. Frank, Yuan Taur, Meikei Ieong, and Hon, "Monte Carlo Modelling of Threshold Variation due to Dopant Fluctuations," in *Proc. IEEE International Electron Devices Meeting*, 1999, pp. 93–94.

[75] Xinghai Tang, Vivek K. De, and James D. Meindl, "Intrinsic MOSFET Parameter Fluctuations Due to Random Dopant Placement," in *Proc. IEEE International Electron Devices Meeting*, 1997, pp. 369–376.

[76] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohner, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM Journal of Research and Development*, vol. 50, no. 4/5, pp. 433–450, 206.

[77] Y. Zorian, "Embedded infrastructure IP for SOC yield improvement," in *Proc. IEEE/ACM Design Automation Conference*, 2002, pp. 709–712.

[78] Y. Ooi, M. Kashimura, H. Takeuchi, and E. Kawamura, "Fault-Tolerant Architecture in a Cache Memory Control LSI," *IEEE Journal on Solid State Circuits*, vol. 44, no. 2, pp. 170–179, 1995.

[79] Alfredo Benso, Silvia Chiusano, Giorgio D. Natale, Paolo Prinetto, and Monica L. Bodoni, "A family of self-repair SRAM cores," in *Proc. IEEE International On-Line Testing Workshop*, 2000, pp. 214–218.

[80] Volker Schober, Steffen Paul, and Olivier Picot, "Memory Built-in Self-Repair using Redundant words," in *Proc. IEEE International Test Conference*, 2001, pp. 995–1001.

[81] T.R.N.Rao and E. Fujiwara, "Error-Control Coding for Computer Systems," Prentice Hall, Inc, 1989.

[82] Gustavo Neuberger, Fernanda Gusmao De Lima, Kastensmidt, and Ricardo Reis, "An Automatic Technique for Optimizing Reed-Solomon Codes to Improve Faut Tolerance in Memories," *IEEE Transactions on Computers*, vol. 22, no. 1, pp. 50–58, 2005.

[83] Jean-Loup Baer and Wen-Hann Wang, "On the inclusion properties for multi-level cache hierarchies," in *Proc. IEEE International Symposium on Computer Architecture*, 1988, pp. 73–80.

[84] Mohamed Gomaa, Chad Scarbrough, T. N. Vijaykumar, and Irith Pomeranz, "Transient-fault recovery for chip multiprocessor," in *Proc. International Symposium on Computer Architecture*, 2003, pp. 96–109.

[85] NonStop Advanced Architecture, "David Bernick and Bill Brukert and Paul Del Vigna and David Garcia," in *Proc. International Conference on Dependable Systems and Networks*, 2005, pp. 12–21.

[86] Todd M. Austin, "DIVA: a reliable substrate for deep submicron microarchitecture design," in *Proc. International Symposium on Microarchitecture*, 1999, pp. 196–207.

[87] Toshinori Sato, "Exploiting Instruction Redundancy for Transient Fault Tolerant," in *Proc. of the 18th IEEE International Symposium on Defect and Fault Tolerant in VLSI Systems (DFT03)*, 2003, pp. 547–554.

[88] Steven K. Reinhardt and Shubhendu S. Mukherjee, "Transient fault defection via simultaneous multithreading," in *Proc. International Symposium on Computer Architecture*, 2000, pp. 25–36.

[89] Eric Rotenberg, "AR-SMT: A micro-architecture approach to fault tolerance in microprocessors," in *Proc. IEEE International Symposium on Fault-Tolerant Computing*, 1999, pp. 84–92.

[90] Y. Tosaka, S. Satoh, and T. Itakura, "Neutron-Induced Soft Error Simulator and Its Accurate Predictions," in *International Conference on Simulation of Semiconductor Processes and Devices*, 1997, pp. 253–256.

[91] P. C. Murley and G. R. Srinivasan, "Soft-error Monte Carlo modeling program, SEMM," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 109–118, 1996.

[92] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor ," in *Proc. International Symposium on Microarchitecture*, 2003, pp. 29–40.

[93] Xiaodong Li, Sarita V. Adve, Pradip Bose, and Jude A. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," in *Proc. International Conference on Dependable Systems and Networks*, 2005, pp. 496–505.

[94] R. Rajaraman, J. S. Kim, N. VijayKrishnan, Y. Xie, and M. J. Irwin, "SEAT-LA: A Soft Error Analysis tool for Combinational Logic," in *Proc. International Conference on VLSI Design (VLSID06)*, 2006.