Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

SOFT-ERROR RESILIENT ON-CHIP MEMORY STRUCTURES

by
Shuai Wang

Soft errors induced by energetic particle strikes in on-chip memory structures, such as L1 data/instruction caches and register files, have become an increasing challenge in designing new generation reliable microprocessors. Due to their transient/random nature, soft errors cannot be captured by traditional verification and testing process due to the irrelevancy to the correctness of the logic. This dissertation is thus focusing on the reliability characterization and cost-effective reliable design of on-chip memories against soft errors.

Due to various performance, area/size, and energy constraints in various target systems, many existing unoptimized protection schemes on cache memories may eventually prove significantly inadequate and ineffective. This work develops new lifetime models for data and tag arrays residing in both the data and instruction caches. These models facilitate the characterization of cache vulnerability of the stored items at various lifetime phases. The design methodology is further exemplified by the proposed reliability schemes targeting at specific vulnerable phases. Benchmarking is carried out to showcase the effectiveness of these approaches.

The tag array demands high reliability against soft errors while the data array is fully protected in on-chip caches, because of its crucial importance to the correctness of cache accesses. Exploiting the address locality of memory accesses, this work proposes a Tag Replication Buffer (TRB) to protect information integrity of the tag array in the data cache with low performance, energy and area overheads. To provide a comprehensive evaluation of the tag array reliability, this work also proposes a refined evaluation metric, detected-without-replica-TVF (DOR-TVF), which combines the TVF and access-with-replica (AWR) analysis. Based on the DOR-TVF analysis, a TRB scheme with early write-back (TRB-EWB) is proposed, which achieves a zero DOR-TVF at a negligible per-

formance overhead.

Recent research, as well as the proposed optimization schemes in this cache vulnerability study, have focused on the design of cost-effective reliable data caches in terms of performance, energy, and area overheads based on the assumption of fixed error rates. However, for systems in operating environments that vary with time or location, those schemes will be either insufficient or over-designed for the changing error rates. This work explores the design of a self-adaptive reliable data cache that dynamically adapts its employed reliability schemes to the changing operating environments in order to maintain a target reliability. The experimental evaluation shows that the self-adaptive data cache achieves similar reliability to a cache protected by the most reliable scheme, while simultaneously minimizing the performance and power overheads.

Besides the data/instruction caches, protecting the register file and its data buses is crucial to reliable computing in high-performance microprocessors. Since the register file is in the critical path of the processor pipeline, any reliable design that increases either the pressure on the register file or the register file access latency is not desirable. This work proposes to exploit narrow-width register values, which represent the majority of generated values, for making the duplicates within the same register data item. A detailed architectural vulnerability factor (AVF) analysis shows that this in-register duplication (IRD) scheme significantly reduces the AVF in the register file compared to the conventional design. The experimental evaluation also shows that IRD provides superior read-with-duplicate (RWD) and error detection/recovery rates under heavy error injection as compared to previous reliability schemes, while only incurring a small power overhead.

By integrating the proposed reliable designs in data/instruction caches and register files, the vulnerability of the entire microprocessor is dramatically reduced. The new lifetime model, the self-adaptive design and the narrow-width value duplication scheme proposed in this work can also provide guidance to architects toward highly efficient reliable system design.

SOFT-ERROR RESILIENT ON-CHIP MEMORY STRUCTURES

by
Shuai Wang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Engineering

Department of Electrical and Computer Engineering

January 2010

APPROVAL PAGE

SOFT-ERROR RESILIENT ON-CHIP MEMORY STRUCTURES

Shuai Wang

Dr. Jie Hu, Dissertation Advisor — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

Dr. Sotirios G. Ziavras, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Roberto Rojas-Cessa, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Joseph Y. Leung, Committee Member — Date
Distinguished Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Shuai Wang

Degree: Doctor of Philosophy

Date: January 2010

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Engineering,
  New Jersey Institute of Technology, Newark, New Jersey, 2010
- Bachelor of Science in Computer Science,
  Nanjing University, China, 2003

Major: Computer Engineering

Publications:

Shuai Wang, Jie Hu, and Sotirios G. Ziavras. On the Characterization and Optimization of On-Chip Cache Reliability against Soft Errors, IEEE Transactions on Computers (TC), Volume 58, Issue 9, pp. 1171 - 1184, September 2009.

Jie Hu, Shuai Wang, and Sotirios G. Ziavras. On the Exploitation of Narrow-Width Values for Improving Register File Reliability, IEEE Transactions on Very Large Scale Integration Systems (TVLSI), Volume 17, Issue 7, pp. 953 - 963. July 2009.

Shuai Wang, Jie Hu, Sotirios G. Ziavras, and Sung Woo Chung. Exploiting Narrow-Width Values for Thermal-Aware Register File Designs, In Proc. of the Conference on Design, Automation and Test in Europe (DATE 2009), pp. 1422 - 1427, Nice, France, April 20-24, 2009.

Shuai Wang, Jie Hu, and Sotirios G. Ziavras. Self-Adaptive Data Caches for Soft-Error Reliability, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Volume 27, Issue 8, pp. 1503 - 1507, August 2008.

Shuai Wang, Hongyan Yang, Jie Hu, and Sotirios G. Ziavras. Asymmetrically Banked Value-Aware Register Files for Low Energy and High Performance, Microprocessors and Microsystems, Volume 32, Issue 3, pp. 171 - 182, May 2008.

Shuai Wang, Jie Hu, and Sotirios G. Ziavras. BTB Access Filtering: A Low Energy and High Performance Design, In Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2008), pp. 81 - 86, Montpellier, France, April 7-9, 2008.

Shuai Wang, Hongyan Yang, Jie Hu, and Sotirios G. Ziavras. Asymmetrically Banked Value-Aware Register Files, In Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2007), pp. 363 - 368, Porto Alegre, Brazil, May 9-11, 2007.

Hongyan Yang, Shuai Wang, Sotirios G. Ziavras, and Jie Hu. Vector Processing Support for FPGA-Oriented High Performance Applications, In Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2007), pp. 447 - 448, Porto Alegre, Brazil, May 9-11, 2007.

Shuai Wang, Jie Hu, and Sotirios G. Ziavras. On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors, In Proc. of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS VI), pp. 14 - 20, Samos, Greece, July 17-20, 2006.

Jie Hu, Shuai Wang and Sotirios G. Ziavras. In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability, In Proc. of the International Conference on Dependable Systems and Networks (DSN 2006) - Dependable Computing and Communications Symposium (DCCS), pp. 281 - 290, Philadelphia, PA, June 25-28, 2006.

Jie Hu, Greg M. Link, Johnsy John, Shuai Wang, Sotirios G. Ziavras. Resource-Driven Optimizations for Transient-Fault Detecting SuperScalar Microarchitectures, In Proc. of the Asia-Pacific Computer Systems Architecture Conference (ACSAC 2005), pp. 200 - 214, Singapore, October 24-26, 2005.

To My Mom and Dad

ACKNOWLEDGMENT

First, I would like to thank my advisor Professor Jie Hu for supporting me throughout this work and for his constant advice and encouragement. He has showed me the whole picture of research on computer architecture, guided me in entering and exploring this exciting area. I treasure my learning experience under his invaluable guidance. I also like to thank Professor Sotirios G. Ziavras for spending incalculable effort and time working with me on my research projects. Over the course of our interaction, his devotion and persistence in his research, intelligent and insightful thoughts, and diligent work have been an inspiration.

Thanks to Professor Roberto Rojas-Cessa. I have gotten a lot out of my collaboration with him. Thanks to Professor Edwin Hou for advising me on my TA work for four years. Thanks to Professor Joseph Y. Leung for reading my dissertation and sitting on my committee.

I would like to thank all the member of our CAPPL research group for their friendly assistance and interesting discussions. Thanks to Xiaofang Wang, Hongyan Yang, Mohammad Z. Hasan, Xizhen Xu, and Johnsy K. John.

Thanks to all my friends who have made my life in NJIT so enjoyable and memorable. Thanks to Mr. and Mrs. Huang for helping a lot in my everyday life. Special thanks to Xianhong Feng for all her support and making my life more colorful.

Last but not lest, I like to thank Mom and Dad for their endless love, patience, support and encouragement. Most of all, thank you for sharing this journey and believing in me.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Soft Errors

System failures are caused mainly by two types of faults: hardware faults and software faults. Software faults are design faults that are closely related to human factors and the design process. In contrast, hardware faults are dominated by physical faults. Given the fault occurrence pattern during system operation, hardware faults can be divided into permanent faults and temporary faults. Permanent faults originate from incorrect designs, manufacturing defects, device wearout, etc. Temporary faults can be further classified into two groups with different origins: transient and intermittent. Intermittent faults are mainly due to operation margin problems, weak parts, process variation, random dopant fluctuation, etc. Different from other faults, transient faults (soft errors) in electronic systems are caused by external interferences such as energetic particles from radioactive impurities and cosmic rays, electrical noise, electromagnetic interference, etc. Many permanent faults can be avoided by thorough validation, testing, and early life failure screening. Modular redundancy is also commonly employed for highly reliable systems design. Due to the transient/random nature, transient faults cannot be captured by traditional verification and testing process due to the irrelevancy to the correctness of the logic. On the other hand, the high expense of applying techniques such as hardware triple modular redundancy (TMR) or N-modular redundancy (NMR) for addressing soft errors might not be acceptable to commercial computer systems in most market segments. As the electrical noise and electromagnetic interference can be effectively addressed in a satisfactory manner by shielding and sound designs, energetic particle induced soft errors present tremendous challenges in systems design[1][2][3].

With continuous technology scaling down, on-chip memory structures, such as on-

chip *Caches* and *Register Files*, suffer from a significantly higher Soft Error Rate (SER) than on-chip combinational logic at the current and near future technologies, due to their large share of the transistor budget and die area [4]. In SRAM cells, an upset event caused by the soft errors can charge or discharge a particular node to incur a bit flip. Figure 1.1 shows the schematic of soft error generation by a cosmic ray [1]. However, this single upset event (SUE) doses not damage the circuit. A method to estimate the SER in CMOS circuits was developed in [5]. The following equations summarize this model:

$$\begin{aligned} SER &= NF \times CS \\ CS &\propto A_{Drain} \times exp(-\frac{Q_{Critical}}{Q_S}) \end{aligned} \tag{1.1}$$

where NF is the intensity of the Neutron Flux, CS is the atmospheric neutron Cross Section, $Q_{Critical}$ is the Critical Charge of a particular node, Q_S is the Collection Slope, and A_{Drain} is the Drain Area. If a collected charge Q caused by a particle strike exceeds critical charge $Q_{Critical}$ of a circuit node, it results in a bit flip in that node and a soft error occurs. Q_S depends on the doping and the supply voltage V_{CC}. $Q_{Critical}$ is proportional to the node capacitance and the V_{CC}. According to this model, at device/circuit level, the soft error rate can be reduced by hardening the CMOS transistors either increasing $Q_{Critical}$, reducing Q_S, or reducing A_{Drain} [6][7][8][9][10]. However, due to their inability to exploit architectural

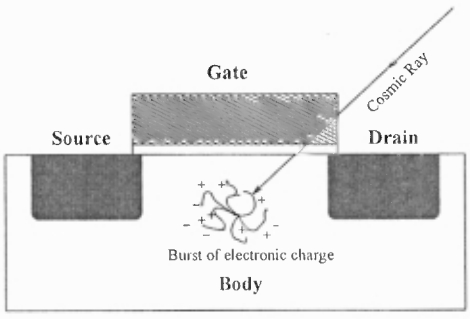


Figure 1.1 Soft error generation by a cosmic ray.

or application specific features, circuit level techniques are increasingly recognized as non-cost-effective over-designs. In contrast, this dissertation is targeting at microarchitecture-level designs and analysis of reliable on-chip memory structures against soft errors.

1.2 On-Chip Caches

To bridge the speed gap between the fast CPU and the main memory, today's microprocessors adopt the memory hierarchy design by exploiting the principle of locality [11]. Figure 1.2 shows the levels of a typical memory hierarchy in the modern computer system. Basically, there are two types of L1 caches, data cache and instruction cache. The data cache contains the data that programs need and can support the read and write operations. The instruction cache is a read-only memory structure that stores the instructions of programs. The L2/L3 caches can be either on-chip or off-chip caches [11]. This dissertation focuses on the reliability design of the on-chip L1 data/instruction caches.

Each cache contains two main components, the data array and tag array. The data information is stored in the data array. The tag array stores the tag information to determine cache hit or miss during cache accesses. Figure 1.3 shows how an address is formatted to locate a certain byte in the cache. In a cache access, Set Index locates the set and Byte Offset selects the corresponding byte from that set. Tag field of the address are compared with the tag of the selected set stored in the tag array. If the result is a match, the cache hits. Otherwise, the cache misses. The number of bits in each address field are summarized in Equation 1.2, where Bit_{BO} is the number of bits in the Byte Offset field, Bit_{SI} is the number of bits in the Set Index field, Bit_{Tag} is the number of bits in the Tag field, Bit_{Addr} is the number of bits of the entire address, N is the number of bytes in each set, and M is the number of sets in the cache.

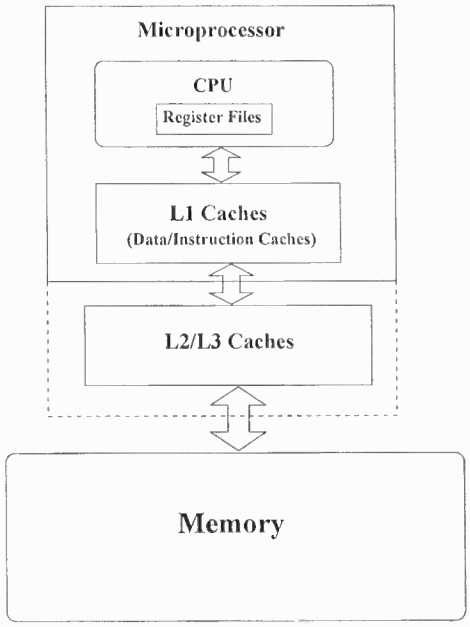


Figure 1.2 Levels of a typical memory hierarchy in the modern computer system.

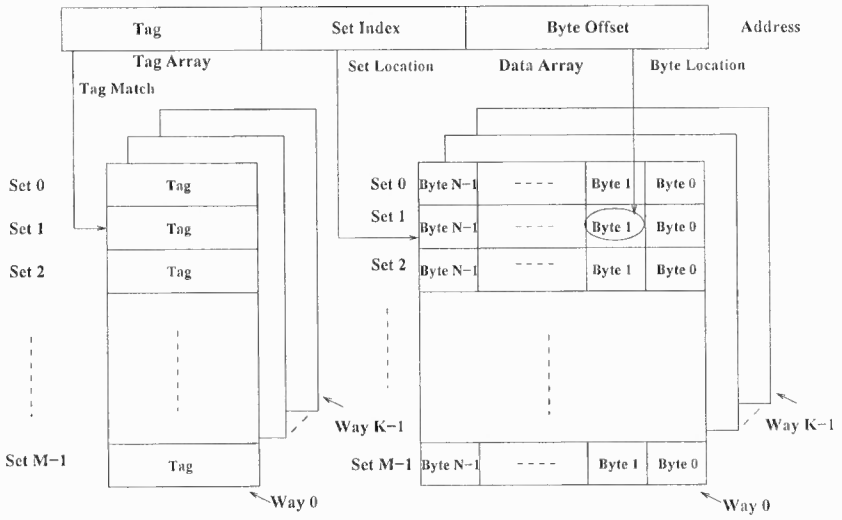


Figure 1.3 Address format for the cache access.

$$
\begin{aligned}
Bit_{BO} &= \log_2 N \\
Bit_{SI} &= \log_2 M \\
Bit_{Tag} &= Bit_{Addr} - Bit_{BO} - Bit_{SI}
\end{aligned} \tag{1.2}
$$

When the L1 data cache is updated by the CPU, the corresponding data in the lower memory hierarchy (L2/L3 caches and main memory) also need to be updated. The *write policy* controls the time of the update. In a *write-through* cache, every write to the cache causes a write to the lower memory hierarchy. In a *write-back cache*, writes are not immediately mirrored to the lower levels. Instead, the cache marks these updated cachelines as dirty. Data in these dirty cachelines are written to the lower levels when these cachelines are evicted from the cache.

1.3 Register Files

Superscalar microprocessors dynamically exploit instruction-level parallelism (ILP) to issue multiple instructions per cycle for improved performance. Register renaming is one of the fundamental techniques employed in superscalar microprocessors to increase the ILP by eliminating the two false data dependences, write-after-read (WAR) and write-after-write (WAW) [11]. Microprocessors supporting register renaming present two views of the register files, the architectural/logical register file that is visible to the compiler/programmer, and the physical register file that is managed by the register renaming mechanism. From the implementation point of view, the architectural and physical register files can be either two separately hardware-implemented register files or just one combined register file. In the separate register files implementation, once the instruction is committed, the result value in the physical register needs to be copied to its architectural register. In the combined implementation, no explicit data copy or movement is required when instructions

are committed. The mapping of the architectural registers is dynamically changing in the combined architectural/physical register file implementation.

In the register renaming stage, the logical register ids of the source operands in a decoded instruction are used to access the register alias table (RAT), a.k.a. register mapping table. The table entry indexed by the logical register id contains the physical register id that the source register was renamed to. For the destination register, a free physical register is allocated from the register free list and the RAT is updated as follows: the old physical register id is read out from the RAT and stored in the active list entry allocated to the instruction, and then the new physical register id is written to the same RAT entry indexed by the logical destination register id. The destination register is said to have been remapped to the new physical register and the old physical register is said to have been unmapped. In case the register free list is empty, the renaming stage is stalled till some physical register is freed [12]. Notice that a physical register cannot be freed until an instruction that previously unmapped this physical register is committed. Furthermore, a physical register is susceptible to soft errors only after a value is written into the register and before it is freed.

1.4 Related Work

Fault-tolerant designs based on modular redundancy have been widely used to build highly reliable systems [13]. For example, cycle-by-cycle lockstepping of dual-processors and comparison of their outputs are employed for error detection in Compaq Himalaya [14] and IBM z900 [15] with G5 processors. Other designs use asymmetric redundancy to include a watch-dog processor [16] or a low-performance checker processor in DIVA [17] to verify the correctness of the execution on the main processor.

Targeting the increasing processor vulnerability to soft errors at new technologies, temporal redundancy based reliable schemes exploiting simultaneous multithreading (SMT) architectures have been extensively studied for both single processors and chip-multiprocessors, such as AR-SMT [18], SRT [19][20], SRTR [21], and Slipstream [22].

Lately, many research efforts have been spent on exploiting the redundant resources in superscalar processors for instruction-level redundant execution against transient faults. In [23], each instruction is executed twice and the results from duplicate execution are compared to verify the absence of transient errors in functional units. However, each instruction only occupies a single re-order buffer (ROB) entry. On the other hand, the dual-instruction execution scheme (DIE) in [24] physically duplicates each decoded instruction to provide a *Sphere of Replication* including the instruction issue queue/ROB, functional units, physical register files, and the interconnection among them. Due to the substantially increased pressure on the hardware resources, dual-instruction execution in general suffers from significant performance loss. Follow-up work such as DIE-IRB [25], SHREC [26], and PER-IRTR [27], try to alleviate the resource contention in DIE processors in order to recover the performance loss.

1.4.1 Reliable Design of On-Chip Caches

Most of the above techniques protecting the datapath within a single processor are quite independent of the memory hierarchy, where the on-chip caches and external memories are assumed to be error-free by means of some error protection schemes.

Information redundancy is fundamental in building reliable memory structures. Various coding schemes are used to protect information integrity in latches, register files, and on-chip caches, providing different levels of reliability at different performance, energy, and hardware costs. For example, simple parity coding is capable of detecting the odd number of bit errors but is not able to recover from detected errors. On the other hand, error correcting codes (ECCs) typically provide single error correction and double error detection (SEC-DED). However, the performance overhead and additional energy consumption due to ECC encoding/decoding make ECC a reluctant choice for high speed on-chip caches, i.e., L1 data and instruction caches [28]. Another form of information redundancy is to maintain redundant copies of the data in cache memories [29][30]. In these schemes,

cachelines are duplicated when they are brought to L1 caches on read/write misses or on write operations. During a cache write (store), the replicas should be also updated with the latest value. On a cache read (load) operation, multiple copies may need to be read out and compared against each other to verify the absence of soft errors or to perform majority voting. Notice that maintaining redundant copies of cachelines presents great challenges to the bandwidth and power dissipation of the caches [28][30].

For the reliable tag array design, a fault behavior of the CAM (content addressable memory) tags has been studied and single-error tolerant solutions were provided in [31]. A functional level framework was also proposed in [32] for implementing a fault-tolerant/self-checking CAM architecture, with a focus on CAM cell designs. Compared to their hardware circuit solutions, this work focuses on the microarchitecture design of the reliable cache. Biswas et al. [33] presented some initial efforts on vulnerability analysis of the tag array. However, they did not provide any direct reliability optimization schemes on the tag array. In [34], a caching address tag (CAT) scheme was proposed to reduce the area cost of the on-chip caches. Due to the CAM implementation of the pointer part in the tag side, their scheme will incur extremely high energy consumption caused by the CAM search operation if it is adopted for reliability improvement.

1.4.2 Reliable Design of Register Files

Previous work [35] has exploited utilizing free registers or predicted dead registers to maintain a replica of the value in the register file to increase its error resilience. Recent work [36] studied the trade-offs between performance and reliability of the register file when over-clocking is applied to increase the operation frequency. In [37], compiler-guided techniques were proposed to improve the register file reliability by changing the instruction scheduling and register assignment. Recent work [38] studied a register replication approach by selectively copying register values to the unused physical registers for enhancing reliability. Work [39] proposed to selectively protect registers by generating, storing, and checking the

ECCs of only the most vulnerable registers with useful data, while parity coding is used for all the registers. Different from their work, the proposed in-register duplication scheme is based on the detection and capture of narrow-width register values such that redundant copies are generated within a single 64-bit data item to improve the reliability of the register file system, eliminating the need for copy registers and related hardware enhancements.

1.5 Contribution

The contributions of this dissertation consist of four parts: (1) cache lifetime models to characterize the vulnerability of the on-chip caches, (2) optimizing schemes to improve the on-chip caches reliability, (3) the tag replication buffer for enhancing cache tag array reliability, (4) self-adaptive data caches for soft-error reliability, and (4) reliable register files with narrow-with duplication.

1.5.1 Cache Lifetime Models for Reliability

In this dissertation, detailed lifetime models are developed for L1 data and instruction caches to capture all possible activities of all data items. A data item under consideration can be at different granularities such as cacheline, sub-block, word, half word, byte, or even bit. The new lifetime models distinguishes among different lifetime phases for each data item according to the previous activity and the current one, and further categorizes them into two groups, *vulnerable* and *non-vulnerable* phases. A vulnerable phase is characterized by the fact that any error occurring during this phase has the potential to propagate either to the CPU (by load operations) or to the L2 cache (via a dirty line writeback). The cache temporal vulnerability factor (TVF) is defined as the percentage of data items present in vulnerable phases over all possible data items that the cache can hold, an average along the time axis. Therefore, the lifetime vulnerability factor indicates how reliable the cache is. A smaller value of TVF implies that the cache is more resilient to soft errors.

1.5.2 Optimizing Schemes to Improve On-Chip Caches Reliability

Based on the fact that the WPL (lifetime phase between the last write and the replacement without any read in between) vulnerable phase contributes the most to TVF in the data cache, the multiple-dirty-bits (MDB) scheme is proposed to reduce the WPL vulnerable phase as well as the energy consumption during the writeback. To further reduce the the second largest vulnerable phase, RR (lifetime phase between two consecutive reads of a clean data item), a clean cacheline invalidation (CCI) scheme is proposed. A combined scheme that incorporates the previous DTEWB (Dead Time based Early Write Back) [40][41] and NWVC (Narrow Width Value Compression) [42][43] schemes are proposed to reduce the overall TVF of the data cache. For the instruction cache, a variation of the cacheline scrubbing (CS) with CCI is proposed to achieve a lower TVF with the minimized performance and energy overheads.

1.5.3 Tag Replication Buffer for Enhancing Cache Tag Array Reliability

Exploiting the address locality of memory accesses, this dissertation proposes a Tag Replication Buffer (TRB), a small buffer that captures and maintains the replicas of frequently accessed tag entries, to enhance the reliability of the tag array in the on-chip data cache. A detailed design space exploration is performed for the TRB implementation and several optimized schemes are proposed to improve the tag array reliability as well as to reduce the area and energy overheads of the TRB. To further improve the protection effectiveness and the provided reliability of the TRB, a selective TRB scheme that only duplicates tag entries for dirty cachelines is proposed. In order to provide a comprehensive evaluation on the reliability of the cache tag array, the dissertation conducts a cache tag vulnerability factor analysis and propose a refined cache tag reliability evaluation metric DOR (detected without replica) TVF that combines the TVF and access-with-replica (AWR) analysis. Based on the DOR-TVF analysis, a new TRB scheme with early write-back (TRB-EWB) triggered by the TB (tag buffer) replacement is proposed, which can achieve a 100% AWR

rate and a zero DOR-TVF with a minimum performance and energy overhead.

1.5.4 Self-Adaptive Data Caches for Soft-Error Reliability

For the systems working in the changing operating environments, a self-adaptive reliable data cache is proposed to dynamically adapt its employed reliability schemes to maintain a target reliability. This self-adaptive data cache is implemented with three levels of error protection schemes, a monitoring mechanism, and a control component that decides whether to upgrade, downgrade, or keep the current protection level based on the feedback from the monitor. The self-adaptive data cache is evaluated by injecting errors with a changing soft error rates to prove that it can achieve similar reliability to a cache protected by the most reliable scheme, while maintaining the minimized performance and energy overheads.

1.5.5 Reliable Register Files with Narrow-With Duplication

To improve the reliability of the register files in the microprocessor, this dissertation proposes to make a duplication of the value within the same data item by exploiting narrow-width register values. This in-register duplication (IRD) does not require additional copy registers. The datapath pipeline is augmented to efficiently incorporate parity encoding and parity checking such that error recovery is seamlessly supported in IRD and the parity checking is overlapped with the execution stage to avoid increasing the critical path. IRD can achieve extremely high read-with-duplicate (RWD) and error detection/recovery rates under heavy error injection with a negligible power overhead.

1.6 Organization of the Dissertation

The rest of the dissertation is organized as follows. The next chapter presents the experimental setup used in this work. Chapter 3 discusses the proposed new lifetime model for

the cache vulnerability analysis and the improving schemes to enhance the reliability of the on-chip caches. Chapter 4 presents the tag replication buffer for reliable cache tag array design. Chapter 5 describes the design of a self-adaptive data cache for soft-error reliability. A reliable register file by exploiting and duplicating the narrow-width value is proposed in Chapter 6. Chapter 7 gives conclusions and describes the directions of future work.

CHAPTER 2

EXPERIMENTAL SETUP

2.1 Simulated Processor

The simulator used in this work is derived from SimpleScalar V3.0 [44]. In Chapter 3 and 5, it is modified to model a contemporary high-performance microprocessor similar to Alpha 21364 [45]. In the new simulator, the original RUU (register update unit) structure is replaced by a separated integer issue queue, a floating-point issue queue, an integer register file, a floating-point register file, and an active list (a.k.a. the re-order buffer). A MIPS R10000 [12] style register renaming scheme is adopted in the implementation. There is no separate architectural/logical register file. Committing the current instruction frees the physical register that is being renamed to the immediately previous instruction with the same destination/result logical register. The new simulator also implements the tournament branch predictor (the local predictor uses 2-bit counters) used in Alpha 21364 microprocessors [45]. Table 2.1 gives the detailed configuration of the simulated microprocessor in Chapter 3, 4, and 5. Cacti 3.2 [46] and Wattch [47] are used for energy profiling during the simulation.

Since Chapter 6 focuses on the reliable register file design, the simulator is further modified to model a modern microprocessor similar to Alpha 21464 [48], in which the integer register file size is set to 128 to simulate the register pressure in SMT (simultaneous multithreading) environments. Table 2.2 shows the modified processor core different from Table 2.1

2.2 Benchmarks

For experimental evaluation, this work uses the SPEC CPU2000 benchmark suite [49] compiled for the Alpha instruction set architecture using the "-arch ev6 -non_shared" op-

Table 2.1 Parameters for the simulated microprocessor in Chapter 3 and 5.

Processor Core	
Int/FP issue queue	20/15 entries
Load/Store Queue	64 entries
Active list (ACL)	80 entries
Int/FP Register File	80/72 registers
Datapath width	4 instructions per cycle
Function Units	4 IALU, 1 IMULT/IDIV
	2 FALU, 1 FMULT/FDIV/FSQRT
	2 MemPorts
Branch Predictor	
Branch Predictor	Tournament predictor with a 4K meta-table, a 4K bimodal predictor table, and a 2-level gshare predictor with 12-bit history
	2048-entry, 2-way BTB, and 32-entry RAS
Memory Hierarchy	
L1 I/DCache	64KB, 2 ways, 64B blocks, 2 cycle latency
L2 UCache	4MB, 8 ways, 128B blocks, 12 cycle latency
Memory	225 cycles first chunk, 12 cycles rest
TLB	Fully-assoc., 128 entries, 30-cycle miss penalty
Technology Parameters	
Vdd	0.9V
Clock frequency	3GHz
Technology	70nm

Table 2.2 The Modified Processor Core in Chapter 6.

Processor Core	
Int/FP issue queue	128 entries
Load/Store Queue	256 entries
Active list (ACL)	512 entries
Int/FP Register File	128/512 registers
Datapath width	8 instructions per cycle
Function Units	8 IALU, 2 IMULT/IDIV, 4 FALU
	2 FMULT/FDIV/FSQRT, 4 MemPorts

tion with "peak" tuning. The reference input sets is used for this study. Each benchmark is first fast-forwarded to its early single simulation point (*gap* and *ammp* use the standard single simulation point instead of the very large early single simulation point) specified by SimPoint [50]. The last 100 million instructions during the fast-forwarding phase are used to warm-up the caches if the number of skipped instructions is more than 100 million. Then, the next 100 million instructions are simulated in detail. The description of the simulated SPEC CPU2000 benchmarks is shown in Table 2.3.

Table 2.3 SPEC CPU2000 benchmark suite

Benchmark	Language	Fast Forword	Category
CINT2000 (Integer Benchmarks)			
164.gzip	C	300M	Compression
175.vpr	C	7100M	FPGA Circuit Placement and Routing
176.gcc	C	10900M	C Programming Language Compiler
181.mcf	C	31600M	Combinatorial Optimization
186.crafty	C	0M	Game Playing: Chess
197.parser	C	1600M	Word Processing
252.eon	C++	1800M	Computer Visualization
253.perlbmk	C	100M	PERL Programming Language
254.gap	C	67600M	Group Theory, Interpreter
255.vortex	C	5700M	Object-oriented Database
256.bzip2	C	900M	Compression
300.twolf	C	3100M	Place and Route Simulator
CFP2000 (Floating Point Benchmarks)			
168.wupwise	Fortran 77	58400M	Physics / Quantum Chromodynamics
171.swim	Fortran 77	58400M	Shallow Water Modeling
172.mgrid	Fortran 77	500M	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	1800M	Parabolic / Elliptic Partial Differential Equations
177.mesa	C	8900M	3-D Graphics Library
178.galgel	Fortran 90	67600M	Computational Fluid Dynamics
179.art	C	6700M	Image Recognition / Neural Networks
183.equake	C	19400M	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	13600M	Image Processing: Face Recognition
188.ammp	C	67600M	Computational Chemistry
189.lucas	Fortran 90	3500M	Number Theory / Primality Testing
191.fma3d	Fortran 90	29800M	Finite-element Crash Simulation
200.sixtrack	Fortran 77	8200M	High Energy Nuclear Physics Accelerator Design
301.apsi	Fortran 77	4600M	Meteorology: Pollutant Distribution

CHAPTER 3

ON-CHIP CACHE VULNERABILITY ANALYSIS AND OPTIMIZATION

3.1 Introduction

Most of the previous works have studied tradeoffs between performance, energy consumption, area overheads and the achieved cache reliability for their proposed schemes [51][52][53][28][40][29][54][55][56][57][41]. Therefore, a more systematic study of cache vulnerability is needed. Such a study could provide enough insight into cache reliability behavior, that the designer could take advantage of to design highly cost-effective reliable caches. Recent papers [41][55][51][58][59][60] present some initial efforts towards such a cache vulnerability analysis. However, their cacheline- or word-based vulnerability characterization used some simple generation model [61] that could not explore the temporal vulnerability of the cache, i.e., how different lifetime phases of the cache data contribute to vulnerability. This temporal information is of critical importance in determining *which data* in the cache should be protected at *what time* with *which protection schemes*, in order to achieve high reliability. This dissertation targets at providing such a bridge from perception to practice in designing reliable caches.

For the aforementioned purpose, a detailed lifetime model is proposed for the data arrays in the L1 data and instruction caches, as the first step, to capture all possible activities that could involve these data items. A data item under consideration can be at various granularities such as cacheline, sub-block, word, half word, byte, or even bit. In the data cache, the new lifetime model distinguishes among nine lifetime phases for each data item according to the previous activity and the current one, and further categorizes them into two groups, *vulnerable* and *non-vulnerable* phases. A cache vulnerable phase is defined as the phase during which any occurring error has the potential to propagate either to the CPU

datapath (by load operations) or to the L2 cache (via a dirty line writeback). The cache temporal vulnerability factor (TVF) is defined as the percentage of data items present in vulnerable phases over all possible data items that the cache can hold, an average along the time axis.

To derive highly cost-effective reliability schemes for on-chip cache memories, new design methodologies driven by TVF characterization and analysis are proposed. First, a cacheline-based TVF analysis is performed on the entire data array. The results show that the vulnerable phase *write-replace* (WPL, the lifetime phase between the last write and the replacement without any read in between) contributes the most to TVF in the data cache. A writethrough data cache can effectively eliminate this phase by immediately writing back the data to the L2 cache after a store operation. However, the excessive accesses to the L2 cache degrade the performance and increase the energy consumption. An alternative to solve this problem is to early write back dirty lines, such as the deadtime based early writeback (DTEWB) scheme in [40]. Further analysis indicated that this cacheline-based analysis cannot fully capture the nature of CPU accesses to the data cache. Since the unit size for data cache accesses is the byte, different bytes in the same cacheline may be in different lifetime phases at any given time, e.g., some bytes in a dirty cacheline may be in the clean state. Treating all the bytes in a cacheline equally may lead to inaccurate calculation of the cache TVF. It concludes that fine-grain (e.g., byte-based) lifetime models should be considered for more accurate TVF characterization. Based on the byte-level analysis, the work also proposes the multiple-dirty-bits (MDB) scheme to further reduce the WPL vulnerable phase as well as the energy consumption during the writeback.

After WPL optimization, the vulnerable phase *read-read* (RR, the lifetime phase between two consecutive reads of a clean data item) with the potential to propagate errors to the CPU raises as another major part in the vulnerability factor of the data cache. Experimental study shows that a 87.8% majority of RRs have a short time interval (<= 0.5K cycles) and account for only 15.5% of the overall RR vulnerable intervals. Based on this

observation, a clean cacheline invalidation (CCI) scheme is proposed to invalidate clean lines after being idle for a certain amount of time. Note that this scheme may result in performance loss when the invalidated cachelines are accessed lately by the CPU. However, by carefully choosing the invalidation interval, the induced performance overhead can be controlled to a minimum. The further analysis on data items in cachelines shows that a significant portion of stored data is narrow width data, which complies with previous research findings [42][43][62][63][64][65][66][67][68][69][70][71]. The work integrates a narrow-width value compression (NWVC) scheme with the lifetime models for further reducing the WPL, RR, and other vulnerable phases. The *Combined* scheme with DTEWB, MDB, CCI and NWVC achieves a significantly reduced TVF of 3.5% compared to the original 39.2% of the data array in the data cache, at a minor performance loss of 0.7%.

Different from the data cache, the instruction cache is read-only (from the datapath side) and this read-only activity dramatically simplifies the lifetime model for the data array in the instruction cache. In this lifetime model, RR is the only vulnerable phase. To optimize this RR phase, the clean cacheline invalidation (CCI) scheme is explored, similarly to the data cache. However, the experimental results show that the performance loss due to the instruction cache CCI is much higher than for the data cache CCI. This is mainly because of the high pipeline stall penalty due to increased instruction cache misses incurred by the CCI scheme. To reduce the performance overhead, a variation of the cacheline scrubbing (CS) scheme is proposed to scrub idle clean lines from the L2 cache. While reducing the RR phase without significantly impacting the performance, the scrubbing scheme dramatically increases the accesses to the L2 cache. Consequently, the dissertation further proposes to combine the CCI and CS schemes to optimize the RR phase while minimizing the performance and energy overheads. The evaluation results show that the CS-CCI scheme effectively reduces the TVF of the instruction cache data array from 19.9% to 5.3% at a 0.9% performance loss and a 29% energy increase in the L2 cache.

Previous work [31] has studied the fault behavior of CAM (content addressable

memory) tags and provided single-error tolerant solutions to protect them. A functional level design framework was also proposed in [32] for implementing a fault-tolerant/self-checking CAM architecture, with a focus on CAM cell designs. To provide a comprehensive view of cache reliability, this dissertation also strives to study the reliability behavior in the tag array for both the data and instruction caches. During an access to a set-associative cache, all tags in the same set are read out and compare simultaneously with the tag in the CPU-issued address, which puts the tags of valid cachelines into a vulnerable phase. However, if the single bit error model is assumed, Hamming-distance-one analysis (HDO) [33] can be employed to dramatically reduce the TVF of the tag array. A new lifetime model for the tag array to extend the Hamming-distance-one analysis is proposed. Furthermore, the effect of the early write back and clean cacheline invalidation schemes is studied on optimizing the TVF of the tag arrays. In summary, the tag array TVF is reduced to 7.72% and 0.08% for the data and instruction caches from their original 46.7% and 0.3%, respectively.

3.2 Temporal Vulnerability Factor of the Data Array in Data Caches

3.2.1 A General Lifetime Model of the Data Array

In this section, the detailed lifetime model of the data array is introduced for the purpose of vulnerability characterization. A cacheline is first brought into the L1 data cache on a read or write miss. The cacheline will be accessed at most a couple of times, either by reads or writes, and then may wait for a long time before it is replaced [61]. Such cacheline generation information can be exploited for cache leakage optimization [61]. However, it is not sufficient for reliability analysis. Notice that not all of the soft errors occurring in the data cache will result in a failure. If errors occur in the data field of invalid cachelines, they are simply masked off by the invalid bits and have no impact on the correctness of the execution. Errors occurring in the data field of clean cachelines after the last read are

similarly masked off by the dirty bit (= 0) and, therefore, are discarded at replacement. Other errors may be overwritten by subsequent writes before a CPU read or a write back to the L2 cache, thus presenting no harm to reliability. In the new model, the lifetime of a data item, e.g., a cacheline, is divided into the following phases: WRR, RR, WR, WPL, WRPL, RPL, RW, WW, and Invalid. They are:

- WRR: lifetime phase between two consecutive reads of a dirty data item,
- RR: lifetime phase between two consecutive reads of a clean data item,
- WR: lifetime phase between a write and its first read,
- WPL: lifetime phase between the last write and the replacement without any read in between,
- WRPL: lifetime phase between the last read and the replacement of a dirty data item,
- RPL: lifetime phase between the last read and the replacement of a clean data item,
- RW: lifetime phase between the write and the last read before the write,
- WW: lifetime phase between two consecutive writes without any read in between,
- Invalid: lifetime phase when the data item is in the invalid state.

Figure 3.1 shows the correlation among these lifetime phases for typical data cache activities. In this dissertation, a *vulnerable phase* is defined as being a lifetime phase in which errors may propagate out of the cache, either to the CPU or to the lower level memory hierarchy, i.e., L2 caches. Clearly, the first five phases, WRR, RR, WR, WPL, and WRPL, are vulnerable because errors occurring in phases WRR, RR, or WR will have the opportunity to be read by the CPU, and errors occurring in phases WPL or WRPL will have the opportunity to propagate to the L2 cache. RPL and Invalid are *non-vulnerable phases* since errors occurring during these two phases will be discarded or ignored. However, phases RW and

WW present different vulnerability behavior for data items at different granularities. If the data item under consideration is a byte, RW and WW are non-vulnerable phases. Otherwise, RW and WW are *potential vulnerable phases*. This vulnerability characteristic of RW and WW are discussed in the following section.

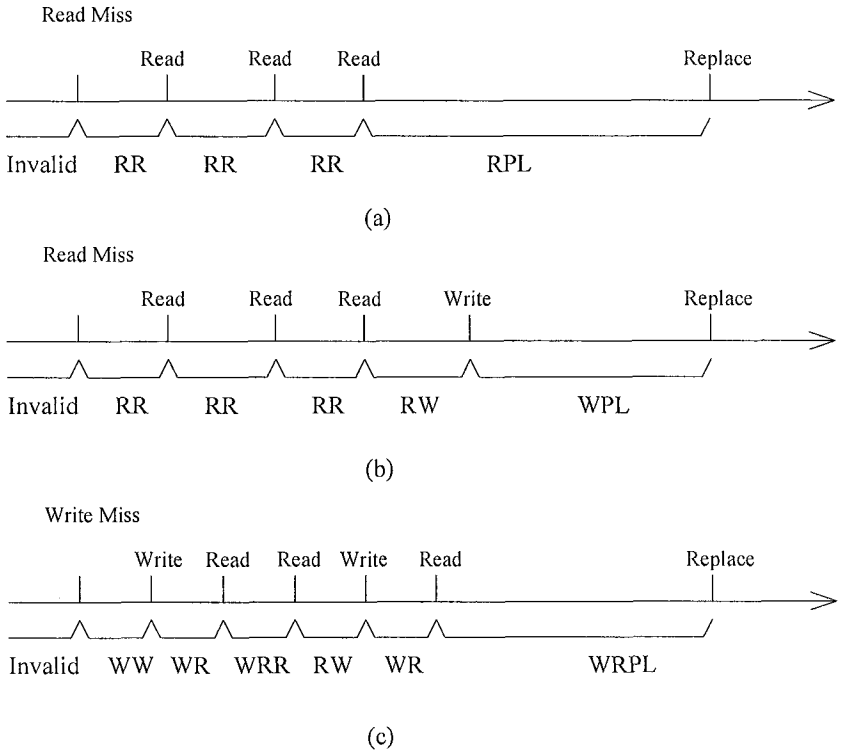


Figure 3.1 The lifetime of a cacheline with respect to various access activities.

3.2.2 Temporal Vulnerability Factor (TVF)

The cache temporal vulnerability factor (TVF) introduced in this work is defined as the average rate of data items in vulnerable phases over the total data items that the cache can accommodate during the execution. TVF can be calculated as follows:

$$TVF_{Cache} = \frac{\sum_{i}^{n}(data\_item_i * \sum_{j} vul\_phase_j)}{\sum(data\_item * Exec\_Time)}, \tag{3.1}$$

where $data\_item_i$ can be a cacheline, a word, or a byte, vul_phase_j is the time of j^{th} vulnerable phase of $data\_item_i$, and $Exec\_Time$ is the total time simulated for the benchmark.

The vulnerability factor is used to evaluate the reliability of the data cache. If the data cache has a high vulnerability factor, it has more data items in vulnerable phases during the execution, thus is more vulnerable to soft errors. Therefore, a main objective in designing a reliable data cache is to reduce its vulnerability factor. Notice that the temporal vulnerability factor (TVF) is different from the architectural vulnerability factor (AVF) [72] of the data cache. Since soft errors induced during the vulnerable phases in the data cache only present the potential to crash the execution or the lower memory hierarchies, TVF defines the upper bound on AVF and can be estimated more accurately than AVF. Further, TVF is also different from the critical time [59] in that the critical time is calculated based on the word-level vulnerability analysis while TVF is derived from a flexible lifetime model for detailed vulnerability analysis at different granularities, e.g., a cacheline, a word, or a byte.

3.2.3 Data Array Vulnerability Characterization

In this section, both cacheline based and byte based vulnerability characterization are performed, and the deficiency of the cacheline based scheme is analyzed.

A Cacheline based Characterization

In conventional cache designs, each cacheline is associated with a dirty bit indicating whether it is a clean line or a dirty one. The dirty bit is set once the cacheline is written by the CPU. In writeback caches, the dirty cacheline is written back to the lower level caches upon replacement, as a single unit. Thus, it is very straightforward to perform data cache vulnerability analysis based on the cacheline lifetime information [41]. Applying the lifetime model, the data item here will be a cacheline. Obviously, the initial phase of all

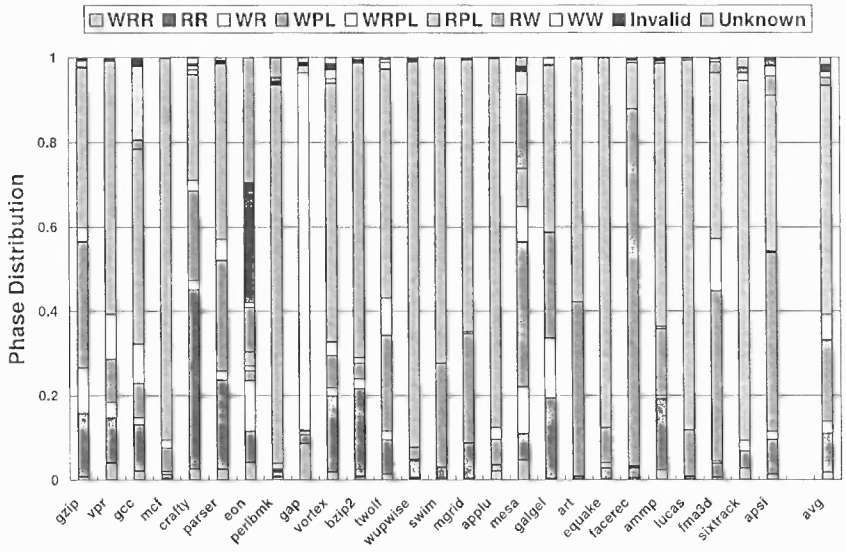


Figure 3.2 The lifetime distribution of the data array in the data cache with the 64-byte cacheline.

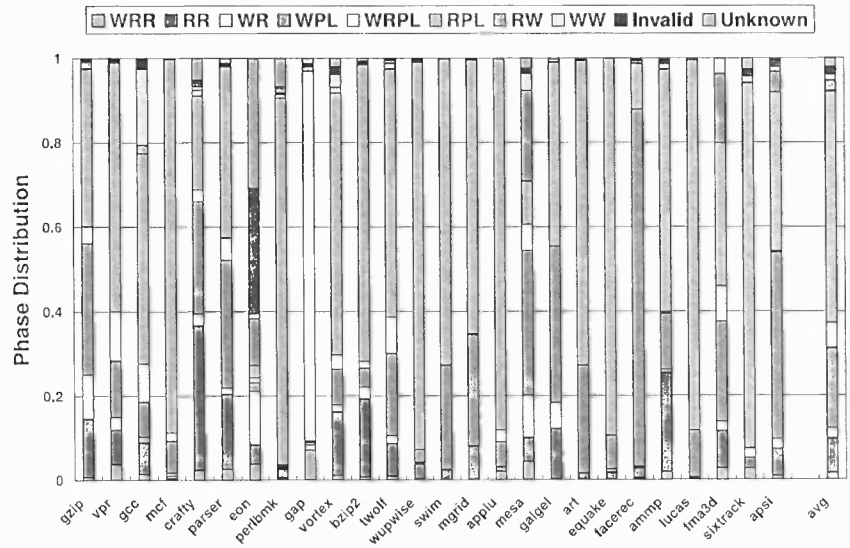


Figure 3.3 The lifetime distribution of the data array in the data cache with the 32-byte cacheline.

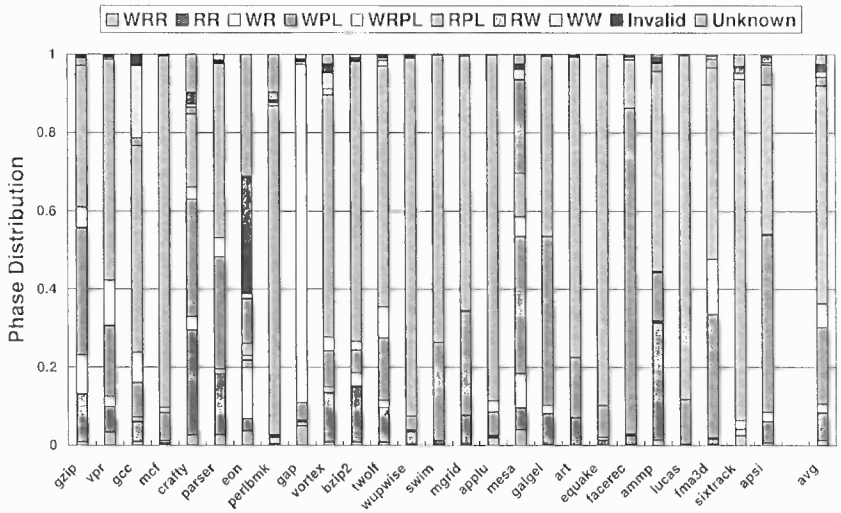


Figure 3.4 The lifetime distribution of the data array in the data cache with the 16-byte cacheline.

cachelines in the data cache is `Invalid`. Upon different CPU access activities, the cachelines enter different phases, i.e., `RR`, `RW`, `WW`, `WR`, `WRR`, `RPL`, `WPL`, or `WRPL`, at different time points.

First, the impact of the cacheline size on the lifetime distribution and thus the vulnerability factor of the data array are analyzed. Figure 3.2, 3.3, and 3.4 shows the distribution of the cacheline lifetime under three cacheline sizes, namely 64, 32, and 16 bytes. For line-based lifetime analysis, previous research [41] considered only `WRR`, `RR`, `WR`, `WPL`, and `WRPL` (phase names here may be different from [41]) as vulnerable and all other phases as non-vulnerable. However, this is not accurate. As discussed early in Chapter 3.2.1, the other two phases `RW` and `WW` have the potential to propagate errors to either the CPU side or the L2 caches. A scenario involving such an error propagation to L2 caches is illustrated in Figure 3.5. If errors hit the clean bytes of a cacheline before a write updates other bytes, the error-corrupted clean bytes may also be written back to the L2 cache at a later replacement. However, if the erroneous clean bytes are overwritten by subsequent writes before CPU reads or a writeback operation to L2 caches, they present no harm to the correctness of program execution. Thus, `RW` and `WW` are classified as potential vulnerable phases.

Although the temporal vulnerability factor decreases as the cacheline size is re-

duced from 64 bytes to 16 bytes, as shown in Figure 3.2, 3.3, and 3.4, the improvement is not significant. The TVF values for the data cache with 64-byte, 32-byte, and 16-byte cachelines are 39.2%, 37.3%, and 36.3%, respectively. Moreover, if simply reducing the cacheline size, the performance normally degrades because of the spatial locality property. For the default line size (64 bytes), Figure 3.2 shows that the vulnerable phases account for about 39.2% of a cacheline's lifetime, among which `WPL` and `RR` contribute about 19.3% and 9.3%, respectively. The two potential vulnerable phases `RW` and `WW` together account for 3.2%. The only truly non-vulnerable phases of the cacheline are `RPL` and `Invalid`. Note that `Unknown` represents a phase where the state cannot be determined because of the limited simulation time. `RPL` represents the largest part in the lifetime, around 54.3%, which is non-vulnerable. Therefore, to improve cache reliability, the time spent by a cacheline in the `WPL` and `RR` phases needs to be reduced.

Vulnerability Characterization at Fine Granularities

Since the unit size for CPU data accesses is in the byte, a write operation does not update the entire cacheline. This characteristic of the data cache accesses makes different bytes in the same cacheline into different phases during the execution. For example, in a clean cacheline, if a byte write operation occurs, it will only update a particular byte in that cacheline and bring the entire cacheline to the dirty state. However, there is only one byte

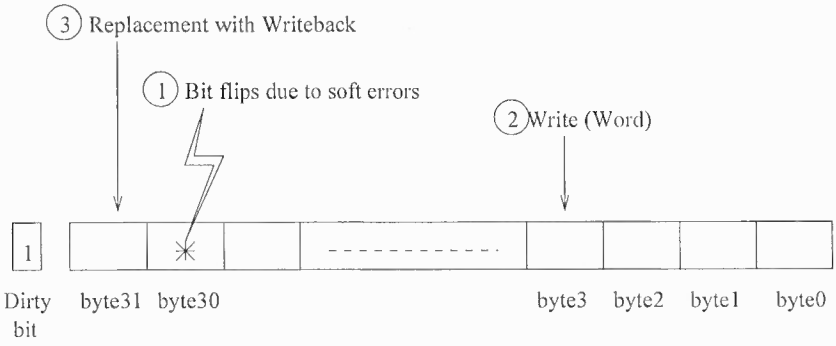


Figure 3.5 A scenario of cache accesses and error occurrences that contribute `RW` or `WW` to vulnerable phases.

in the dirty state after the write, while others may still be in the clean state. Therefore, it is not accurate and efficient to assume that these clean bytes in a dirty cacheline are actually in the dirty state. Another problem with the line-based characterization is the inaccuracy in RW and WW profiling, as illustrated in Figure 3.5. For more accurate data cache vulnerability characterization, the lifetime analysis is performed while the data item granularity is scaled down to a word (8-byte) or a byte. Each data item can only be in one particular phase at a given time.

Figure 3.6 and 3.7 show the lifetime distribution based on word-level and byte-level characterization. The TVF based on word-level characterization is 25.7%, compared to 39.2% for cacheline-level analysis. This TVF value is further reduced to 19.9% for byte-based characterization. It is important to note that there is no potential vulnerable phase in the byte-based lifetime model. RW and WW are then true non-vulnerable phases as any error that occurred in a particular byte should be cleaned/overwritten by the subsequent write to the same byte. However, in the word-based lifetime model, RW and WW still contribute to potential vulnerable phases because of the same reason as for the cacheline based model. Table 3.1 summarizes the comparison of the results by using different granularities for vulnerability characterization.

Table 3.1 The comparison of vulnerability characterization at different granularities.

Granularity	Cacheline	Word	Byte
Vul. Phase	39.2%	25.7%	19.9%
Potential Vul.	3.2%	3.0%	0%

3.2.4 The Impact of Different Cache Write Policies

Write Through vs. Write Back

From the cacheline based lifetime model, phase WPL alone contributes about 19.3% towards the 39.2% temporal vulnerability factor of the data array. A straightforward solution

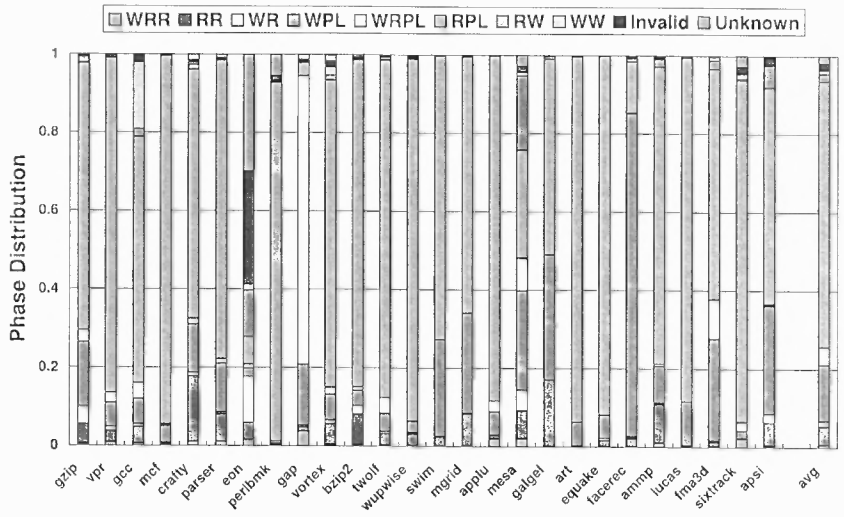


Figure 3.6 The lifetime distribution of the data array in the data cache for the fine granularity data item (64-bit word).

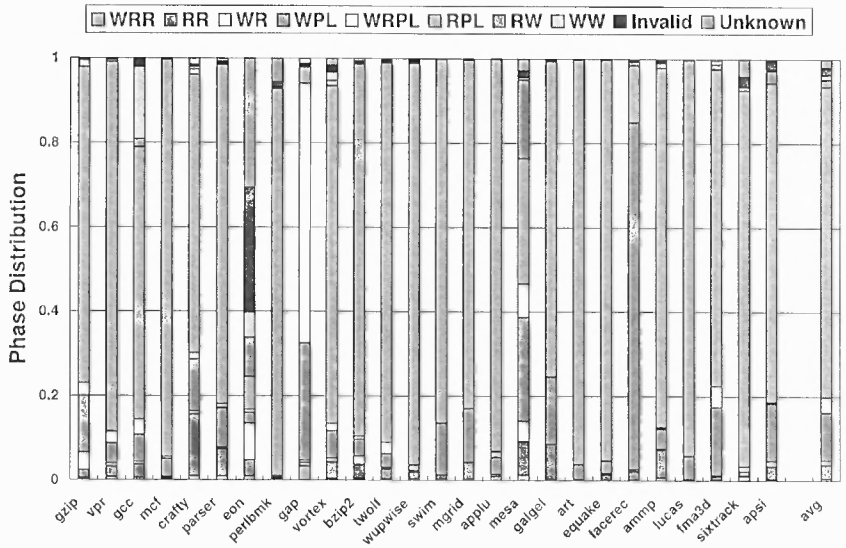


Figure 3.7 The lifetime distribution of the data array in the data cache for the fine granularity data item (8-bit byte).

to reduce the WPL phase is to use a writethrough cache, where a write operation updates both L1 data cache and the L2 cache. In a writethrough cache, phase WPL is effectively converted to the non-vulnerable phase RPL.

However, besides reliability, performance and energy consumption are also key factors to consider in processor design. In general, the writethrough cache needs to update the L2 cache with every write to the L1 data cache. A similar study is performed as in [30][41]. Figure 3.8 compares the performance of writethrough and writeback caches. The writethrough cache is implemented with an 8-entry write buffer in order to alleviate the high pressure on the bandwidth and to reduce the write stalls. For the simulated benchmarks, a writethrough cache incurs a performance loss of 3.8% as compared to a writeback data cache. Furthermore, Figure 3.9 shows that the energy consumption in the L2 cache is more than doubled if the L1 data cache changes its policy from writeback to writethrough. Therefore, for applications that require high performance and low energy consumption, the writeback cache is still preferable.

Multiple-Dirty-Bit (MDB) Data Cache

From the results of line-based and byte-based vulnerability analysis, a major contributor to the TVF in a writeback cache is phase WPL. Based on the same idea as for the byte-level lifetime model, if the clean bytes in a dirty cacheline are not written back to lower level caches during a replacement, any error occurring in clean bytes will be simply discarded. Thus, the WPL phase can be reduced as well as other vulnerable phases, as shown in Figure 3.7. To achieve a similar TVF with the byte-level lifetime model, a multi-dirty-bit (MDB) scheme is proposed.

In conventional data caches, there is only one dirty bit per cacheline. Therefore, identifying whether a particular byte is dirty or not is not possible. In the MDB cache, each byte is provided with a dirty bit and these dirty bits are updated according to the read and write operations. For example, when the CPU writes an 8-byte word to the data cache,

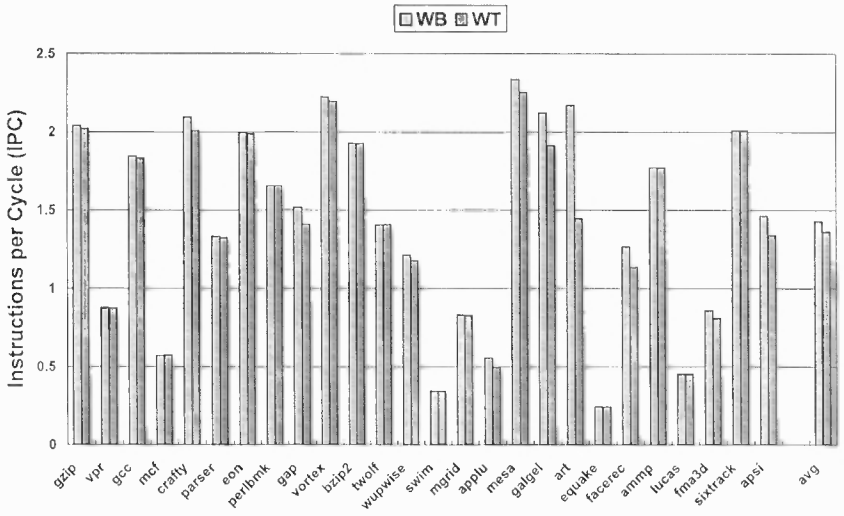


Figure 3.8 The comparison of IPCs between writethrough and writeback caches.

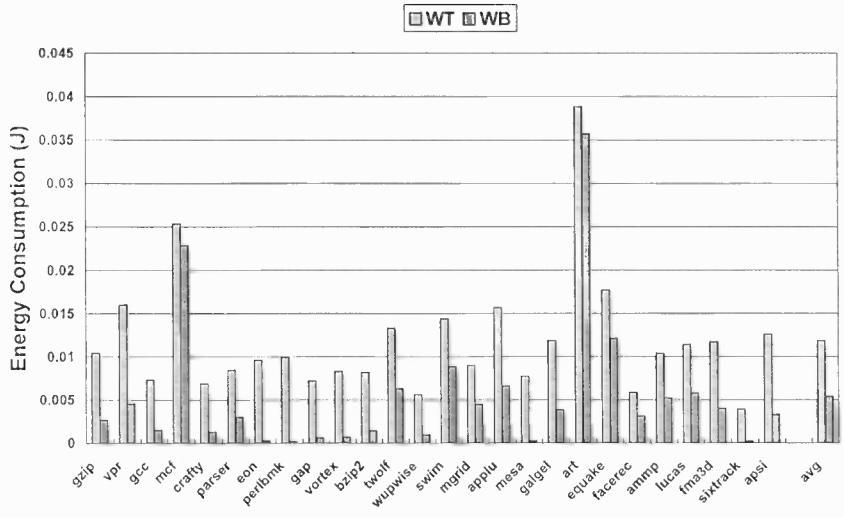


Figure 3.9 The comparison of dynamic energy consumption in the L2 cache for writethrough and writeback data caches.

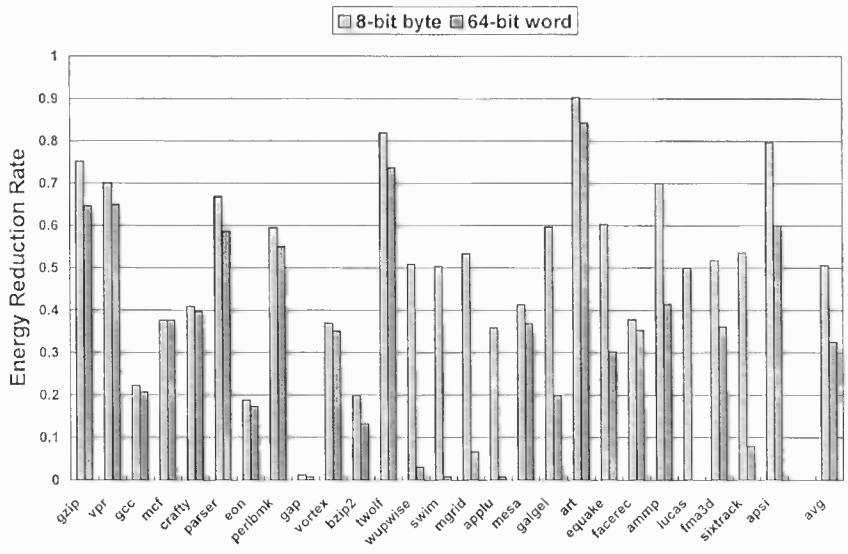


Figure 3.10 The energy savings in cache writeback when applying the MDB scheme at various granularities.

the 8 dirty bits associated with that word in a particular cacheline are set to one. When a dirty line is to be replaced, the dirty bits control which bytes should be written back to the L2 cache. Furthermore, by writing back only dirty bytes in a dirty cacheline, the cache energy consumption can be also reduced, due to reduced energy in data transfer bus and the L2 cache [73]. Notice that the dirty bit of a dirty byte is vulnerable, because if it flips to zero, the dirty byte will not be written back to the L2 cache at replacement time. However, the dirty bit of a clean byte is not vulnerable (when a single bit error model is assumed), because if a soft error flips that dirty bit, it will only cause the clean byte to be written back.

Although the MDB scheme may incur an area overhead similar to that of providing a parity bit for each byte, this scheme has a negligible performance overhead. If the die area is highly constrained, the requirement can be relaxed by using a dirty bit per each word. As the comparison shown in Table 3.1, the vulnerability factor is slightly increased to 25.7% if one dirty bit is associated with each word (8 bytes). On the other hand, the area overhead is reduced to one-eighth of the byte-level dirty bit scheme. Figure 3.10 also shows the energy savings of 50.6% and 32.5%, in the writeback when applying the byte-level and word-level MDB schemes, respectively.

Dead Time based Early Write Back (DTEWB)

Previous work [40][41] proposed early write back schemes to reduce the vulnerable WPL phase while avoiding a dramatic increase in the accesses to the L2 cache. Early write back schemes can be either LRU-based or dead time based [41]. A major design issue in the early write back scheme is to decide when to perform the writeback in order to reduce the WPL phase as well as the accesses to the L2 cache.

The dead time based early write back (DTEWB) scheme [41] could be a solution. A study based on different dead times is conducted. Figure 3.11 shows that the dynamic energy consumption in the L2 cache decreases when the dead time (the idle time interval for dead prediction) increases from 500 to 4K cycles, also comparing to writethrough and writeback caches without DTEWB scheme. Figure 3.12 shows how different dead times affect the vulnerable WPL phase. From these two figures, DTEWB with 2K or 4K cycles can be good choices, which can dramatically reduce the vulnerable phase WPL to 0.8% or 1.4%, at an increase of the energy consumption of 59% or 37% in the L2 cache over the conventional writeback scheme. Notice that from the simulation results, the DTEWB scheme has a negligible performance overhead compared to the writeback cache.

3.2.5 Clean Cacheline Invalidation (CCI)

In the data array of the data cache, the RR phase, which is the time between two reads in a clean cacheline, contributes the second largest share to the vulnerability factor. This share becomes even dominant once the DTEWB scheme is employed, making the RR optimization of critical importance to achieving further improvement of TVF.

The basic idea for RR optimization is to reduce the time that a clean data item, i.e., a cacheline, resides in the data cache by invalidating the cleanlines after being idle for some predefined intervals. Notice that if the clean cacheline is accessed subsequently, additional performance overhead incurs due to the additional cache misses as well as the energy overhead. However, if there is no subsequent access, this invalidation does not cause any

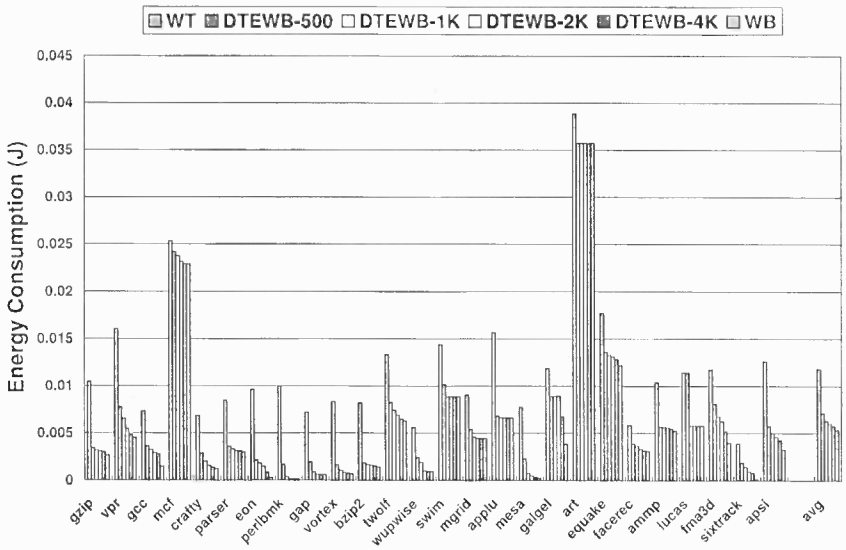


Figure 3.11 The comparison of dynamic energy consumption in the L2 cache at different dead times.

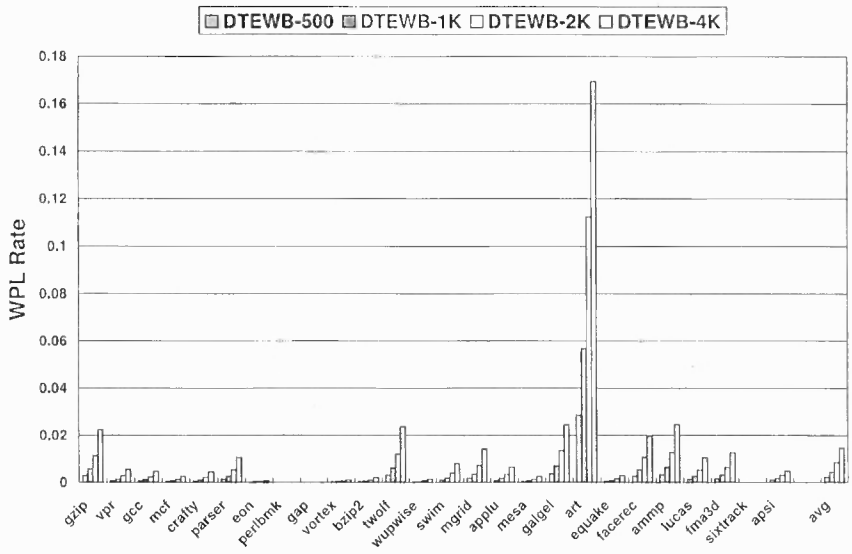


Figure 3.12 The comparison of WPL rates at different dead times.

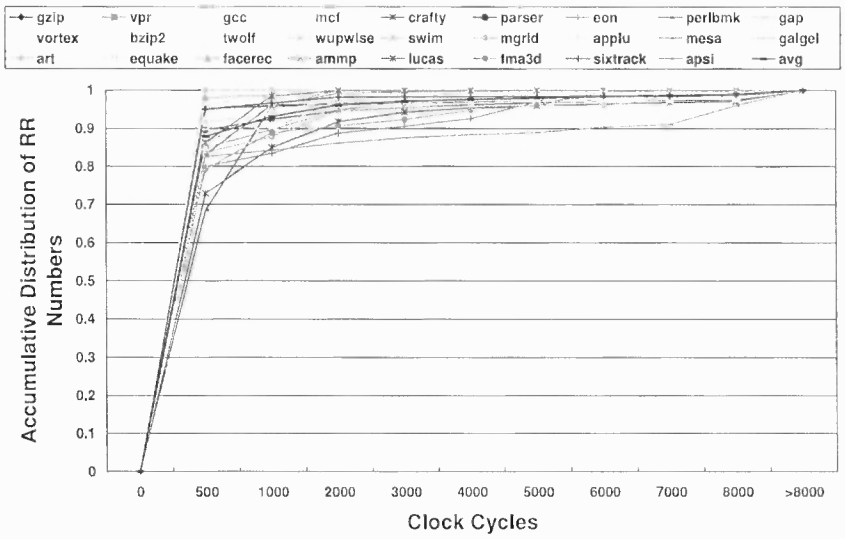


Figure 3.13 Cumulative distribution of the time intervals between two reads (RR) in clean cachelines.

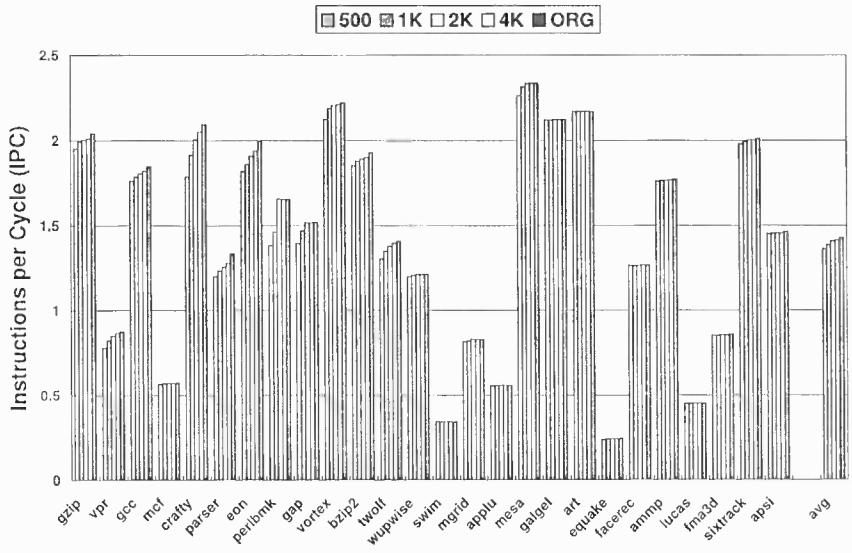


Figure 3.14 The IPC comparison of different invalidation intervals.

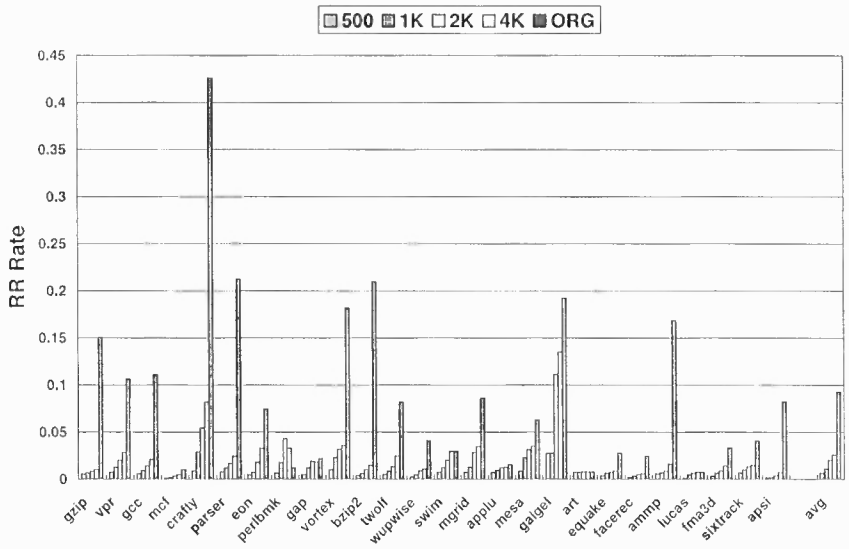


Figure 3.15 (The RR phase comparison of different invalidation intervals. (ORG is the conventional data cache without the invalidation scheme.)

performance loss and neither reduces the RR time. Thus, there is a clear tradeoff between the improved TVF and the performance degradation. The key is to locate such an idle interval for RR such that the RR time reduction can be maximized while the performance loss is minimized.

As shown in Figure 3.13, the number of instances with two consecutive reads to the clean cachelines based on the time interval between the two reads is profiled. The figure shows the cumulative distribution and clearly indicates that most read-read instances, around 87.8% (or 93.1%) of them, have an interval less than 500 (or 1000) cycles. However, the results also show that a small number of read-read instances with intervals (>= 1000 cycles) dominate the overall RR time, 84.5% on the average. The profile results convince us that a scheme capturing only long read-read instances should be able to substantially reduce RR time while keeping the performance loss to a minimum. The experimental results in Figure 3.14 and 3.15 show that 4K cycles is a good choice for this cleanline invalidation. The performance loss is only 0.7% and the RR phase is reduced from 9.3% to 2.6%.

3.2.6 Narrow Width Value Compression (NWVC)

Value awareness can be exploited for reliability enhancement [65][66][67]. Narrow-width as one form of value awareness has been exploited for energy and performance optimization [42][43][62][63][64][68][69][70][71]. In [65][66][67], narrow-width values are duplicated in the register file and the data cache thus improving their reliability via information redundancy. Different from these approaches, this dissertation explores lifetime model driven reliability optimization through narrow width value compression (NWVC). NWVC uses additional narrow tag bits to mask leading zeros in a narrow width value. The narrow tag bit masking can be applied at different granularities, for each 8-bit (byte), 16-bit, 32-bit, or 64-bit (word) data item. For instance, byte-level masking sets the narrow tag bit to one if the corresponding byte contains all zeros. Otherwise, the tag bit is reset to zero. When the data in the cacheline is accessed, the narrow tag bits are checked. If the tag bit is one, it means that the corresponding byte contains all zeros. If any error occurred in this byte, it is simply masked off by the narrow tag bit. Therefore, all the bits in the zero byte are converted into a non-vulnerable state, leading to lower TVF. Moreover, the energy consumption in the data cache can be also reduced with NWVC schemes [74][75] since reading all-zero bytes can be avoided. Figure 3.16 shows the percentage of narrow width values in the L1 data cache at different granularities. For the byte-, 16-bit-, 32-bit-, and word-level narrow tag scheme, the percentage of narrow values is 43.8%, 37.0%, 25.5%, and 17.7% respectively, implying the potential for TVF reduction at similar level. Notice that the narrow tag bit of a non-zero-item is vulnerable, because if it flips to one, the non-zero-item will be mistreated as zero. However, the narrow tag bit of a zero-item is non-vulnerable if the single bit error model is assumed. This is because if the error occurs in this tag bit, the zero-item will be treated as a regular value that is still zero, and will not be affected by that single bit error.

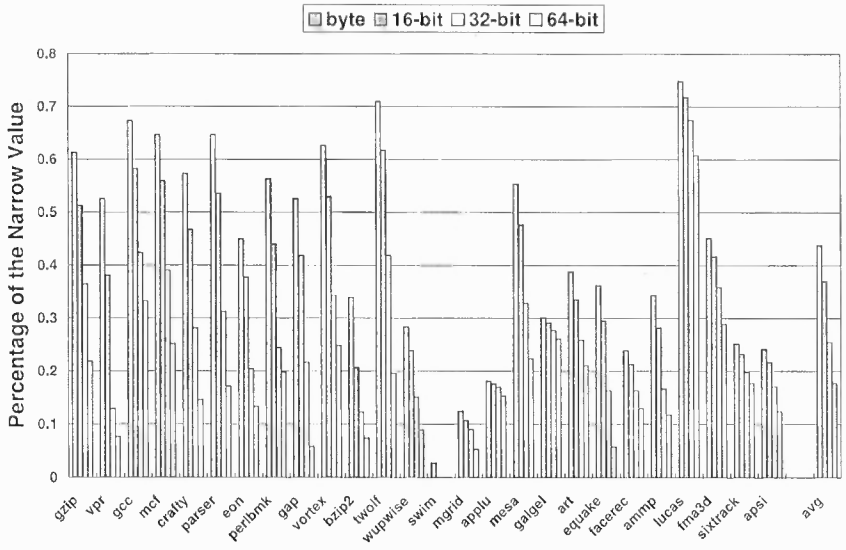


Figure 3.16 The percentage of narrow width values in active cachelines at different granularities.

3.2.7 The Combined Scheme

With the above schemes each targeting at a particular aspect in the lifetime model, this work proposes to evaluate the possibility and effectiveness of combining the DTEWB, MDB, CCI, and NWVC schemes in further improving the data array reliability, i.e., reducing the TVF of the data array. In the evaluation, a 4K-cycle interval is chosen for both deadness prediction and cleanline invalidation. A similar implementation as in the cache decay scheme [61] is used. Each cacheline maintains a 2-bit local counter which is ticked every 1K cycles by a global counter. Both the dead time based early write back scheme [40][41] and the clean cacheline invalidation scheme use the same local counter. The dirty bit of the cacheline controls whether a simple invalidation or an early write back should be performed when the local counter saturates. Considering the hardware and energy overheads, the word-level tag bits for both the MDB and NWVC schemes are chosen, which associate each 64-bit word with two tag bits. For the energy evaluation, all additional tag bits are included. Figure 3.17, 3.18, and 3.19 present the temporal vulnerability factor, performance and cache energy consumption for data caches with and without the combined scheme. By combining DTEWB, MDB, CCI and NWVC, it achieves a vulnerability factor

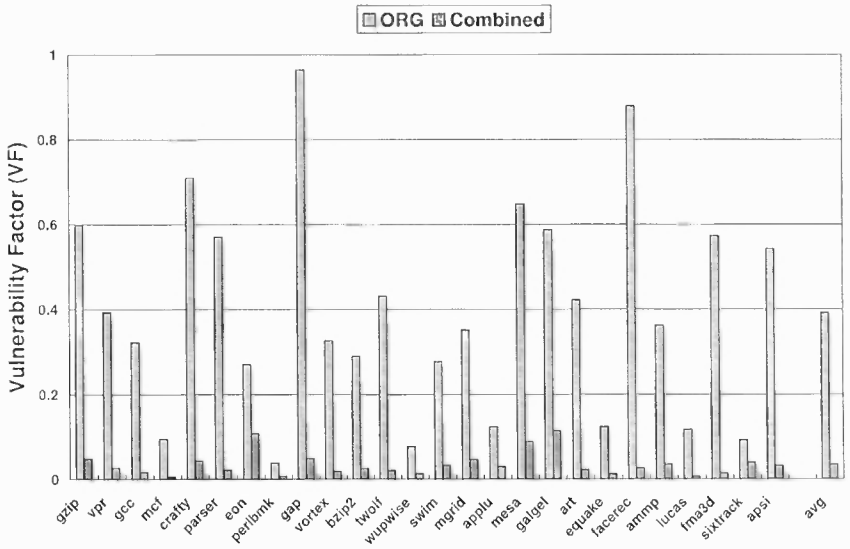


Figure 3.17 The comparison between the data cache employing the combined scheme and the conventional data cache for the temporal vulnerability factor (TVF).

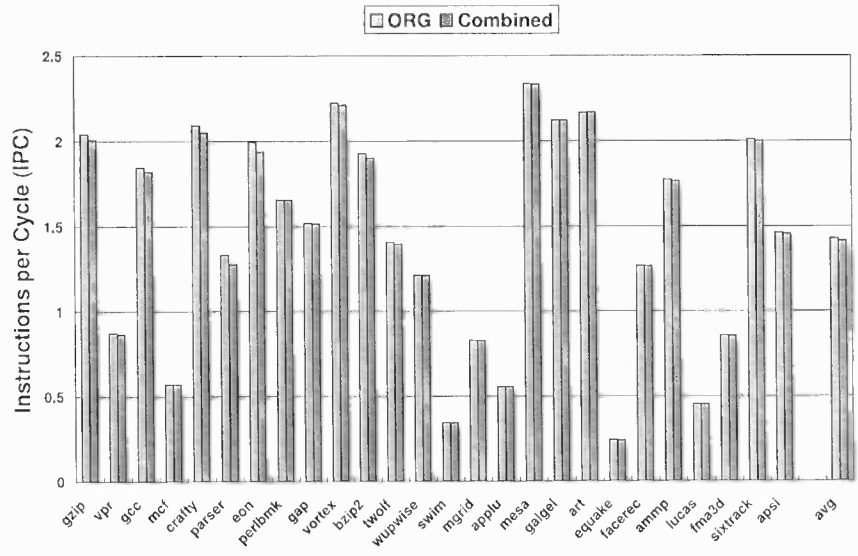


Figure 3.18 The comparison between the data cache employing the combined scheme and the conventional data cache for the performance (IPC) impact.

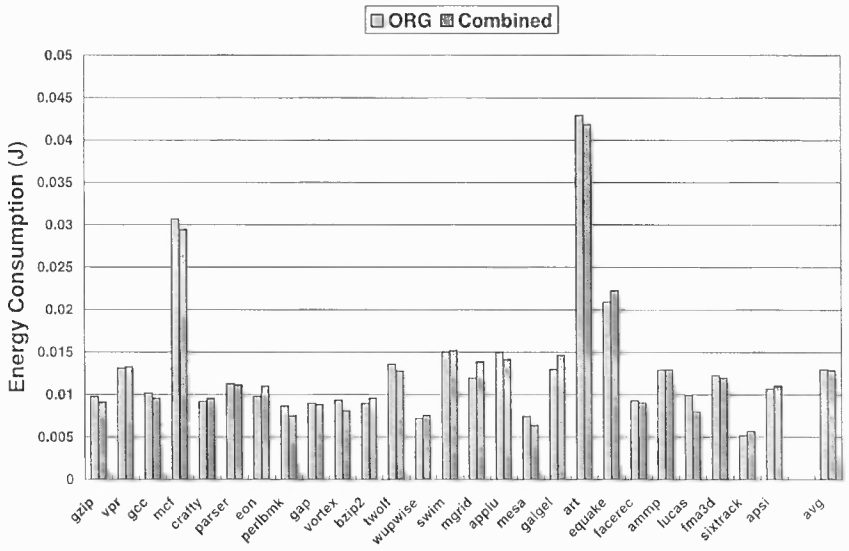


Figure 3.19 The comparison between the data cache employing the combined scheme and the conventional data cache for the energy consumption in L1 data cache and the L2 cache.

as low as 3.5%, which significantly improves the data array reliability in the data cache, at a small performance loss of 0.7%. The total dynamic energy consumption in L1 data cache and L2 caches almost remains the same because of the energy saving from the MDB and NWVC schemes. Table 3.2 summarizes the overhead of the combined scheme.

3.3 Analyzing the Data Array of the Instruction Cache

3.3.1 The Lifetime Model

The lifetime model of the data array in the instruction cache is much simpler compared to that of the data cache, because of the read-only property. There are only three phases in

Table 3.2 Overhead of the combined scheme

Dirty tag bits for MDB (per line)	1 byte (per 64-byte)
Narrow tag bits for NWVC (per line)	1 byte (per 64-byte)
Interval Counters shared by DTEWB and CCI (per line)	2 bits (per 64-byte)
Total storage overhead	3.5% (18b/512b)
Performance loss	0.7%
Energy impact	negligible

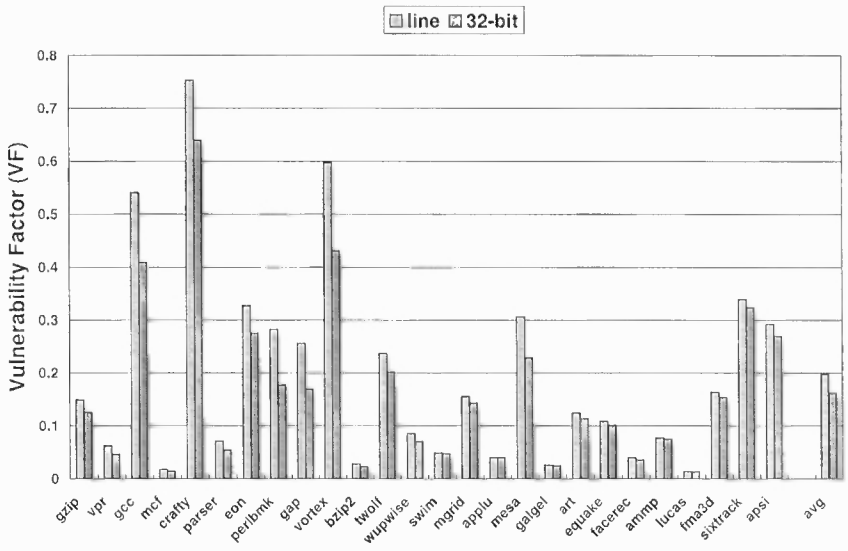


Figure 3.20 The temporal vulnerability factor of the data array in the instruction cache at different granularities of a cacheline or 32-bit data.

this model: `RR`, `RPL`, and `Invalid`, with the same definition as in the model for the data cache. The only vulnerable phase in this model is `RR`, i.e., the time between two reads.

Unlike the data cache, all data items accessed in the instruction cache are of the same size, which is the 32-bit instruction in the simulated processor. Therefore, in a fine-granularity characterization, the 32-bit based model is accurate enough for the data array in the instruction cache. Figure 3.20 shows that the TVF of the data array in the instruction cache is 19.9% and 16.2% for the cacheline based and 32-bit based models, respectively. There is small reduction in the vulnerability factor when applying 32-bit characterization. This can be explained by the access behavior in the instruction cache, which usually exploits the spatial locality for sequential accesses to instructions in the same cacheline.

3.3.2 CCI Scheme for TVF Optimization

Since the `RR` phase is the only contributor to the TVF of the instruction cache data array, the proposed clean cacheline invalidation (CCI) scheme can be the option for TVF optimization. The work evaluates the CCI scheme for the instruction cache with the invalidation interval ranging from 1K cycles to 16K cycles. The results shown in Figure 3.21 and 3.22

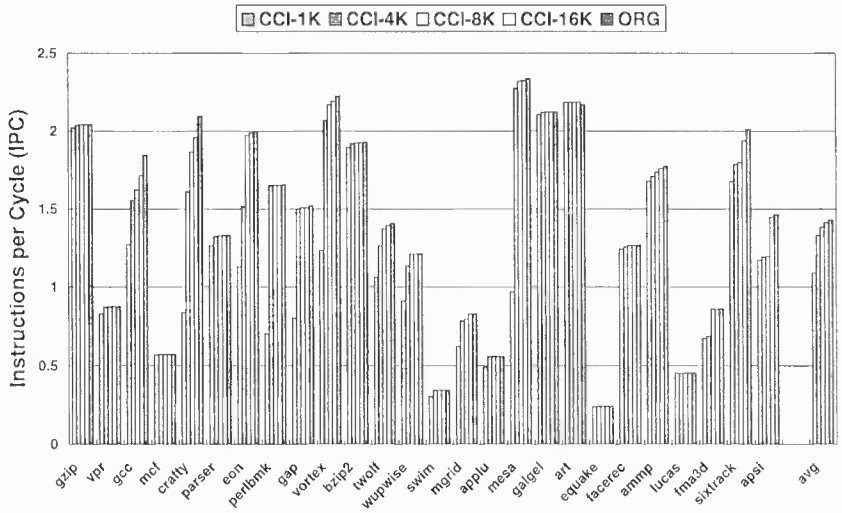


Figure 3.21 The IPC comparison at different invalidation intervals while applying the CCI scheme to the instruction cache. (`ORG` is the conventional instruction cache without CCI.)

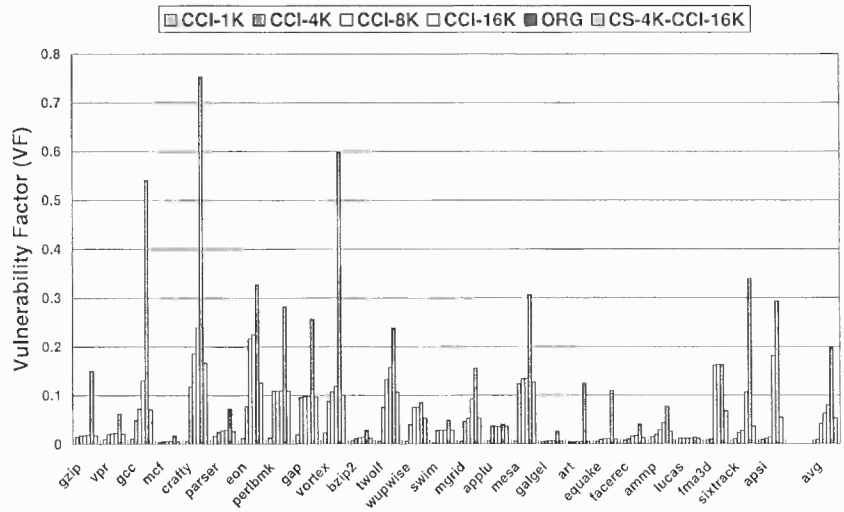


Figure 3.22 The TVF comparison at different invalidation intervals while applying the CCI scheme to the instruction cache. (`ORG` is the conventional instruction cache without CCI. `CS-4K-CCI-16K` is the combined scheme with 4K-cycle CS interval and 16K-cycle CCI interval.)

indicate a clear tradeoff between TVF and performance. If a 1K-cycle interval is used, though the TVF can be significantly reduced to 0.9% from the original 19.9%, the performance overhead is also tremendous, 20% performance loss, on the average. This extremely high performance loss is mainly because of the high pipeline stall penalty due to increased instruction cache misses incurred by the CCI scheme and is not affordable in high performance designs. On the other hand, if a 16K-cycle interval is chosen, the performance loss is well under 0.9%, while the TVF goes back to 8.0%. Even with a 4K-cycle interval, CCI achieves a TVF of 4.1% at a performance loss of 5.8%. Therefore, simply applying CCI to the instruction cache will not be as effective as for the data cache. Solutions in next section specifically address the performance issue in the CCI scheme for the instruction cache.

3.3.3 Cacheline Scrubbing (CS)

An accessed cacheline in the instruction cache is very likely to be accessed again due to the temporal locality property. The CCI scheme, on the other hand, invalidates the cacheline after it has been idle for a predefined time interval and incurs performance loss due to an extra cache miss if the line is to be reaccessed after the invalidation. To avoid this performance loss while still optimizing the TVF, this work proposes to consider cacheline scrubbing instead of invalidation, i.e., a cache miss is triggered to re-fetch the cacheline from the L2 cache. For this study, this work assumes that the L2 cache is protected by some means of ECC coding and therefore is error free. To minimize the performance overhead, the cache miss to re-fetch the cacheline can be scheduled during cache idle cycles. Notice that the scrubbing scheme is different from the schemes in [76][33][77] that scrub the data by recomputing the ECC codes to eliminate single bit errors based on a fixed scrubbing interval.

Figure 3.23 shows the TVF of the instruction data array employing the CS scheme with different scrubbing intervals. With a 4K-cycle scrubbing interval, the TVF is reduced to 5.5%. If the scrubbing interval increases to a larger one, such as 32K cycles, the TVF

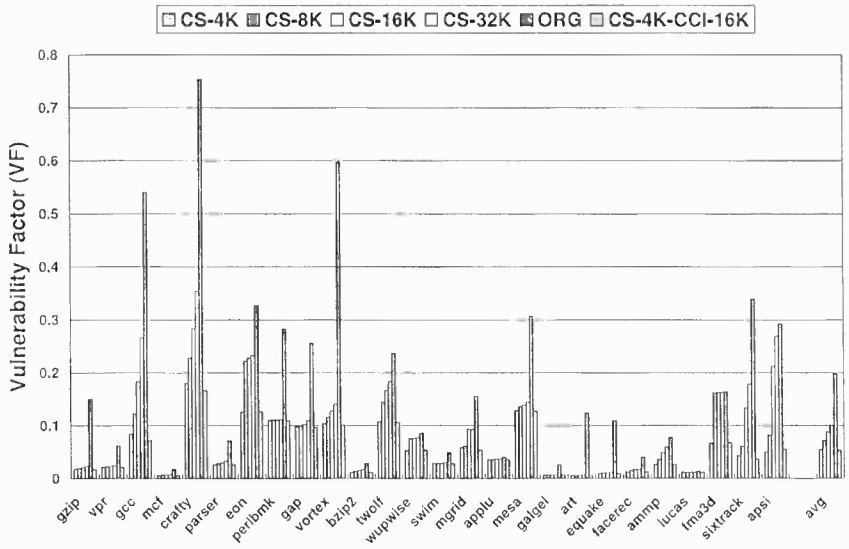


Figure 3.23 The TVF comparison at different scrubbing intervals at different scrubbing intervals. (ORG is the conventional instruction cache without scrubbing. CS-4K-CCI-16K is the combined scheme with 4K-cycle CS interval and 16K-cycle CCI interval.)

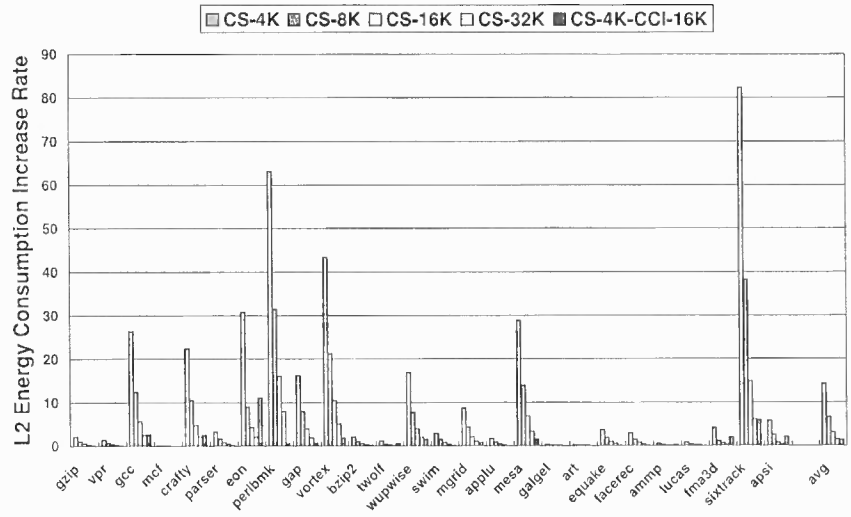


Figure 3.24 The comparison of the energy consumption increase rate (x times) in the L2 cache at different scrubbing intervals. (ORG is the conventional instruction cache without scrubbing. CS-4K-CCI-16K is the combined scheme with 4K-cycle CS interval and 16K-cycle CCI interval.)

also increases to 10.0%. Furthermore, if smaller intervals are chosen, there will be a huge increase in the number of accesses to the L2 cache. As shown in Figure 3.24, the energy consumption in the L2 cache is 14.3 times that of the original one if the instruction cache scrubs with a 4K-cycle interval. Even if the interval increases to 32K cycles, the energy consumption in the L2 cache still becomes 1.4 times that of the original one. Once again, there is a reliability-energy tradeoff. Without a solution to this energy issue, cacheline scrubbing may not be acceptable in energy efficient designs.

3.3.4 The Combined (CS-CCI) Scheme

Clean cacheline invalidation (CCI) benefits the most from capturing large RRs, while cacheline scrubbing (CS) optimizes relatively small RRs with negligible performance impact. To exploit the strength of both CCI and CS, this work proposes to explore combining CCI and CS for TVF optimization in the instruction cache. In the proposed combined scheme, an idle cacheline is first scrubbed after a small time interval. If the cacheline continues to be idle for a long interval, it is invalidated in order to prevent further (unnecessary) scrubbing. From the simulation results, a 4K-cycle interval for CS and a 16K-cycle interval for CCI are chosen. The results in Figure 3.22 show that the TVF of the CS-4K-CCI-16K combined scheme is 5.3% compared to the 8.0% of the CCI-16K scheme. Further, the performance of the CS-4K-CCI-16K scheme is almost the same as for the CCI-16K scheme, which is within 0.9% of the original scheme. Figure 3.24 shows that the L2 cache energy consumption of the CS-4K-CCI-16K scheme is about 1.29 times of that for the original scheme, as compared to the 14.3 times for the CS-4K only scheme.

3.4 TVF Characterization of Tag Arrays

3.4.1 Tag Array of the Data Cache

Lifetime of the Tag Array

The lifetime model of the tag array is quite different from that of the data array. This is because of the unique access pattern in the tag array. In the data array, if a clean cacheline is to be replaced, it is simply discarded, which makes the `RPL` time non-vulnerable. However, the `RPL` time of the tag array is still vulnerable. For example, during an access to a set-associative cache, all tags of different ways in the mapped set need to be read out and compared with the address tag field simultaneously. If one tag matches, the current access hits the cache. Otherwise, a cache miss is signaled. Thus, before a cacheline is selected as the candidate for replacement during a cache miss, its tag has been compared and the result is an unmatch. Now if there are errors in the tag, it is possible to cause a false match on this cache access. Furthermore, there is no update operation on the tag. Thus, the non-vulnerable phases `RW` and `WW` in the data array are not suitable for the tag array.

False Hit and False Miss

If errors occur in the tag array, it may cause erroneous cache hits or misses. However, false hit and false miss have different impacts on TVF characterization. A false hit happens when a tag struck by soft errors matches the tag field of the address, which was supposed to be a cache miss. On the other hand, a false miss happens when an error affected tag does not match the coming address tag, which should be a cache hit. A false hit will cause an incorrect execution by loading data from or updating a wrong cacheline. However, a false miss causes an additional cache miss and thus incurs performance loss. Its impact on TVF depends on whether it is in a clean line or a dirty line, since a false miss in a dirty cacheline will load stale data from the L2 cache. In a writeback cache, if the tag of a dirty cacheline is flipped by soft errors, the cacheline will be written back to a wrong location in the L2

cache, which is likely to cause an erroneous output.

Lifetime Model Based on the Extended Hamming-Distance-One (HDO) Analysis

If the single-bit error model is assumed, the false hit will happen only when the tag has one single bit different from the incoming address tag and this particular bit is flipped by the soft error. This work utilize the Hamming-distance-one analysis [33] to track false hits and further extend this HDO analysis method to characterize the TVF of the tag array. Notice that if a tag entry (its original value) matches an incoming address tag, any bit flipped by a soft error will cause a false miss. For tag entries with multiple bits different from the incoming tag, no false hit or false miss will happen. Furthermore, only the single different bit in the HDO tag entry is vulnerable for a clean cacheline. However, in a writeback cache, all bits in the tag entry of a dirty cacheline are vulnerable since either a false hit or a false miss will load erroneous data or corrupt the L2 cache.

Based on extended HDO analysis, this work propose to divide the lifetime of the tag array in a writeback cache into six phases: `RH`, `FWPL`, `RHFW`, `HFW`, `HPL`, and `Invalid`.

- `RH`: lifetime phase between the first read and the last Hamming-distance-one (HDO) match of a clean cacheline,
- `FWPL`: lifetime phase between the first write and the replacement of a dirty cacheline,
- `RHFW`: lifetime phase between the first read and the last HDO match before the first write of a dirty cacheline,
- `HFW`: lifetime phase between the last HDO match and the first write of a dirty cache-line,
- `HPL`: lifetime phase between the last HDO match and the replacement of a clean cacheline,
- `Invalid`: lifetime phase in the invalid state.

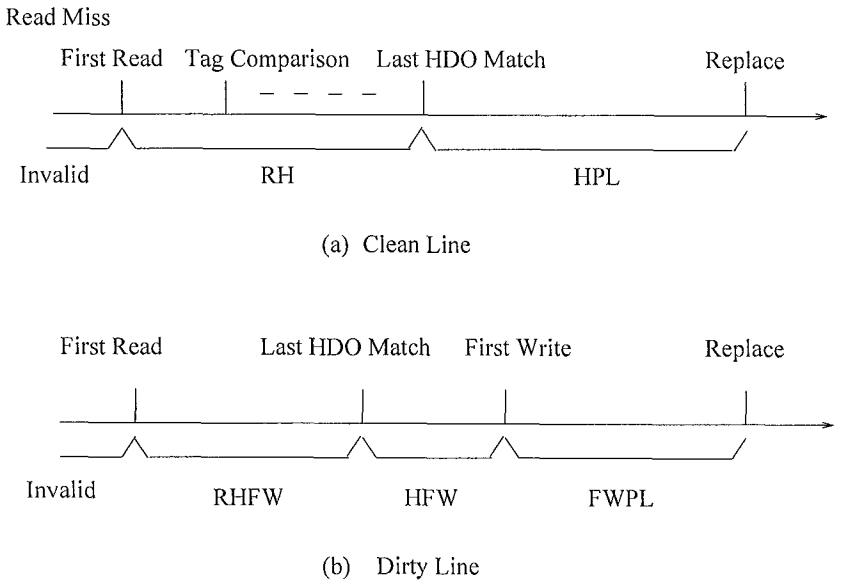


Figure 3.25 The tag lifetime of a cacheline in the writeback cache.

Figure 3.25 shows the correlation among the lifetime phases for typical tag activities. The RH, FWPL, and RHFW phases are vulnerable because errors occurring in the RH and RHFW phases will cause false hits, and errors occurring in the FWPL phase will cause incorrect writebacks to the L2 cache or erroneous data load. Phases HFW, HPL, and Invalid are non-vulnerable because errors occurring in the HFW phase will only cause a false miss on the first write in a clean cacheline, and errors occurring in the HPL phase will be discarded at replacement. Figure 3.26 shows the phase distribution of the tag entry in a writeback data cache. About 14.4% of the tag entry lifetime is in the RH phase. The FWPL phase contributes about 31.7%. Phases RHFW and HFW together account for 0.47%. Consequently, the TVF of the tag array is around 46.7%.

However, to improve the accuracy, TVF characterization based on the extended HDO analysis needs to be performed at the bit level. The bit-level analysis results in Figure 3.27 show that the RH vulnerable phase is reduced to 0.76% from 14.4% in the entry-level analysis (as shown in Figure 3.26). Notice that the FWPL vulnerable phase remains the same because all the bits in the FWPL phase are vulnerable. In the following study, the bit-level analysis is used for TVF characterization.

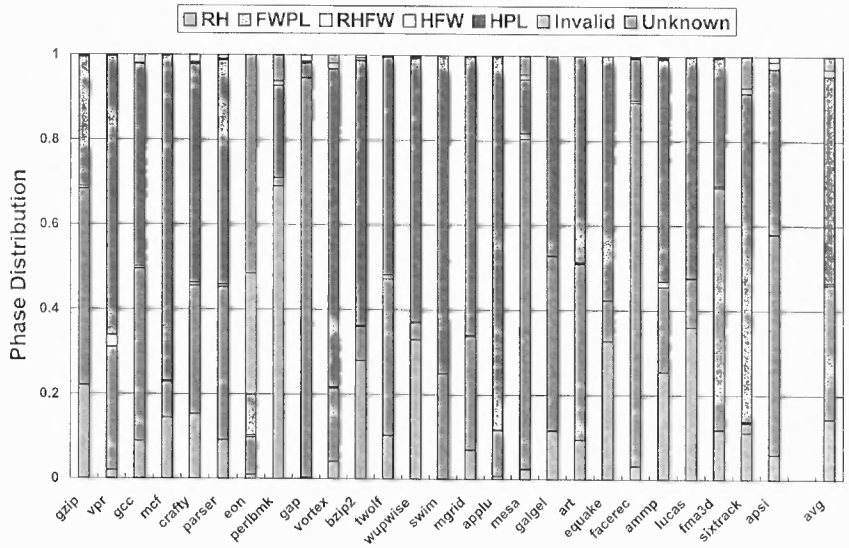


Figure 3.26 The lifetime distribution for the tag array in the writeback data cache at entry level.

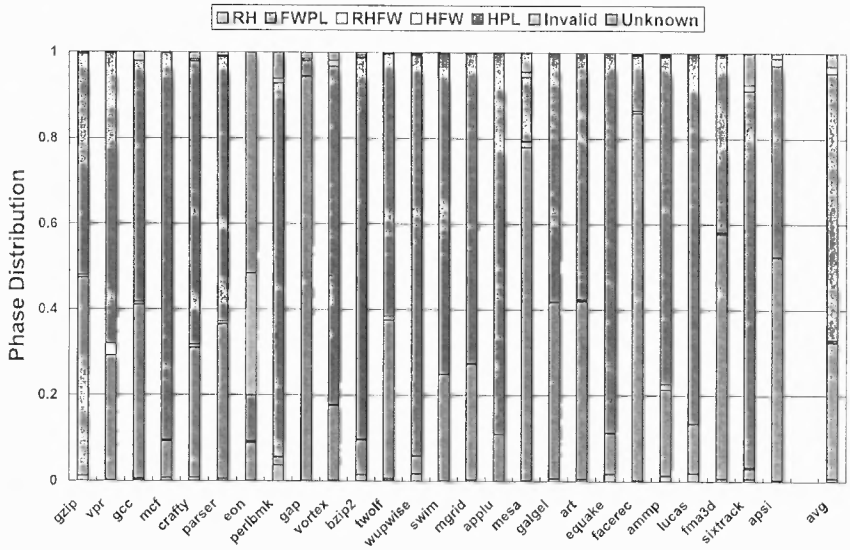


Figure 3.27 The lifetime distribution for the tag array in the writeback data cache at bit level.

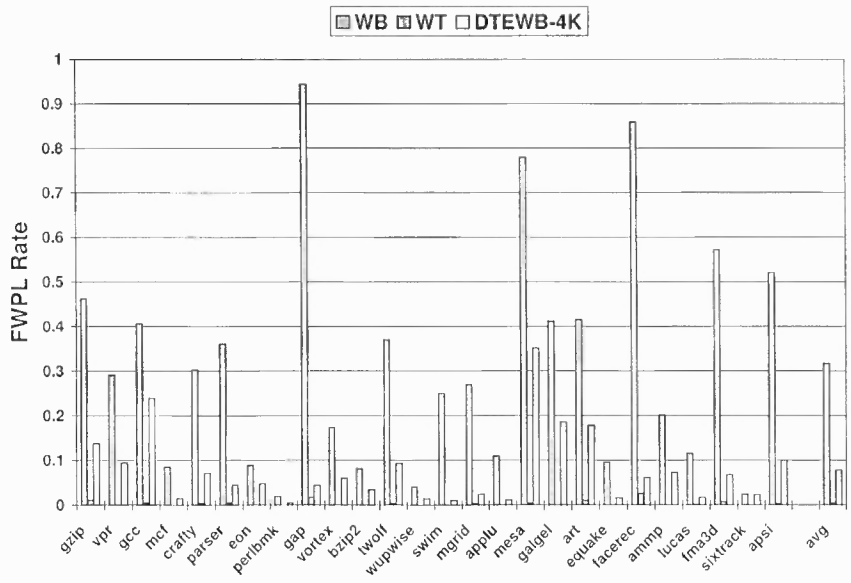


Figure 3.28 The FWPL rate comparison for the tag array in writeback (WB), writethrough (WT), and DTEWB caches.

In a writethrough cache, the FWPL phase is eliminated. The lifetime of read-only cachelines in writethrough caches is similar to that of the clean lines in writeback caches. However, the lifetime of cachelines with write operations in the writethrough cache is quite different from that of the dirty lines in the writeback cache. In order to illustrate this difference, this work compares the TVF of the cachelines with write operations in both the writethrough and writeback data caches. Figure 3.28 shows that the TVF of the cachelines with write operations in the writethrough cache is only 0.4%, as compared to 31.7% for the writeback cache.

The Impact of DTEWB and CCI on the TVF of the Data Cache Tag Array

The DTEWB and CCI schemes in the data array also help reduce the TVF of the tag array. The DTEWB scheme will reduce the FWPL phase of a tag entry in a writeback cache, while the CCI scheme will reduce the RH phase. In order to be consistent with the data array, the same 4K-cycle interval is used for both DTEWB and CCI in the tag array study. Figure 3.28 shows that the DTEWB scheme reduces the dirty line tag FWPL to 7.7% and Figure

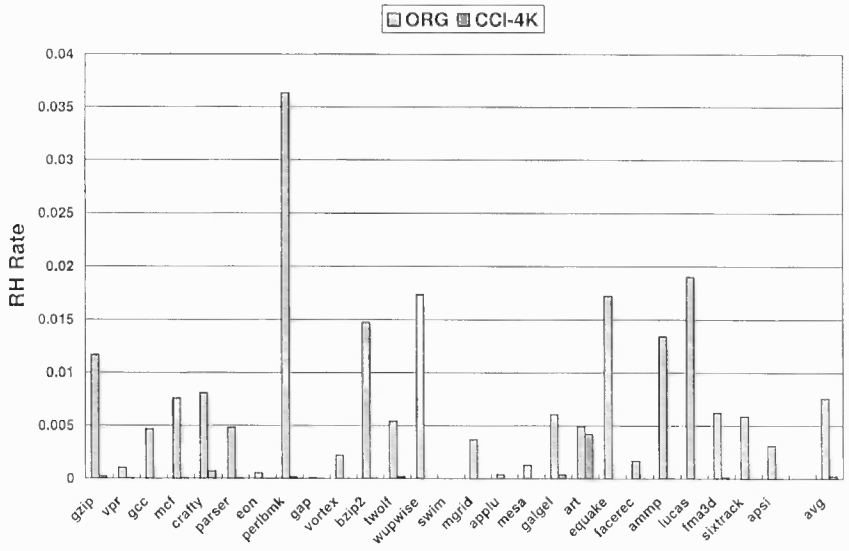


Figure 3.29 The RH rate comparison between the original and CCI schemes in the data cache.

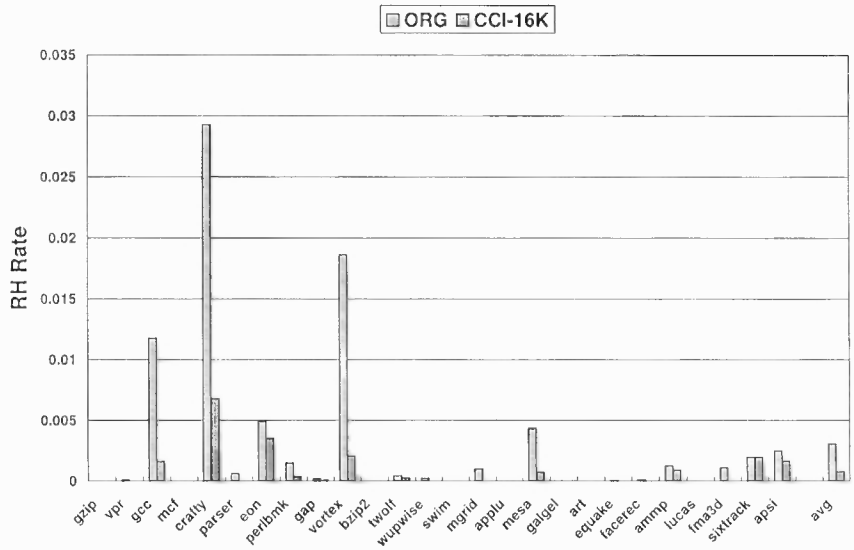


Figure 3.30 The RH rate comparison between the original and CCI schemes for the tag array in the instruction cache.

Table 3.3 Summary of targeting vulnerable phases of all proposed schemes.

Scheme	Data Cache		Instruction Cache	
	Data Array	Tag Array	Data Array	Tag Array
DTEWB	WPL	FWPL	-	-
MDB	WPL	-	-	-
CCI	RR	RH	RR	RH
NWVC	Overall	-	-	-
CS	-	-	RR	-
Combined	Overall	Overall	-	-
CS-CCI	-	-	RR	RR

3.29 shows that the CCI scheme reduces the RH rate of the clean line tags to 0.02%.

3.4.2 Tag Array of the Instruction Cache

The lifetime of the tag array in an instruction cache has only three phases: RH, HPL, and Invalid, among which only the RH phase contributes to TVF. Figure 3.30 shows that the TVF of the tag array in the instruction cache is only about 0.3%. Among the TVF reduction schemes for the data array of the instruction cache, the CCI scheme can also help reduce the TVF of the tag array. However, the CS scheme does not have any noticeable improvement on TVF. Therefore, this work only considers the CCI scheme and conduct a study on the CCI with the same 16K-cycle invalidation interval as in the combined scheme for the data array. The results in Figure 3.30 show that the CCI scheme reduces the tag array TVF to only 0.08%.

3.5 Summary

This work develops new lifetime models to analyze the cache vulnerability against soft errors. The major contributors (vulnerable phases) to the cache vulnerability are identified. Driven by the results from the temporal vulnerability factor (TVF) characterization, reliability schemes are proposed to target at specific vulnerable phases, which are summarized in Table 3.3 and 3.4. With the proposed schemes, the vulnerability of data and tag arrays

Table 3.4 Comparison of all proposed schemes

Scheme	Description
DTEWB	Reducing the WPL and FWPL vulnerable phases of the data and tag arrays in the writeback data cache, while minimizing the performance and energy overheads as compared to the writethrough cache.
MDB	Reducing the vulnerable phases (mainly WPL) of the data array in the data cache by preventing writing back clean data items to the L2 cache. Additional dirty tag bits are needed (1/64 storage overhead for the word-level tag).
CCI	Reducing the RR and RH vulnerable phases of the data and tag arrays in both the data and instruction caches. However, for the instruction cache, the reduction effect comes at the cost of high performance loss.
NWVC	Reducing all vulnerable phases by exploiting the narrow width values in the data array in the data cache. Additional narrow tag bits are needed (1/64 storage overhead for the word-level tag).
CS	Reducing the RR vulnerable phase of the data array in the instruction cache. However, the reduction effect comes at the cost of high energy consumption.
Combined	Reducing the overall vulnerable phases in the data cache with minimized performance and energy overheads by combining the DTEWB, MDB, CCI, and NWVC schemes (word-level tag bits for both MDB and NWVC, 4K-cycle intervals for both DTEWB and CCI).
CS-CCI	Reducing overall vulnerable phases in the instruction cache with minimized performance and energy overheads by combining the CS and CCI schemes (a 4K-cycle interval for CS and a 16K-cycle interval for CCI).

in both the data and instruction caches can be dramatically reduced while performance and energy overheads are minimized.

CHAPTER 4

TAG REPLICATION BUFFER FOR ENHANCING THE RELIABILITY OF THE CACHE TAG ARRAY

4.1 Introduction

While most of the previous work is targeting at improving the reliability of the data array in on-chip caches [59][51][55][58][60][41][78][56][52][53][28][40][29][54][79], few researchers have directed their attention to the reliability characterization and optimization of the cache tag array. In Chapter 3, a lifetime vulnerability model for the tag array has been proposed and studied. However, only those schemes, such as DTEWB and CCI, which targeted at improving the reliability of the data array, have been evaluated for their benefit on improving tag array reliability. Due to its crucial importance to the correctness of cache accesses, the tag array demands high reliability against soft errors while the data array is fully protected. The parity coding scheme is widely used to protect the on-chip L1 caches in today's reliable microprocessors [80][81] due to its low cost. However, simple parity coding is only capable of detecting an odd number of bit errors without error recovery capability. On the other hand, error correcting codes (ECCs) typically provide single error correction and double error detection (SEC-DED). Nevertheless, the performance overhead and additional energy consumption due to ECC encoding/decoding make ECC a reluctant choice for high speed on-chip L1 caches [28]. Instead, ECC codings are widely adopted in L2/L3 caches that can tolerate longer access latencies [80][81]. Exploiting the address locality of memory accesses, this work proposes a Tag Replication Buffer (TRB), a small buffer that captures and maintains the replicas of frequently accessed tag entries, to enhance the reliability of the tag array in the on-chip data cache. This work performs a detailed design space exploration for the TRB implementation and proposes several optimized schemes to improve the tag array reliability as well as to reduce the area and energy

overheads of the TRB. To further improve the protection effectiveness and the provided reliability of the TRB, this work then proposes a selective TRB scheme that only duplicates tag entries for dirty cachelines, which demand much higher reliability than clean cachelines. The experimental results show that the selective-TRB scheme can achieve a 97.4 % AWR (access-with-replica) rate with minimum overheads. In order to provide a comprehensive evaluation on the reliability of the cache tag array, this work conducts a cache tag vulnerability factor analysis and proposes a refined cache tag reliability evaluation metric DOR (detected without replica) TVF that combines the TVF and AWR analysis. Based on the DOR-TVF analysis, a new TRB scheme with early write-back (TRB-EWB) triggered by the TB (tag buffer) replacement is proposed, which can achieve a 100% AWR rate and a zero DOR-TVF for the tag entries of dirty cachelines with a minimum performance and energy overhead.

4.2 Tag Replication Buffer (TRB) for Improving Tag Array Reliability

4.2.1 Basics of the TRB Design

Microprocessor issued memory accesses exhibit various localities. The address locality is a form of locality due to the spatial and temporal locality of memory accesses. It means that if a memory address is referenced at a particular time, the same address and its nearby memory addresses are very likely to be referenced in the near future. In other words, only a small set of the memory addresses are referenced during certain execution time intervals. Since the tag entry of a cacheline is the higher portion of the referenced address, it has a better locality property than the full memory address. In this work, it is called cache tag locality (CTL). Exploiting the CTL, the work proposes to duplicate the tag entries in a small cache-like structure, called the tag buffer (TB), to enhance the reliability of the tag array in the data cache.

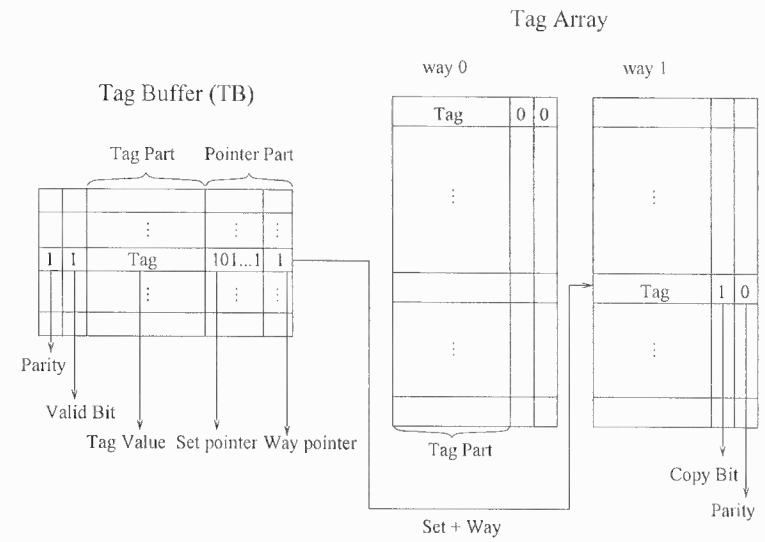
The tag replication buffer (TRB) design is an information redundancy based reliable scheme. However, simply keeping two or more identical copies of the tag array is not area

and energy efficient. By exploiting the CTL, a small TB (e.g., 32 entries) can capture most of the tag references. Thus, by keeping the most recently accessed (MRA) tag entries in the small TB, it can achieve a high *access-with-replica* (AWR) rate, providing a high reliability for the tag array. Notice that although the TRB design studied in this work is targeting at the data cache, it also applies well to the instruction cache since the instruction cache has a better locality than the data cache.

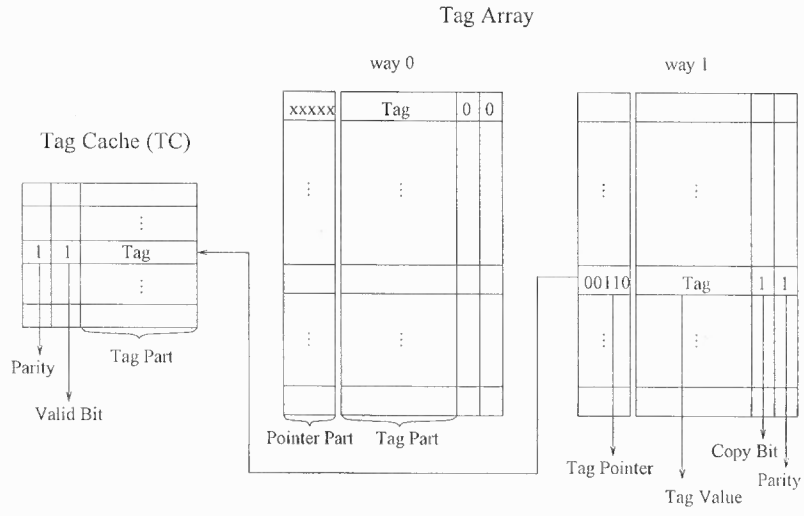
4.2.2 TRB Design

One of the key issues in the TRB design is how to identify the original tag with its replica. Figure 4.1 (a) shows the block diagram of the TRB design. Each entry in the TB has an additional space to store a pointer. The pointer contains two parts: the set pointer and the way pointer. The set pointer indicates the set of the original tag entry and the way pointer indicates the way of the original tag entry in a set-associative cache. Notice that the way pointer is not needed in a direct-mapped cache. An additional one bit (Copy Bit) is added for each original tag entry to indicate whether it has a replica in the TB or not.

In [34], a similar caching scheme CAT was proposed to optimize the area of the tag array. Since the CAT scheme aimed at area reduction, it replaced the original tag array with a CAM structure that only stores the pointers to the Tag Cache (TC) [34]. For reliability purposes, this work modified their scheme by just putting back the original tag array. The modified CAT design is shown in Figure 4.1 (b). The pointer part is at the original tag array side. Each tag entry in the original tag array is associated with a pointer pointing to the location of its replica in the TC. The TC entry only contains the tag replica. The Copy Bit in the original tag array is also needed to indicate whether the tag has a replica or not. The advantage of the CAT design is that multiple entries in the tag array can share the same replica in the TC, which has been discussed in [34]. However, area (37.3%) and energy (181%) overheads are extremely high compared to the TRB design, which makes it not a good choice for the duplication schemes.



(a) TRB



(b) CAT

Figure 4.1 The block diagrams of the TRB and modified CAT designs.

4.3 Exploring the Design Space of the TRB

4.3.1 How to Deal With Soft Errors

In the TRB design, all the tag bits including the original ones and the replicas in the TB are protected by parity coding. If the single-bit error model is assumed, all errors occurring in the tag can be detected but not recovered with parity coding. When a tag entry is accessed, the parity checking is performed. If it passes the check, there is no error in the tag. The normal routine of the cache access will continue. If the parity checking fails, the Copy Bit is examined. If the Copy Bit is one, it means that this tag has a replica in the TB. Then, all the pointers in the TB need to be checked simultaneously to find the replica. Therefore, the pointer part in the TB needs to be implemented as a CAM structure. If the replica passes the parity checking, the original tag can be recovered by copying back from the replica. If the Copy Bit is zero or the parity checking of the replica fails, the error in the original tag entry cannot be corrected by the TRB design. Since the probability of error occurring in both the original tag and its replica is extremely low, this work uses the AWR rate, which is the ratio of the tag accesses with a replica in the TB over the total number of tag accesses, as one of the evaluation metrics for the TRB design. The higher AWR rate indicates higher reliability of the tag array. Notice that the pointer entry in the TB is also protected by the parity coding.

4.3.2 When to Duplicate

A common issue in the information redundant strategy is when to make a redundant copy. In the TRB design, two mechanisms are explored to create replicas: a) duplicating with a new cacheline (DNC) - making a replica in the TB when a new tag entry is written into the tag array, i.e., at the time when a new cacheline is brought into the data cache from the L2 cache, b) duplicating with a TB miss (DTBM) - making a replica when the original cacheline is hit and there is no replica in the TB for its tag entry, i.e., the Copy Bit in its tag

entry is zero.

Based on these two mechanisms, two duplication schemes are proposed. The first one is the DNC-only scheme in which it only makes the replica with a new cacheline. The other one is the DNC+DTBM scheme in which it makes the replica in both conditions, i.e., when a new cacheline is written into the data cache or when a hit cacheline does not have a tag replica in the TB. Basically, if the DNC+DTBM scheme is chosen, it can keep more recently accessed tag entries in the TB to achieve a higher AWR rate. However, the DNC+DTBM scheme will incur more control overhead and energy consumption than the DNC-only scheme.

4.3.3 How to Do the Replacement

In the TB side, if a replica needs to be made for a tag entry while all the entries in the TB are occupied, a victim entry needs to be selected. Then, the original tag of the victim entry needs to be located by its pointer bits and the Copy Bit of that tag entry needs to be reset to zero. After that, the victim entry along with the pointer bits are replaced by the new value.

In the cache side, when a cacheline is to be replaced due to a cache miss and the Copy Bit in its tag entry is one, the replica in the TB needs to be located and the Valid Bit needs to be reset to zero.

4.3.4 Replacement Policies in the TB

In order to exploit the tag locality, the LRU (least recently used) policy is a good choice to select the victim entry in the TB. The LRU information needs to be updated during every tag access. Due to the highly-frequent updating operations and the implementation complexity, the LRU policy will incur high energy and area overheads. Therefore, this work also study the FIFO (first in first out) and Random policies in the TB, both of which are less expensive to implement. The FIFO policy can be implemented with a queue structure and

a head pointer indicating the head of the queue. In the random policy, a random number is generated to determine the victim entry in the TB.

4.4 Optimizing the TRB Design

4.4.1 Improving the Replacement Policy in the TB: LRU+ and FIFO+

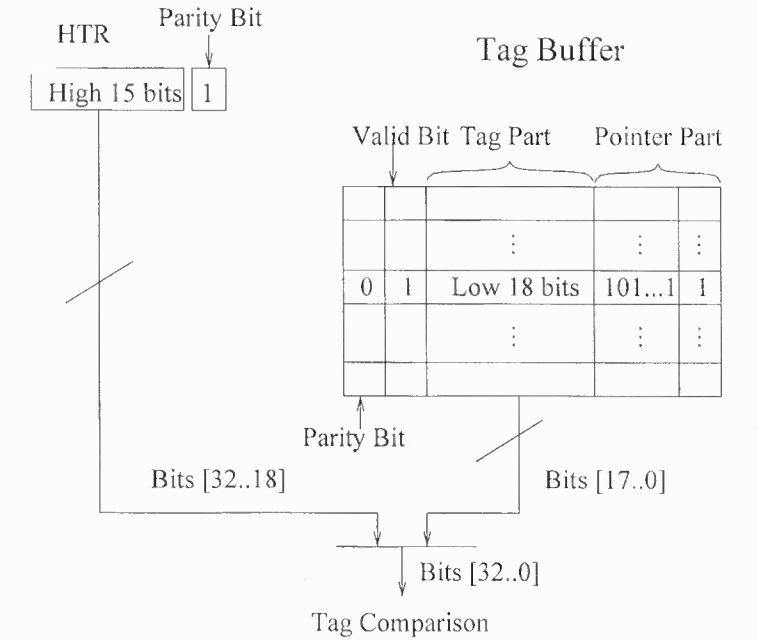
In the DNC scheme, when a new cacheline is brought into the cache, a victim entry needs to be selected in the TB for the replica. Normally, the victim entry is selected through the LRU or FIFO policy as discussed in Section 4.3.4. However, most of the new cachelines will cause the replacement of another valid cacheline in the cache. If that cacheline has a tag replica in the TB, the corresponding replica will be located and invalidated as discussed in Section 4.3.3. In that case, it should be beneficial to use that invalidated entry as the victim entry instead of applying LRU or FIFO policy, which may replace other valid replicas used by other tag entries. This new improved replacement policy is called LRU+ or FIFO+.

4.4.2 Tag Value Compression

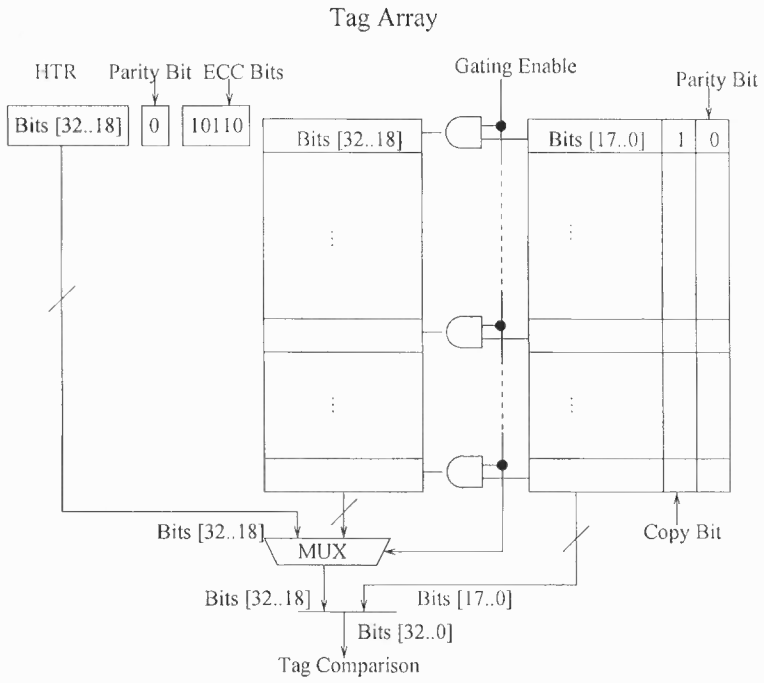
From the profiling results using the SPEC CPU2000 benchmark suite, It can be observed that only part of the entire set of tag bits is needed to resolve the tag conflict. These short tags have been identified as the *active tags* and studied for power optimization of tag arrays in [82]. In the simulated processor, the leading (high) 15 bits of the entire tag entry (33 bits) for most benchmarks almost never change during the execution. Therefore, this work proposes to adopt tag value compression to improve the area and energy efficiencies of the TRB design.

TB Side Compression (TBSC)

To reduce the area and energy overheads of the TB, the first tag value compression scheme, TB side compression (TBSC), shown in Figure 4.2 (a), is proposed. The high 15 bits of the



(a) TBSC



(b) TASC

Figure 4.2 The block diagrams of TBSC and TASC designs.

tag replica in the TB, which remain unchanged during the execution, are stored in a special register called high tag register (HTR) protected by the parity coding. The remaining 18 bits are stored in the TB similar to the original TRB design. When there is a TB write operation, it only writes the low 18 bits to the TB. If the error needs to be recovered from the replica, the values in the HTR and TB are read out simultaneously to form the entire tag values. Since the bit size of the TB is reduced in this scheme, the area and energy overhead in the TB will be reduced.

Notice that in this tag value compression scheme, it is assumed that the high 15 bits of the tag remain unchanged during the execution, which is based on the profiling results. Therefore, it only needs to write the HTR once at the very beginning of program execution. However, for other applications, the high 15 bits may change. In that case, the compiler support can be used to identify that particular code region similar to [82]. A special instruction can be inserted to disable the TRB scheme during the execution of that code region, which leaves the tag array unprotected by replicas. Fortunately, this situation rarely happens and the impact on the reliability of the TRB design is negligible.

Tag Array Side Compression (TASC)

To further reduce the energy consumption in the tag array, the second tag value compression scheme, tag array side compression (TASC), shown in Figure 4.2 (b), is proposed. Different from TBSC, the TASC scheme moves the HTR from the TB side to the tag array side. The high 15 bits of the original tag array are gated for energy savings. The HTR is protected by both parity and ECC codes. During normal access, the value in the HTR and the low 18 bits in the tag array are read out and parity checking is performed. If the parity checking fails in the HTR, the ECC code is used to recover from the error. The low 18 bits are protected by the original TRB scheme. Note that if the execution enters the special code regions discussed above, the high 15 bits will not be gated and the protection scheme will be disengaged.

4.4.3 Selective TRB

Recent work [40] and this work in Chapter 5 claimed that dirty cachelines should have higher priority to be protected than the clean cachelines in a write-back data cache. The clean cachelines in the L1 data cache have their copies in the L2 cache, which can be used to recover from soft errors if the L2 cache is protected by some highly reliable error coding schemes (e.g. ECC) and is error free assuming a single bit error model. Unlike the clean cachelines, the dirty cachelines do not have replicas in the L2 cache. Therefore, more reliable schemes are needed to protect the dirty cachelines. Based on that, this work proposed a selective TRB scheme that only duplicates the tags of the dirty cachelines. This selective-TRB scheme is expected to have a better AWR rate compared to the original one, since it reduces the number of tag entries that need to be duplicated. Basically, the less dirty cachelines the data cache has during the execution, the better AWR rate the selective-TRB can achieve.

4.4.4 Performance Impact

For a normal cache access, tags will be read out and compared with the address tag field. Simultaneously, the parity codes are checked for errors. If there is no error in the tag, the normal routine of the cache access will continue. Meanwhile, in the DTBM scheme, the Copy Bit will be also checked. If it is 0, the duplication routine will be triggered. Since most of the accessed tags should have replicas and the duplication is not on the critical path of the pipeline, the performance impact due to the duplication is trivial. If the parity checking fails, the error recovery routine discussed above will be triggered, which may hurt the performance. However, due to the extremely low error rate in the real world, the recovery routine is rarely triggered. In the error injection simulation even with an error rate of 10^{-7} per selected bit per cycle [40], the performance degradation due to the error recovery is negligible.

4.5 TVF Analysis of Tag Arrays

4.5.1 Lifetime of Tag Arrays

In Chapter 3.4 , based on the HDO analysis, the lifetime of a tag entry in a write-back cache is divided into six phases: RH, FWPL, RHFW, HFW, HPL, and Invalid. The RH, FWPL, and RHFW phases are vulnerable because errors occurring in the RH and RHFW phases will cause false hits, and errors occurring in the FWPL phase will cause incorrect writebacks to the L2 cache or erroneous data load. All tag bits in the FWPL phase are vulnerable. Phases HFW, HPL, and Invalid are non-vulnerable because errors occurring in the HFW phase will only cause a false miss on the first write in a clean cacheline, and errors occurring in the HPL phase will be discarded on replacement.

4.5.2 Detected withOut Replica (DOR) TVF

The above TVF analysis assumes no error protection schemes, such as parity or ECC codings. Note that the TRB design by default is protected by the parity coding. If the single-bit error model is assumed, in the parity-protected tag array, the SDC (silent data corruption) TVF is converted into DUE (detected unrecoverable error) TVF. However, for a clean cacheline with a detected error in its tag, if it is assumed that the L2 cache is protected by some means of ECC coding and is error free under the single bit error model, this single bit error can be recovered by invalidating the cacheline and reloading it from the L2 cache. Thus, these single-bit errors in tag entries of clean cachelines are DREs (detected recoverable errors). If an error hits the tag of a dirty cacheline, the updates to the cacheline will be lost, which contributes to DUE. From the simulation results, the FWPL phase contributes the most to DUE-TVF in the data cache tag array. Therefore, protecting the tag entry of a dirty cacheline should have a much higher priority than that of a clean cacheline.

To evaluate the reliability provided by the TRB design, this work introduces a refined TVF metric that combines the vulnerability factor and AWR analysis. It converts the

tag DUE-TVF of a dirty cacheline into two categories: detected without replica (DOR) TVF and detected with replica (DWR) TVF

Since the situation that there are errors in both the tag entry and its replica rarely happens, lower DOR-TVF means higher reliability in the tag array. If the strong assumption is made that an error affected tag can always be recovered from its replica, the DWR-TVF will be converted into the detected recoverable error TVF (DRE-TVF).

4.5.3 AWR v.s. DOR-TVF

From the previous discussion, the selective-TRB should be a good choice to reduce DOR-TVF of the tag array in the data cache by providing tag replicas at a high AWR rate for error recovery. Although the selective-TRB scheme achieves a higher AWR rate, the reduction effect on the DOR-TVF may not be satisfiable. This is because the selective-TRB scheme does not have a high RWR (replace-with-replica) rate, which is the ratio of the tag replacements with a replica in the TB over the total number of tag replacements. All the duplication and replacement policies studied so far in the TRB design are aiming at protecting the most recently accessed tags by exploiting the address locality. However, the major contributors to the tag TVF are the phases with no access activity. To summarize, in order to achieve a high AWR, the frequently accessed tags need to be protected. However, to significantly reduce the DOR-TVF, the tags with a long dead time need to be protected. These two criteria, AWR and DOR-TVF, in evaluating the reliability of tag arrays seem contradictory. To address this issue, this work explores a new scheme that can achieve optimized results under both criteria.

4.5.4 Early Write-Back Triggered by TB Entry Replacement

A direct solution to reduce the DUE-TVF is to use write-through data caches. A non-protected write-through data cache has a very low TVF in the tag array. In a parity-

protected write-through data cache, the tag array does not have the DUE-TVF. However, due to the performance degradation and the significantly increased accesses to the L2 cache [58][41][78], the write-through data cache is not preferred for applications that require high performance and low energy consumption.

In order to achieve both a high AWR and a low vulnerability factor, this work proposes a new TRB scheme with early write-back (TRB-EWB) triggered by TB entry replacement. In the TRB-EWB scheme, it only duplicates the tags of dirty cachelines, which is similar to the selective-TRB scheme. When a replica entry in the TB is replaced in the TRB-EWB scheme, its corresponding dirty cacheline will be forced to write back to the L2 cache. Therefore, all the tags of dirty cachelines have their replicas in the TB and those dirty cachelines that are to lose their replicas in the TB will become clean due to the early write-back. Since the replacement in the TB does not occur frequently with a high AWR rate, the TRB-EWB scheme incurs much less L2 cache accesses than the write-through scheme. Notice that compared to the dead-time based early write-back schemes in [58][41][78], the TRB-EWB scheme achieves a 100% AWR rate for dirty cachelines and reduces the DOR-TVF to zero.

4.6 Evaluation

4.6.1 TB Duplication Policies: DNC-Only v.s. DNC+DTBM

In the TRB design, there are two duplication policies. One is DNC-Only policy, which performs the duplication only when a new cacheline is written into the data cache. The other is DNC+DTBM policy, which makes the duplication not only when a new cacheline is written into the data cache but also when a hit cacheline does not have a tag replica in the TB. To evaluate these two duplication policies, this work uses the LRU replacement policy in a 32-entry TB. Figure 4.3 shows that the average AWR rate of the TRB with the DNC+DTBM policy is 91.5% compared to 42.1% with the DNC-Only policy. Therefore, to achieve a high AWR rate, the DNC+DTBM policy is preferred. DNC+DTBM is used as

the default policy in the following study.

4.6.2 TB Sizes

Figure 4.4 shows the AWR rates for different TB sizes. An 8-entry TB only has a 69.9% AWR rate and the AWR rate of a 16-entry TB is increased to 82.7%. In comparison, a 32-entry TB achieves a very high AWR rate, 91.5% on the average. If further increasing the size of the TB, the area overhead of the TB and the energy consumption in TB accesses will dramatically increase. Therefore, to balance the achieved AWR rate and the incurred overheads, the 32-entry TB is chosen in the TRB design.

4.6.3 TB Replacement Policies

In the previous sections, in order to study different design schemes and duplication policies in the TRB, he LRU is assumed as the default TB replacement policy. Although the LRU policy is good at exploiting the property of locality, a more cost-effective policy is needed due to the LRU's implementation complexity. Therefore, this work compares three replacement policies, LRU, FIFO, and Random. The results in Figure 4.5 show that the LRU policy achieves the highest AWR rate, 91.5%, while the AWR rate of the FIFO policy is 90.0%. The Random policy has the lowest AWR rate, 88.1%. The LRU information needs to be updated upon every tag access. Therefore, in a cost-effective design, the FIFO policy is preferred. The FIFO is used as the default TB replacement policy in the following study.

4.6.4 Comparison to Related Work

To provide a comprehensive analysis of the TRB design, this work compares it with the ECC, CAT, and full duplication (FD) schemes. The FD scheme maintains an identical copy of the tag array. All the tag values in TRB, CAT, and FD are protected by parity

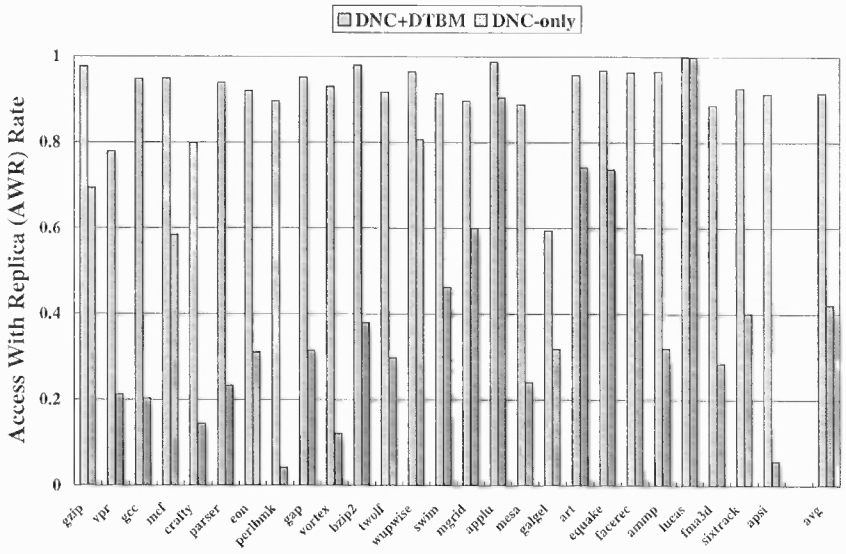


Figure 4.3 The AWR rate comparison of the TRB with different duplication policies.

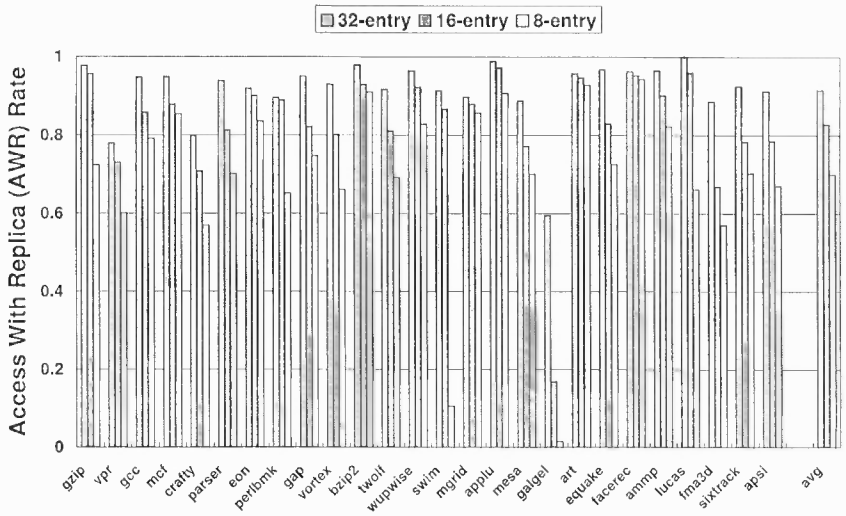


Figure 4.4 The AWR rate comparison of the TRB with different TB sizes.

coding. A 32-entry TB (TC) with FIFO and DNC+DTBM policy is used in TRB (CAT). Table 4.1 shows the comparison of these four schemes in terms of performance, area, and energy overheads as well as the AWR rate. It is assumed that the parity checking can be overlapped with the tag comparison and will not cause additional delay to the cache access. For ECC coding, a (33, 40) coding scheme is used for each tag entry (33 bits) and optimistically assume one additional cycle delay in cache access. The energy number for parity and ECC coding is scaled from [29]. The area overhead is estimated with a modified Cacti 4.2 [83]. For the CAM estimation of the pointer part in the CAT and TRB designs, this work utilizes the implementation of the tag matching in a fully-associative cache from Cacti.

Table 4.1 shows that because of the performance degradation (2%) and the high energy overhead (121%), ECC is not a good choice for protecting the on-chip L1 cache in high-performance processors. The 106% area overhead in the full duplication (FD) scheme also makes it an inefficient design. Although compared to the TRB scheme, the TC entry sharing property in the CAT scheme [34] results in a higher AWR rate (92.5%), the high energy (181%) and area (37.3%) overheads are still not acceptable. In conclusion, the TRB design achieves the lowest area and energy overheads among these four schemes with an AWR rate of 90.0%. Note that the 100% AWR rate for the ECC scheme means that all the single bit errors occurring in the tag entry can be recovered by ECC.

Table 4.1 The Comparison of the ECC, CAT, FD, and TRB schemes.

	ECC	FD	CAT	TRB
Performance Degradation	2%	0%	0%	0%
Area Overhead	21%	106%	37.3%	15.6%
Energy Overhead	121%	44%	181%	19.6%
AWR Rate	100%	100%	92.5%	90.0%

4.6.5 TRB Optimization Schemes

LRU+ and FIFO+

To improve the TB replacement policy, the LRU+ and FIFO+ are proposed in Section 4.4.1. Experimental results show that the LRU+ and FIFO+ can improve the AWR by 0.6% and 1.0% on the average (shown in Figure 4.5) without noticeable overheads. Therefore, FIFO+ policy will be used in the following study.

TBSC and TASC

The TB side compression (TBSC) scheme targets at reducing the area and energy overheads in the TB. Experimental results show that TBSC reduces the area and energy overheads to 12.9% and 16.7%, compared to 15.6% and 19.6% in the original TRB scheme.

Tag array side compression (TASC) scheme targets at reducing the energy overhead in the tag array. From the experimental results, the energy consumption in the tag array access is reduced by 17% due to the gating scheme. If the overall energy consumption is considered in the TASC scheme, it will remain almost the same compared to a conventional cache, which offsets the energy overhead incurred by the TRB.

Selective-TRB

In order to study the effectiveness of the selective-TRB, this work first conducts a profiling on the cacheline distribution in a write-back data cache. Results show that on the average only 33% of the cachelines in the data cache are dirty during program execution. The clean cachelines account for 66% and the rest are invalid (not in use). By duplicating and maintaining the tags of only dirty cachelines in the TB, the selective-TRB should deliver a much higher AWR rate due to the virtually tripled TB size. Figure 4.6 shows that the AWR rate of the selective scheme is increased to 97.4% compared to the 91.0% in the original scheme.

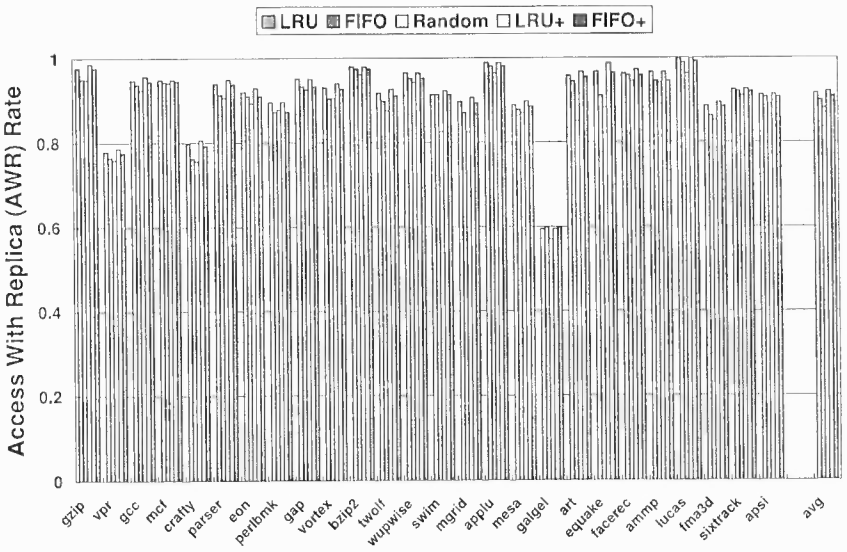


Figure 4.5 The AWR rate comparison of the TRB with different TB replacement policies.

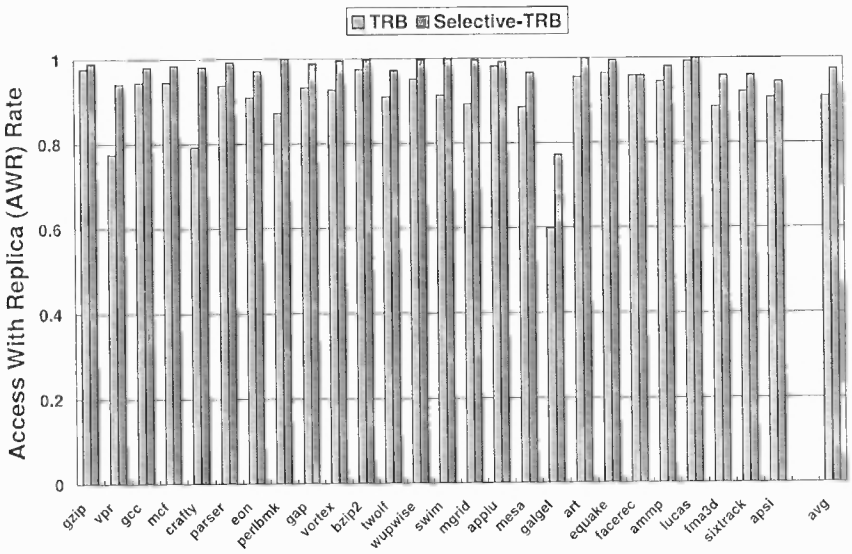


Figure 4.6 The AWR rate comparison between the original TRB and selective-TRB schemes.

4.6.6 Tag Array TVF Analysis

Figure 3.27 in Chapter 3.4 shows the phase distribution of the tag array in a write-back data cache with bit-level analysis. About 0.76% of the tag lifetime is in the RH and RHFW phases. The FWPL phase contributes about 31.7%. Therefore, the total TVF of the tag array is about 32.5%.

4.6.7 TRB with Early Write-Back (EWB)

To evaluate the reliability of the TRB design, a new metric, DOR-TVF, is introduced. Figure 4.7 shows that the original TRB and selective-TRB schemes have reduced the tag DOR-TVF of dirty cachelines from 31.7% to 22.6% and 16.7%, respectively. This moderate improvement of the DOR-TVF is mainly due to the fact that TRB is not directly optimizing the long FWPL phase. As shown in Figure 4.8, the replace-with-replica (RWR) rates stay low, 18.3% for the original TRB and 51.4% for the selective-TRB. However, if a write-through cache is used, the performance will degrade 3.7% and the energy consumption of the L2 cache will be more than doubled compared to that in a write-back cache, as shown in Figure 4.9 and 4.10.

By early writing back dirty cachelines triggered by TB replacement, TRB-EWB achieves a 100% AWR rate and a 100% RWR rate, consequently delivering a zero DOR-TVF for the tags of dirty cachelines. As shown in Figure 4.9, TRB-EWB with a 32-entry TB incurs a negligible performance loss ($< 0.01\%$). Notice that reducing the TB size will cause more early write-back operations to the L2 cache. To further study the energy consumption in the TRB-EWB design, this work conducts the simulation with different TB sizes. Figure 4.10 shows that the TRB-EWB with a 32-entry TB only incurs a 9.7% energy increase in the L2 cache. If further decreasing the TB size to 16-entry or 8-entry, the energy consumption will increase by 18.8% or 27.9% compared to that in the write-back cache.

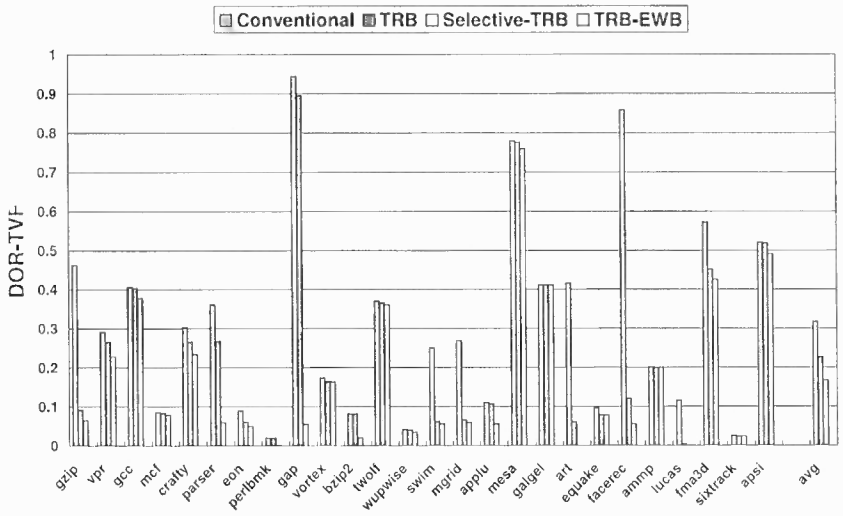


Figure 4.7 The DOR-TVF comparison for the parity-protected tags of the cachelines with write operations among conventional, original TRB, selective-TRB, and TRB-EWB data caches.

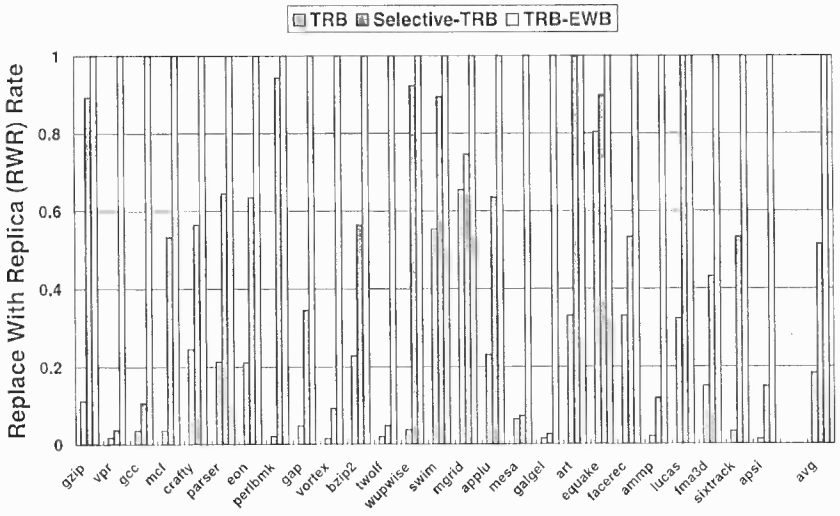


Figure 4.8 The replace-with-replica (RWR) rate of the original TRB, selective-TRB, and TRB-EWB schemes.

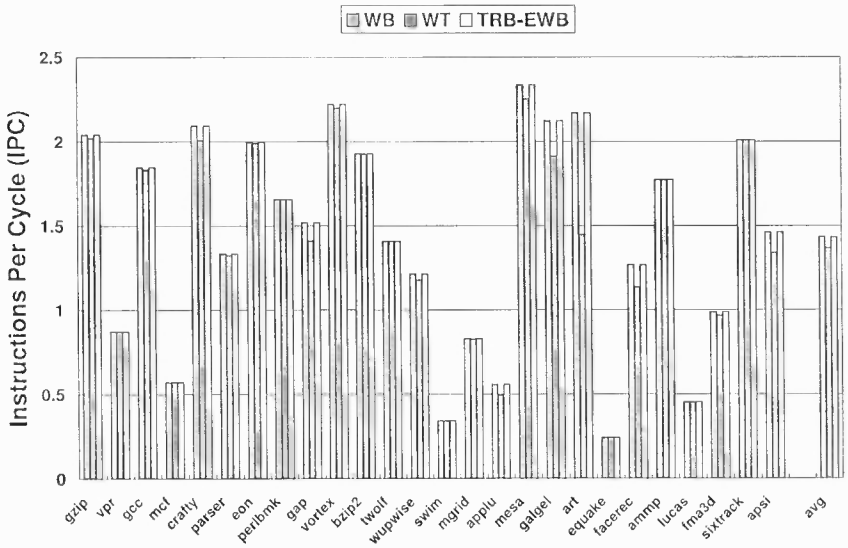


Figure 4.9 The comparison of performance among conventional write-back (WB), write-through (WT), and TRB-EWB data caches.

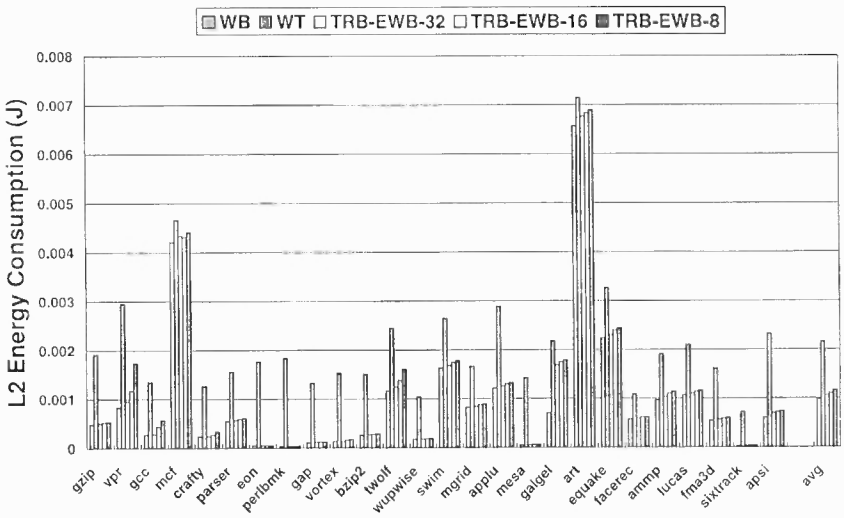


Figure 4.10 The comparison of L2 cache energy consumption among conventional write-back (WB), write-through (WT), and TRB-EWB data caches.

4.7 Summary

In this work, a tag replication buffer (TRB) design is proposed by exploiting the memory address locality to protect the tag array of the on-chip data cache against soft errors. Several optimized schemes such as TBSC, TASC, and selective-TRB are proposed to further improve the reliability and reduce the energy and area overheads in the TRB designs. The simulation results show that the selective-TRB scheme with the DNC+DTBM duplication and FIFO+ replacement policies achieves a high reliability in terms of the AWR rate (97.4%) for the tags of dirty cachelines, at a moderate hardware overhead. To further characterize and optimize the reliability of the cache tag array, this work conducted the TVF analysis and proposed a refined evaluation metric DOR-TVF that combines the vulnerability factor and AWR analysis. Based on the DOR-TVF analysis, this work proposed the TRB-EWB scheme to further improve the tag array reliability by optimizing the contradicting TVF and AWR simultaneously. The experimental evaluation shows that the TRB-EWB scheme achieves a 100% AWR rate and a zero DOR-TVF for the tags of dirty cachelines at a negligible performance loss and a minor energy overhead. These results also confirm that the TRB schemes can be an effective solution to protecting the tag arrays of on-chip caches for high-performance reliable microprocessors.

CHAPTER 5

SELF-ADAPTIVE DATA CACHES

5.1 Introduction

Most of the previous work has studied tradeoffs between performance, energy, area overheads and achieved cache reliability for their proposed schemes [28][30][55], their proposals are mainly targeting at cost-effective reliable cache designs with the assumption of fixed operating environments, e.g., supply voltage, operating temperature, system location, etc. However, many of the environmental conditions may change dynamically during the operation of the system under consideration, which makes the system more or less vulnerable to soft errors. For example, dynamic-voltage scaling (DVS) [84] based on-chip power/energy optimization schemes may increase the chip vulnerability when the supply voltage is scaled down [3]. Systems deployed within mobile objects, such as transportation vehicles, ships, or air planes, may experience very different intensities of (cosmic ray induced) neutron flux and the soft error rates may change with the latitude or altitude during travel. For example, the cosmic ray flux can be 1000x more intense at the altitude of the commercial flight than at the sea level [1][3]. Under such a situation, a conventional reliable design targeting a typical error rate will be either insufficient to provide the required reliability at harsh environments or over-designed for low error-rate environments. Furthermore, microprocessor-based system design is a widespread design methodology that helps to expedite the design process, reduce the design cost, and improve the reliability of the designed system. While these systems may be designed for different application purposes, implying different reliability requirements and operating environments, off-the-shelf commercial microprocessors must be designed with sufficient flexibility and adaptability, in terms of reliability schemes, to support various applications. To address this new requirement, a self-adaptive reliable design will be of great value to future microprocessors.

This work proposes a self-adaptive reliable data cache that dynamically adjusts its reliability scheme to the changing operating environments such that the reliability design target is guaranteed during its operation while performance and energy overheads are minimized simultaneously. Specifically, it provides three levels of soft error protection with increasing strength and recovery efficiency. To track variations in the operating environments that reflect on SERs, a mechanism is developed to dynamically monitor the detected error rate within a given sampling window. After each sampling window, the adaptive controller retrieves this information from a special counter, based on which the controller 1) predicts the undetected error rate or silent data corruption (SDC) rate and the performance/energy overhead for correcting detected errors, 2) decides whether to upgrade, keep, or downgrade the current protection level such that the SDC rate is kept below the preset threshold while the performance and energy overheads can be further optimized, and 3) reconfigures the data cache protection strategies according to the decision made in 2). The experimental evaluation shows that the self-adaptive data cache scheme achieves similar reliability to the most robust scheme while maintaining the performance and energy overheads of a lightweight scheme.

5.2 Error Model and Soft Error Injection

To evaluate the error resilience of the proposed schemes, this work conducted soft error injection during the execution-driven simulation. The soft error injection flips one bit or multiple bits in a selected cache line. Since the multiple-bit error rate is several orders of magnitude lower than the single-bit error rate [40], a single-bit error model is assumed in this study. Therefore, the error injection scheme simulates single-event upsets (SEUs) in the data cache. At each clock cycle, a uniformly distributed random function is called to locate a cacheline and a specific bit within that line. Then, an error is injected with a given probability (e.g., 10^{-6} shown as e-6), i.e., single-bit soft error rate per selected bit. As a general way to perform architectural-level error injection [40], accelerated error rates is

used in order to expose the error behavior and evaluate the reliability of the system.

To avoid crashing the simulation, each injected error is logged using a bitmap for each cache line instead of flipping the real bit value and the error history is also recorded. During the simulation, the soft error bitmap and error history information of a given cache line are used to perform error detection and recovery. Notice that each store operation clears out the errors previously injected into that particular data item in the data cache.

5.3 Reliable Data Caches Built upon Byte-Level Parity Coding

The study has shown that at the same area overhead the *byte-level* parity coding provides the same error detection capability as typical (72,64) ECC codes in the data cache under extremely high soft error rates. In the meantime, byte-level parity coding can be seamlessly integrated into the data cache without modifying the regular cache access procedure. Therefore, in the reliable data caches, the byte-level parity coding is chosen as the error detection scheme. For parity checking, a 10% energy overhead (of a cache access) is assumed for encoding/decoding a word data [30]. This study also assumes that the L2 cache is protected by some means of ECC coding and is error free.

Note that systems targeting different applications and different operating environments may have very different reliability requirements. For a given soft error rate in a target environment, this work considers three levels of soft error protection schemes for the data cache: reliable data cache with byte-level parity coding (RDC-P), RDC-P with in-cache replication (RDC-P-ICR), and RDC-P-ICR with early writeback and clean cacheline invalidation (RDC-P-ICR-EWB-CCI). These three schemes are discussed and evaluated in the following subsections.

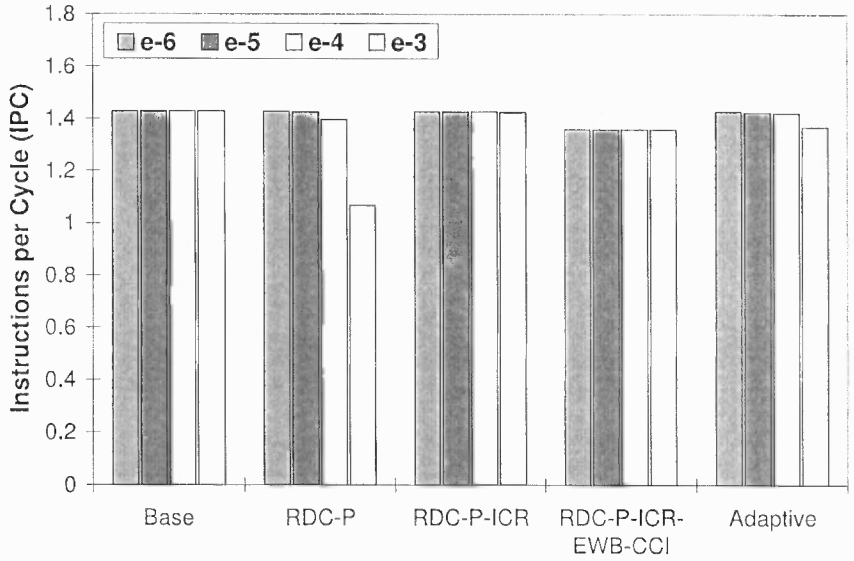


Figure 5.1 The performance comparison between the SA-RDC and fixed RDC schemes under error injection.

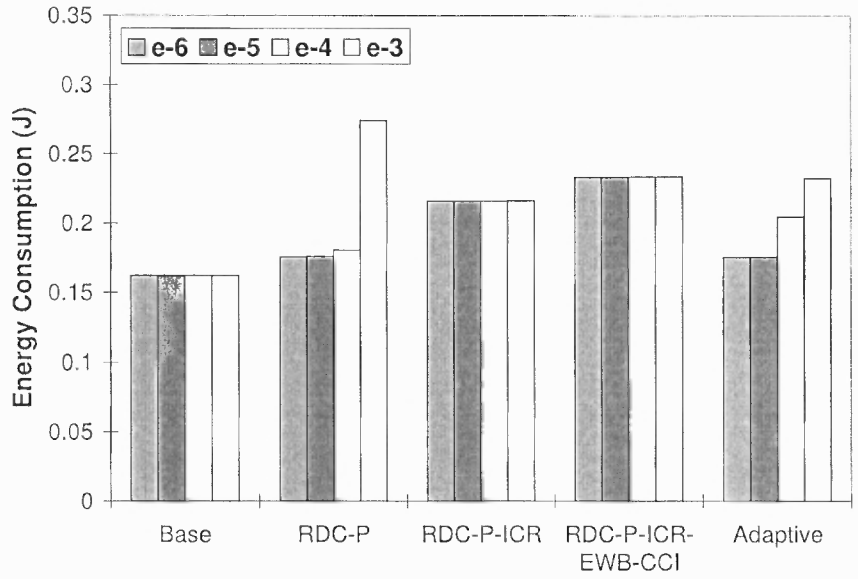


Figure 5.2 The Energy consumption comparison between the SA-RDC and fixed RDC schemes under error injection.

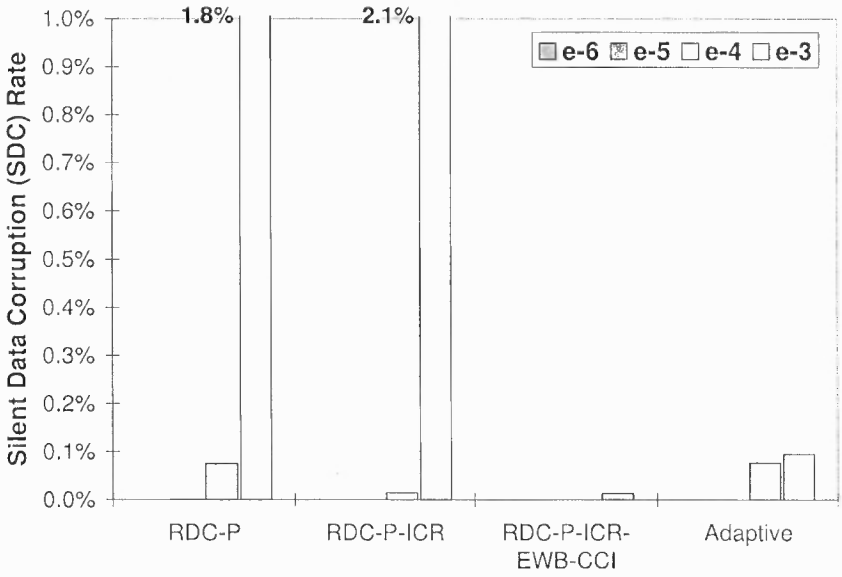


Figure 5.3 The SDC rate comparison between the SA-RDC and fixed RDC schemes under error injection.

5.3.1 Limits of Conventional Reliable Data Caches

RDC-P Data Cache

Basically, the error detection ability of parity coding applied to byte-level data is sufficient to handle relatively low error rates, such as e-6 or e-5. Therefore, the first level of error protection scheme only uses the parity coding to protect each byte in the data cache. When some data in the data cache is read by CPU or written back to the L2 cache, the parity checking is performed for each byte of that data item. Any single-/odd-bit error in a byte will be detected and signaled for error handling. If the erroneous data is in a clean cacheline, it can be recovered by invalidating the current cache line and re-fetching the data copy from the L2 cache. If the error occurs in a dirty line, hardware exception will be generated and the operating system will take over for error recovery based on some checkpointing schemes [85]. The optimistical 1000 cycles for the operating system to handle the error situation is assumed. Note that in real cases this O.S. recovery latency may be much longer.

The RDC-P scheme works best at relatively low error rate conditions. However, as the error rate increases, RDC-P scheme will suffer from two major problems. First, a

high error rate may introduce a large number of double-bit or multi-bit errors that cannot be detected by the parity scheme. Figure 5.3 shows that when the error rate reaches e-3, the undetected error rate in the RDC-P scheme is dramatically increased to 1.8%. This silent data corruption (SDC) presents the potential of crashing the program execution or resulting in erroneous output [86]. Second, the O.S. recovery overhead in the RDC-P scheme becomes significant as the error rate increases. Figure 5.1 shows that the performance loss of RDC-P scheme at error rate e-6 is negligible. This number increases to 0.23% and 2.3% at error rates of e-5 and e-4, respectively. A dramatic performance loss of 25.2% is observed, when the error rate is extremely high (e-3). Similarly, the total energy consumption of the data cache and L2 cache increases by more than 70% as the error rate reaches e-3, as shown in Figure 5.2. This huge performance and energy overhead at high error rates is mainly due to the frequent O.S. invocations in the RDC-P scheme.

RDC-P-ICR Data Cache

To cost-effectively recover from error-corrupted dirty lines at relatively high error rates, the in-cache replication (ICR) scheme [30], which duplicates the dirty cache line within the cache, is borrowed as the alternative solution to alleviate the performance overhead. When erroneous data is detected in a dirty line of the RDC-P-ICR, the recovery scheme will first search its replica and check the parity of the replica. If the replica passes the parity check, it is assumed to be error free and is used to recover the corrupted primary data copy. Therefore, a successful recovery from the in-cache replica can significantly reduce the recovery overhead. However, if the replica is also error-corrupted and detected by the parity checking, RDC-P-ICR use the same strategy, O.S. recovery, as in RDC-P. Figure 5.1 shows that at high error rates such as e-4 and e-3, compared to the RDC-P scheme, RDC-P-ICR improves performance by 2.2% and 33.5%. However, due to cache line replication and replica maintenance, in general, the power consumption of the data cache and L2 cache in RDC-P-ICR is higher than in RDC-P. Figure 5.2 shows that the energy number for RDC-P-

ICR is around 19.5% higher than for RDC-P when the error rate is lower than e-3. However, the picture changes when the error rate reaches e-3, where RDC-P consumes significantly higher energy in the data cache and L2 cache than RDC-P-ICR. A noticeable feature of the RDC-P-ICR scheme is that it performs consistently well in terms of performance and energy consumption, across vastly different error rates.

RDC-P-ICR-EWB-CCI Data Cache

While the ICR scheme solves performance and energy issues related to the software (O.S.) recovery strategy at high error rates, the parity coded data cache still suffers from high silent data corruption (SDC) rates. Figure 5.3 shows that the SDC rate reaches 1.8% and 2.1% in the RDC-P and RDC-P-ICR schemes at the error rate of e-3. Two techniques, namely early write back (EWB) and clean cacheline invalidation (CCI), are considered for reducing the cache vulnerability to soft errors, and therefore the SDC rate.

Dead Time based Early Write Back: Previous work [41][59][51][58] has shown that the time between the last write and the replacement (WPL) of a cache data item contributes the largest part to the vulnerability factor. The dead time based early writeback scheme proposed in [40][41] writes back a dirty cacheline after it has not been accessed for a certain time interval. This scheme can reduce the WPL component while avoiding the dramatic increase in accesses to the L2 cache.

Clean Cacheline Invalidation: After applying EWB for optimizing the WPL phase, the RR time (the time between the first read and last read in a clean cacheline) arises as the major part in the vulnerability factor calculation of the data cache as discussed in Chapter 3. Therefore, the CCI technique is adopted for the RR phase optimization.

The Combined scheme: The RDC-P-ICR-EWB-CCI scheme is a combination of the parity coding, in-cache replication (ICR), dead time based early writeback (EWB), and clean cacheline invalidation (CCI). In the evaluation, a 1K-cycle interval for both deadness prediction and clean line invalidation is chosen. The work uses a similar implementation

as the cache decay scheme [61]. Each cache line maintains a 2-bit local counter which is ticked every 256 cycles by a global counter. Both the dead time based early writeback scheme and the clean cacheline invalidation scheme use the same local counter. The dirty bit controls whether a simple invalidation or an early writeback shall be performed when the local counter saturates. The local counter is reset to zero upon any access to the cacheline.

By reducing the time of cachelines staying in the vulnerable WPL and RR phases, the RDC-P-ICR-EWB-CCI scheme significantly decreases the possibility of data being corrupted by soft errors and being loaded by the CPU or written back to the L2 cache. Consequently, the occurrence of silent data corruption will also be dramatically reduced. Figure 5.3 shows that at the error rate of e-3, RDC-P-ICR-EWB-CCI achieves a significantly low SDC rate of 0.01%, compared to 1.8% and 2.1% for RDC-P and RDC-P-ICR, respectively. On the other hand, due to periodic EWB and CCI, the performance of RDC-P-ICR-EWB-CCI is about 4.6% lower than RDC-P-ICR, and the energy consumption of the data cache and the L2 cache increases by 8.6% over RDC-P-ICR.

5.3.2 Computing the Architectural Vulnerability Factors (AVFs)

Different from error-injection based reliability analysis, AVF analysis [72] is independent of the lower device-level error rate (usually in FITs, failures in one billion hours). This work uses a similar lifetime model as in [33] to compute and compare the AVFs for different reliable data caches discussed in this section. A single-bit error model is also assumed in this AVF study. Figure 5.4 presents the AVF numbers for different data caches, an average across the selected benchmarks. The AVF of a Base data cache is around 35.1%, all contributing to SDC AVF. With parity protection, all single-bit errors can be detected and some of them (in clean lines) can be recovered by the L2 cache. Thus, RDC-P reduces the ACE (architecturally correct execution) time and converts the SDC AVF into DUE (detected unrecoverable error) AVF. As shown in the figure, the DUE AVF is reduced to 23.9%. By duplicating dirty lines, RDC-P-ICR is capable of recovering errors in dirty

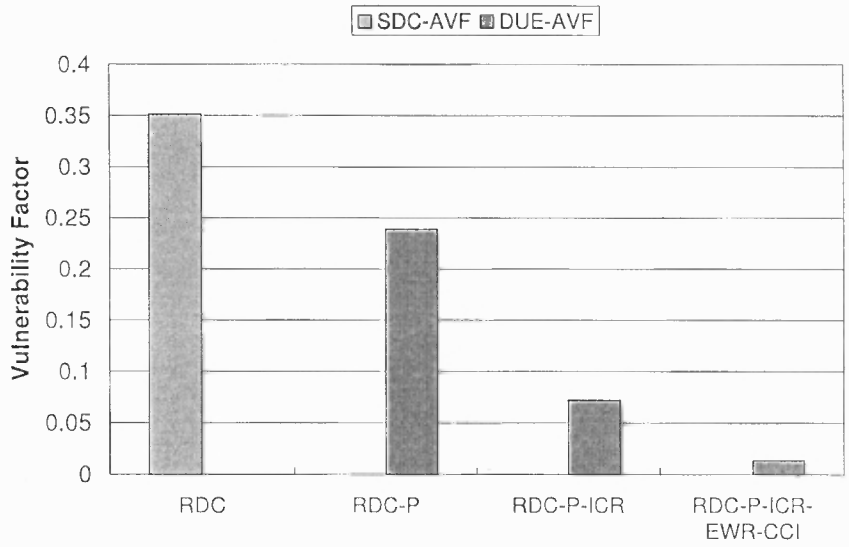


Figure 5.4 AVF comparison for different data caches.

lines provided their duplicates are error free, which improves the cache reliability with a significantly reduced DUE AVF of 7.2%. With EWR and CCI, the ACE time of the data cache can be further reduced and the RDC-P-ICR-EWB-CCI achieves an impressing AVF of 1.4%.

5.4 The Self-Adaptive Reliable Data Cache

5.4.1 Why Self-Adaptive Scheme?

For the three levels of reliable data caches proposed in the previous section, each could be the best cost-effective scheme at a particular error rate. For systems with stringent performance and energy constraints and changing operating environments (i.e., soft error rates), a fixed reliable scheme will be either insufficient (at harsh environments) or too costly (at low error rate environments) in terms of performance and energy overheads. Thus, a self-adaptive scheme will be of critical importance to meet the reliability requirements as well as performance/energy constraints.

5.4.2 A Soft-Error Monitoring Mechanism

Due to the limits of the conventional reliable data caches, this work proposes a self-adaptive reliable data cache (SA-RDC) based on the three levels of protection schemes, RDC-P, RDC-P-ICR, and RDC-P-ICR-EWB-CCI. The SA-RDC scheme dynamically monitors the number of detected errors during the parity check and logs the error occurrences into a special error counter during each monitoring window. Based on the value of the error counter at the end of each monitoring window and the currently applied protection scheme, the SA-RDC predicts the error rate that the system is currently experiencing and determines whether to change the protection scheme or just maintain the current scheme. The error counter is reset to zero at the beginning of each new monitoring window. From the study, a monitoring window of 100K cycles is chosen, the minimum window size to effectively change the protection scheme. Notice that the selection of the window size is strongly related to the error rates. For instance, in the case of more realistic low error rates, the window size should be much larger.

5.4.3 Control of Self Adaptation

After fixing the size of the monitoring window, the thresholds of error occurrences for triggering the different schemes also need to be determined. Therefore, this work simulated the different protection schemes at different error rates and profiled the number of errors detected in each 100K monitoring window. Based on the analysis of the profiling results, the threshold for upgrading from the RDC-P scheme to the RDC-P-ICR scheme is 4 errors. It means that if more than 4 errors are detected in the last window with RDC-P protection scheme, it is predicted that the current error rate might be larger than e-4 and the protection scheme needs to be changed to RDC-P-ICR in order to reduce the performance loss. If the current protection scheme is RDC-P-ICR and more than 16 errors are detected in the last monitoring window, it can be predicted that the current error rate is quite high such that double- or multi-bit errors will occur within a single byte. Therefore, the RDC-P-

ICR-EWB-CCI protection scheme should be invoked in order to reduce the silent data corruption.

To downgrade from the RDC-P-ICR-EWB-CCI scheme to the RDC-P-ICR scheme, this work uses the history information of last three consecutive monitoring windows. If there is no error detected in the last three monitoring windows, the protection scheme is degraded to RDC-P-ICR. From RDC-P-ICR to RDC-P, two consecutive monitoring windows are used. If no error is detected in the previous two monitoring windows under the RDC-P-ICR protection scheme, the scheme is downgraded to the RDC-P. Notice that the threshold is also strongly related to the error rates and the monitoring window size.

5.4.4 Microarchitecture of the SA-RDC

In order to implement the SA-RDC, the hardware should support all three schemes and the mechanism to switch between them. In fact, the hardware requirement of the SA-RDC is similar to that of the RDC-P-ICR-EWB-CCI scheme, since in the three chosen schemes, the higher level (more reliable) scheme is built on top of the lower level (less reliable) scheme, which makes it easy to integrate them together. Figure 5.5 gives the microarchitecture-level schematic of the SA-RDC. To support ICR, the priority encoder in the tag match logic is slightly augmented to generate both Primary_Hit and Duplicate_Hit way numbers. The figure shows that during a write operation the Duplicate_Hit way number is delayed by one cycle and data will be buffered in the write buffer once ICR is enabled such that writing the duplicate copy is performed in the following cycle. To support EWB and CCI, an N-bit global counter ticked by the clock signal and a per line 2-bit local counter ticked by the global counter every 2^N cycles are introduced. The local counter is reset to 0's once the cacheline is accessed. If the local counter saturates, either EWB is performed (if both valid and dirty bits are set) or CCI is performed (if valid bit is set and dirty bit is cleared), then the local counter is reset to 0. Notice that the global counter is enabled by signal EWB/CCI_en and it is the control unit of the SA-RDC that generates the ICR_en and

EWB/CCI_en signals to shut down or turn on ICR and/or EWB/CCI to adaptively choose a protection level. Thus the hardware overhead of supporting SA-RDC should be at the same level as the RDC-P-ICR-EWB-CCI scheme. Moreover, because all three schemes are based on parity coding, the parity bit and the encoding/decoding units are shared by the three schemes to reduce the hardware cost and complexity of the SA-RDC.

5.4.5 Evaluation of the SA-RDC Scheme

Figure 5.1, 5.2, and 5.3 show that the SA-RDC can self-adapt to the best scheme at a fixed error rate. For example, at the error rate e-3, SA-RDC performs nearly the same as RDC-P-ICR-EWB-CCI, which means it automatically tunes itself to the RDC-P-ICR-EWB-CCI scheme in order to reduce the SDC rate. Furthermore, in order to evaluate the efficiency of the SA-RDC scheme, this work also simulates SA-RDC under changing operation environments by varying the soft error injection rate. A soft error rate profile is randomly constructed as shown in Figure 5.6, which is similar to the one in [87]. The profile simulates a pattern of varying soft error rate at four levels between e-6 and e-3. Each error rate lasts for 10M cycles, then the pattern repeats.

Figure 5.7 compares the performance between the self-adaptive reliable data cache (SA-RDC) scheme and fixed RDC schemes for the varying error rate pattern shown in Figure 5.6. The large performance loss in RDC-P is mainly due to the huge recovery overhead at the error rate of e-3. Similarly, the performance degradation in RDC-P-ICR-EWB-CCI is caused by periodic early writeback and clean cacheline invalidation. RDC-P-ICR consistently performs better than other fixed RDC schemes since the error rate has less impact on the performance of RDC-P-ICR. The SA-RDC scheme adaptively adjusts the protection schemes to the detected error rate. It avoids to apply the RDC-P scheme at high error rates or the RDC-P-ICR-EWB-CCI scheme at low error rates, thus eliminating unnecessary performance loss. Figure 5.7 shows that SA-RDC achieves a performance within the 0.8% of the best scheme, RDC-P-ICR.

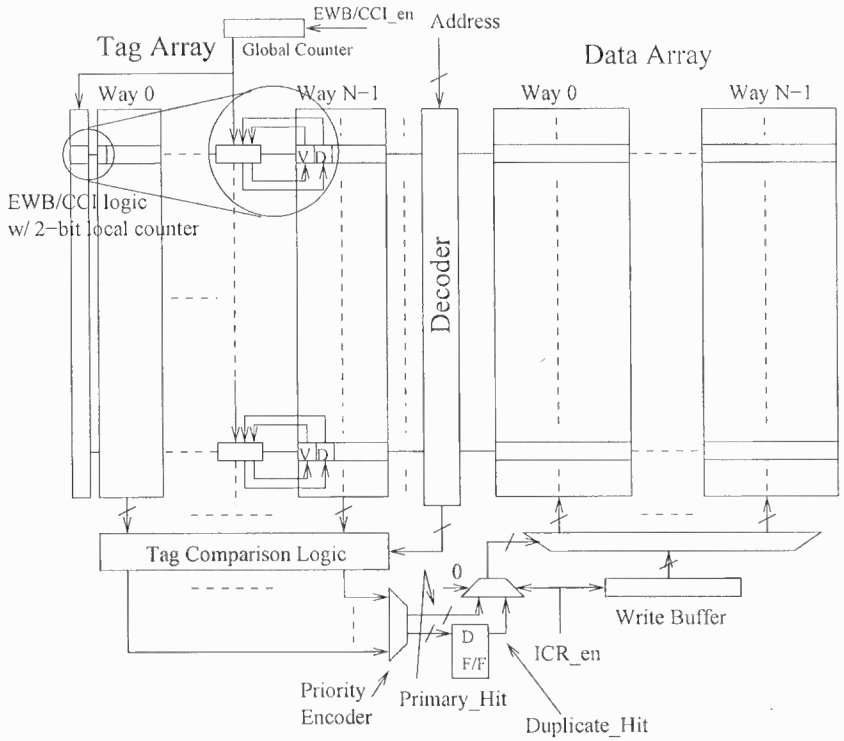


Figure 5.5 The microarchitectural schematic of the proposed SA-RDC.

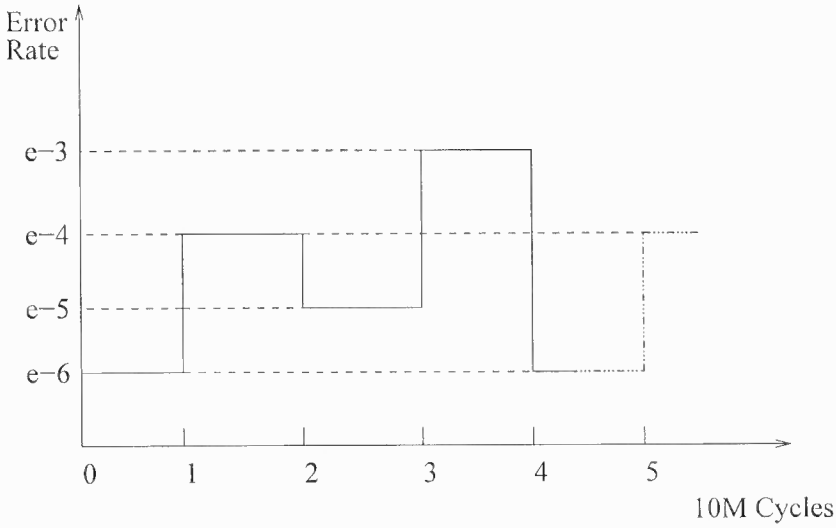


Figure 5.6 Soft error rate profile to simulate the changing error rate.

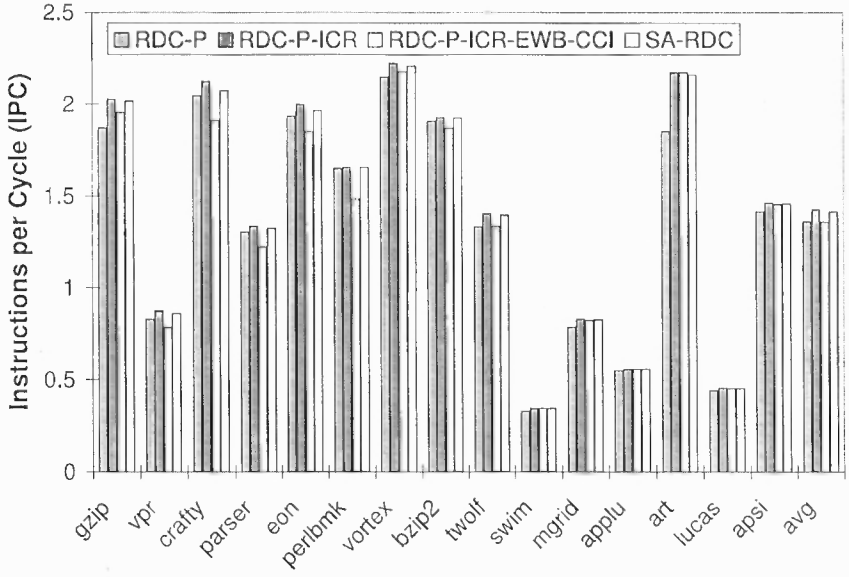


Figure 5.7 The performance comparison between the SA-RDC and fixed RDC schemes.

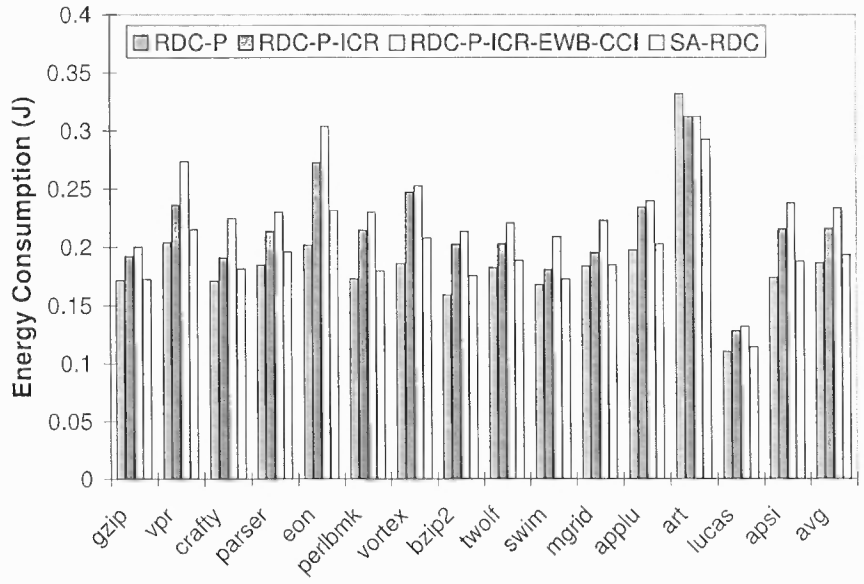


Figure 5.8 The energy consumption comparison between the SA-RDC and fixed RDC schemes.

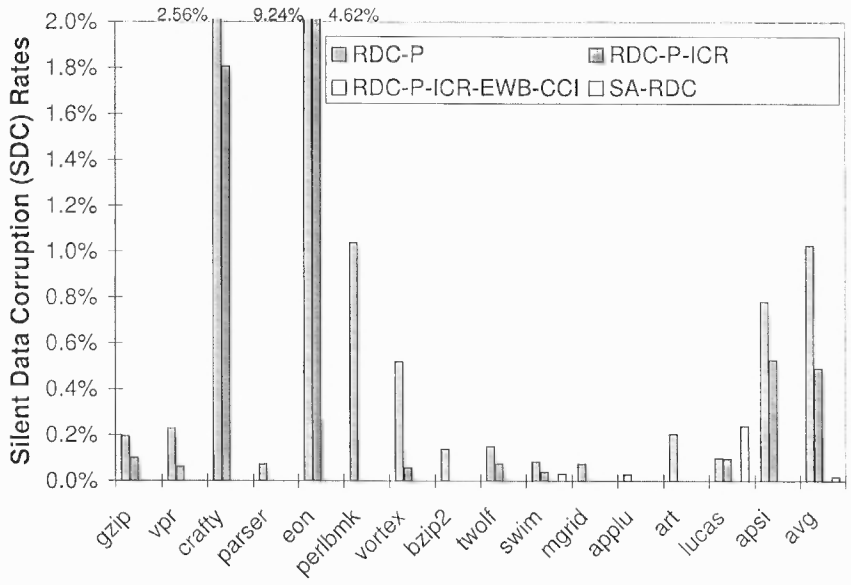


Figure 5.9 The SDC rate comparison between the SA-RDC and fixed RDC schemes.

However, one may argue that if SA-RDC always favors choosing the RDC-P-ICR scheme over others, it could still achieve the performance shown in Figure 5.7. To illustrate that the SA-RDC scheme does adapt itself by invoking different schemes at different error rates during the simulation, this work presents a comparison of the energy consumption of the data cache and L2 cache for all the schemes in Figure 5.8. Overall, the RDC-P scheme has the lowest energy consumption. By choosing the RDC-P scheme at low error rates and RDC-P-ICR-EWB-CCI only at high error rates, SA-RDC achieves a much lower energy consumption than RDC-P-ICR or RDC-P-ICR-EWB-CCI, which is only 3.7% higher than RDC-P.

Finally, this work presents the SDC rate comparison in Figure 5.9. Obviously, neither RDC-P nor RDC-P-ICR works well as the soft error rate varies between e-6 and e-3. On the average, the SDC rate is 1% and 0.5% for RDC-P and RDC-P-ICR, respectively. In contrast, the SDC rate is almost zero in RDC-P-ICR-EWB-CCI due to the effective early writeback and clean cacheline invalidation. In the meantime, RDC-P-ICR-EWB-CCI suffers from the worst performance loss and energy overhead, as shown in Figure 5.7 and 5.8. On the other hand, the SA-RDC achieves a significantly reduced SDC rate, 0.02%

on the average, while simultaneously minimizing the performance and energy overheads. These results confirm that the SA-RDC scheme is very effective for systems operating under changing environments.

5.5 Limitations of this study

First, as discussed in Chapter 5.2, the error rates used in this work are extremely high compared to real ones because of limitations in a simulation study. Second, the dynamic soft error model in Figure 5.6 is also quite different from real world situations where the changing speed and rate are much lower than assumed. This error model is just chosen for simulation purposes. To make the self-adaptive scheme work in the real world situations, the monitoring window size and the threshold need to be changed accordingly. Third, the cache access latency may increase by incorporating the protection schemes. However, a detailed study of this timing impact is out of the scope of this work.

5.6 Summary

To design the reliable systems in the operating environment with changing soft error rates, a new methodology is proposed in this work to adjust the applied reliability scheme to be the best match to the current erroneous situation. The proposed self-adaptive reliable data cache (SA-RDC) supports three levels of protection schemes targeting at different error rates with different performance and energy impacts. The simulation results show that this self-adaptive reliable data cache can achieve similar reliability to a cache protected by the most reliable scheme, while maintaining the minimized performance and energy overheads.

CHAPTER 6

IN-REGISTER DUPLICATION FOR ENHANCING REGISTER FILE RELIABILITY

6.1 Introduction

The presence of narrow-width data (with values that can be represented by fewer bits than the full data width of the processor) in general-purpose applications is well understood and has been utilized for power and performance optimizations [42][62][63][43]. This work proposes to exploit the produced narrow-width register values for designing high-performance error-resilient register files, and protecting the result writeback bus and the bypass network. In the proposed new processor microarchitecture, the existing leading-0/1 detection logic within the functional units is utilized for narrow-width check. Detected narrow-width results that can be represented by no more than 32 bits automatically duplicate themselves in 64-bit processors by muxing (copying) the lower 32 bits into the higher 32 bits before being latched by the pipeline registers. This scheme proposed scheme is called *In-Register Duplication (IRD)*. IRD stores two copies of the narrow-width value in the same register and transmits these two copies of the value using the bandwidth for a single data value over the writeback bus and forwarding bus. Thus, IRD eliminates the need for additional (copy) registers that maintain redundant copies of the register value for error detection and recovery. It also protects the data transfer paths from/to the register file and the functional units for narrow-width values.

To evaluate the effectiveness of the proposed IRD scheme, this study conducts both architectural vulnerability factor (AVF) measurements for register files with and without IRD, and experimental evaluation under software-implemented soft error injection. The experimental evaluation shows that without sacrificing any performance IRD achieves a write-with-duplicate (WWD) rate of 94% at the output of functional units and a read-with-

duplicate (RWD) rate of 95% at the inputs of functional units. In the meantime, IRD only incurs a small 8.8% increase in the register file power consumption. Based on a detailed register lifetime model, the AVF analysis shows that the IRD scheme achieves a dramatic reduction of 98.8% in register file AVF, from 8.4% to 0.1%, on the average. Under error injection with accelerated error rates of $10^{-5}/10^{-4}$ per selected bit per cycle, IRD schemes detect virtually all errors in narrow-width and regular values being read in. To avoid signaling unnecessary errors in the duplicate copy, IRD is further tuned to only check the parity bit of the lower 32-bit half for error detection and utilize the duplicate in the upper half for error recovery. The experimental results show that IRD detects 99.7% of the erroneous reads for narrow-width values and successfully recovers 99.7% and 99.2% of detected errors at error rate 10^{-5} and 10^{-4}, respectively, using the uncorrupted duplicate, which makes the in-register duplication a very cost-effective design for highly reliable register files.

6.2 Basics of Register Renaming in Superscalar Microprocessors

6.2.1 Register Renaming

This work implemented MIPS R10000 [12] style register renaming, where the architectural and physical register files are combined. Figure 6.1 gives the superscalar microprocessor model simulated in this paper. Notice that a physical register is susceptible to soft errors only after a value is written into the register and before it is freed.

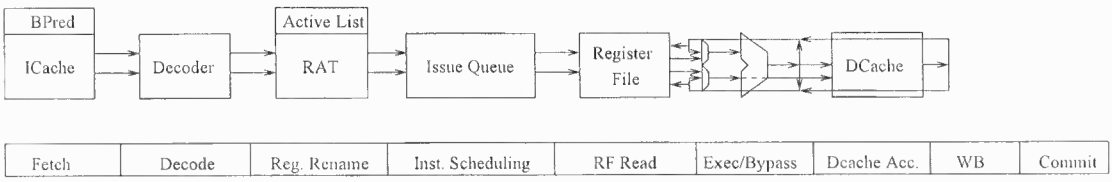


Figure 6.1 Datapath and the pipeline stages of the simulated superscalar microprocessor.

6.2.2 Register File Utilization and Performance Sensitivity

Since each logical register may maintain multiple active physical registers along its unmapping chain, the physical register file may experience high utilization when the issue rate is limited by the instruction-level parallelism exploited. Under such a situation, the physical register file becomes a critical resource whose size limits the number of instructions on-the-fly that a microprocessor can accommodate and explore ILP from. Figure 6.2 shows the accumulative distribution of the register file utilization for different sizes of integer register files, an average for SPEC CINT2000 benchmarks [49]. This distribution is derived from profiling the number of active (non-free) registers at each cycle during program execution. To cover 90% of the execution time [88], register files with size $<= 256$ require the full register file in use. A large register file with 512 entries will need 296 registers in the active state to cover this 90% of the execution time. This result confirms that the register utilization is quite high for small- and medium-size register files.

While the register file size limits the effective size of the instruction window, it presents a major constraint on ILP exploitation in superscalar microprocessors. Figure 6.3 shows a performance comparison for an 8-wide superscalar microprocessor when the integer register file size varies from 40 to 512 entries. Significant performance improvement is achieved when the size increases from 40 to 48, 64, or 80. However, further increasing the size beyond 80 registers, the performance improvement is diminished. Notice that this study has assumed a uniform access latency for the integer register file at different sizes. A reliable design based on full register duplication will either require large size register files or will significantly limit the success rate of duplication. The following study focuses on a medium size integer register file of 128 entries, which can simulate the register pressure in wide-issue datapaths while avoiding the unnecessary performance overhead of the full-duplication scheme.

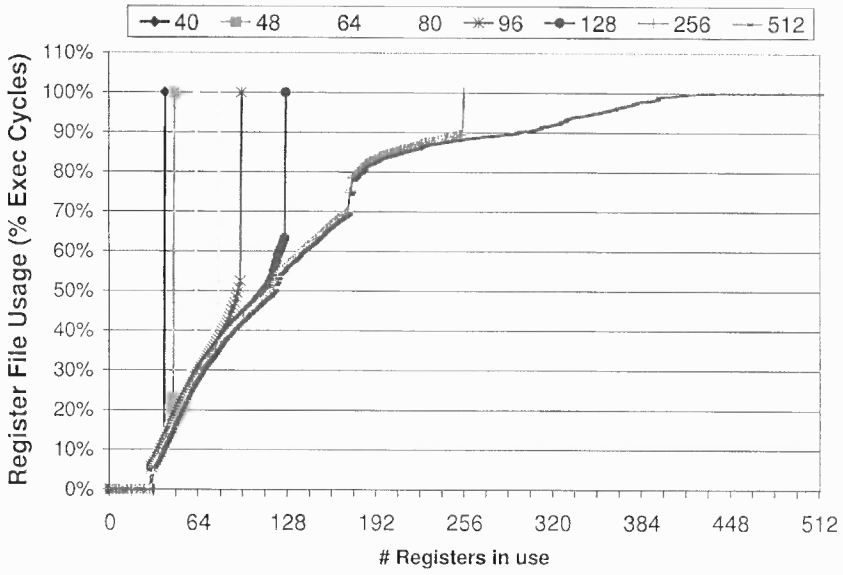


Figure 6.2 A cumulative distribution of register file utilization for different sizes of register files.

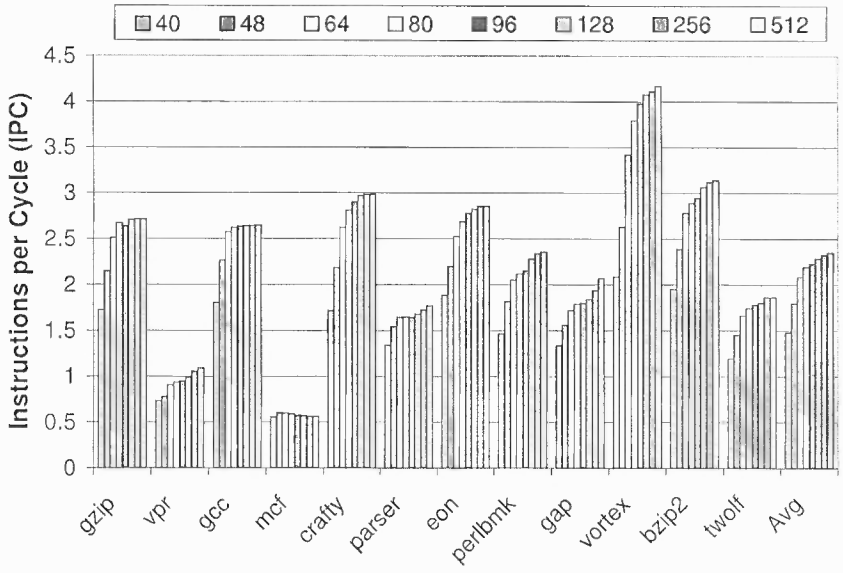


Figure 6.3 Performance sensitivity to the register file size.

6.3 Narrow-Width Register Values

In high-performance 64-bit microprocessors, many generated register values during the execution of general-purpose applications do not require the full width of 64 bits. Values that can be represented by less than 64 bits are called narrow-width values in this paper. The presence of narrow-width values has been well studied and exploited for performance and power optimizations [42][62][63][43]. Different from the previous work, this study exploits narrow-width register values for improving the register file reliability against soft errors.

The detailed cumulative data-width distribution given in Figure 6.4 shows that on the average 54% of the generated register values have a data-width no more than 32 bits, and the difference between 32 bits and 33 bits is negligible. However, there is a significant 40% jump from 33 bits to 34 bits. This is because the memory address in the Alpha ISA uses 33 bits (plus 1 sign bit = 34 bits) and memory operations account for a large portion of the executed instructions. Please notice that 1) the operations generating these memory addresses are different from the address calculation in a load/store instruction, and 2) compiler options or large-size programs may change the data width of memory addresses. Overall, around 94% of the integer values can be represented by no more than 34 bits, an average for SPEC CINT2000 benchmarks, which are exploited in this work for designing high-performance error-resilient register files using in-register duplication.

6.4 Exploiting Narrow-Width Register Values

In this section, reliable register file design that exploits the generated narrow-width register values is presented. Information redundancy is the basic idea for protecting memory structures against soft errors. Instead of duplicating each register value into two registers, this work exploits the majority of narrow-width values ($\leq$ 32 bits and 34-bit memory addresses) to perform in-register duplication.

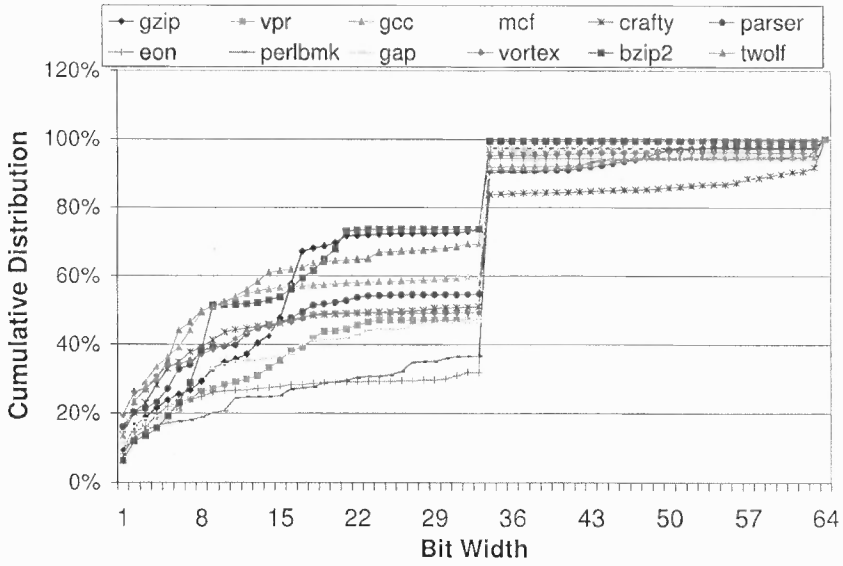


Figure 6.4 Cumulative distribution of the Register value width.

6.4.1 Narrow-Width Value Detection

Based on the data-width analysis presented in the previous section, this design is particularly tuned to capture three types of narrow-width values: 32-bit positive values ($0^{32}0x^{31}$), 32-bit negative values ($1^{32}1x^{31}$), and 34-bit memory addresses ($0^{30}01x^{32}$), where x can be either a "1" or a "0". From now on, only these three types are referred to as narrow-width data. The specific bit patterns of narrow-width data are given in Figure 6.5.

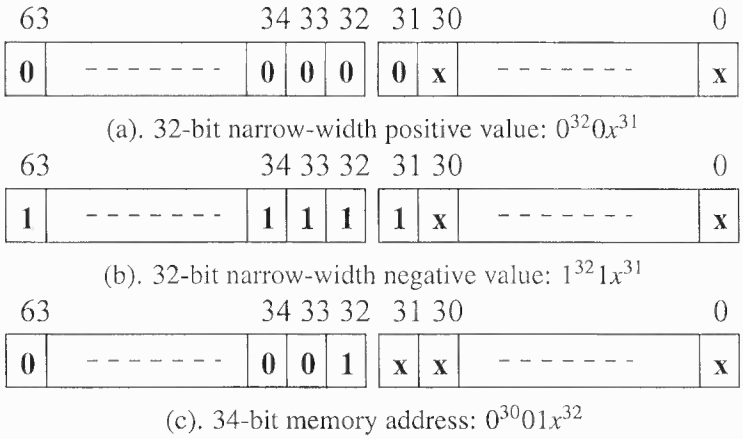
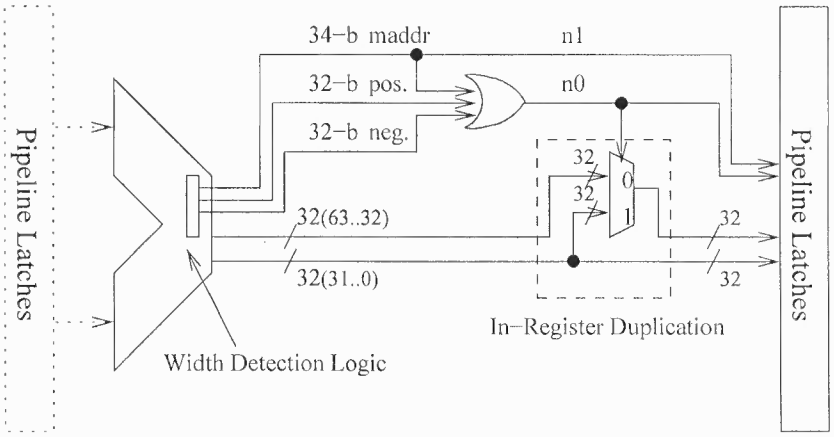


Figure 6.5 Bit patterns for three types of narrow-width values considered: (a). 32-bit positive value, (b). 32-bit negative value, and (c). 34-bit memory address. An "x" bit can be either "1" or "0".

To capture narrow-width values, the internal signals are extracted from the existing leading-0/1 detection logic within the functional units [89] (in order to minimize its timing overhead in deeply pipelined designs at new technology generations [90]), indicating whether the newly generated result from the functional unit is a 32-bit positive value, a 32-bit negative value, or a 34-bit memory address (positive value). After detection, two flag bits ($n_1 n_0$) associated with each register value are set to indicate the narrowness of the current value. The meaning of these two $n_1 n_0$ bits is given in Figure 6.6 (b). The block diagram in Figure 6.6 (a) shows a slightly modified datapath with added logic for setting the flag bit n_0 and in-register duplication. Notice that a narrow-width value will have the flag bit n_0 set to 1. The in-register duplication logic (the Mux in the figure) is controlled by flag bit n_0 to either perform duplication for a narrow-width value (by copying the lower 32-bit half into the higher 32-bit half) or bypass duplication for a regular value.



(a)

n_1	n_0	meaning
0	0	regular value
0	1	32-bit pos./neg. narrow-width value
1	1	34-bit narrow-width memory address
1	0	reserved

(b)

Figure 6.6 (a) Augmented functional unit datapath with narrow-width flag generation and in-register duplication logic. (b) The meaning of the value of narrowness flag bits $n_1 n_0$.

6.4.2 Exploiting In-Register Duplication for Error Detection

Once a narrow-width register value is detected, in-register duplication is automatically performed by copying the lower 32-bit half into its higher 32-bit half such that two copies of the value will be latched into the pipeline register. The incentive of this reliable register file design is not only to protect the register file against soft errors, but also to guarantee reliable data transmission over the writeback and bypass networks. In-register duplication enforces at any time two copies of the narrow-width value to be stored in the register file, latched by the pipeline register, or transferred between the register file and the functional units.

It is important to notice the significant difference between the in-register duplication and conventional redundancy-based reliable designs. In-register duplication incurs much less hardware complexity compared to schemes utilizing idle or predicted dead registers for duplicating a data value, where the register renaming logic needs to be redesigned for copy register allocation, the instruction queue is augmented to hold copy register ids, and the number of register file writeports is doubled or a set of copy ports is required [35]. In-register duplication needs none of the above hardware modifications. More importantly, this scheme also protects the result writeback bus and the bypass network by transferring two copies of the value without increasing the bandwidth requirement. In the schemes presented in [35], a data value hit by errors when transferring over the writeback bus will result in two corrupted copies being stored in the register file due to the use of copy ports. Since around 50%-70% of the input operands are retrieved from the bypass network, hardening both the bypass network and the writeback bus against soft errors is of critical importance, which is naturally supported by in-register duplication.

Since the probability of the two copies of the narrow value being corrupted at exactly the same bit position is negligible, the two copies can be compared against each other to verify the absence of soft errors very effectively. A follow-up question is when to perform this comparison. Notice that soft-error corrupted data only matters when used later in

computation or written out to the memory hierarchy. The probability of resulting in crashed execution or erroneous outputs can be estimated by the architectural vulnerability factors (AVFs) [72] of the microarchitectural blocks that the corrupted value is going through. The error detection (comparing the upper 32 bits against the lower 32 bits of the input operand) is performed at the execution stage when the operands are fed to the functional units. Notice that narrow-width operands are restored into full-width regular values at the inputs of the functional units. As shown in Figure 6.7. the restoration logic, basically a Mux, is controlled by the 2-bit narrowness flag either to sign extend the lower 32-bit half or to reform the memory address for narrow-width values, or to bypass the upper 32-bit half for regular values.

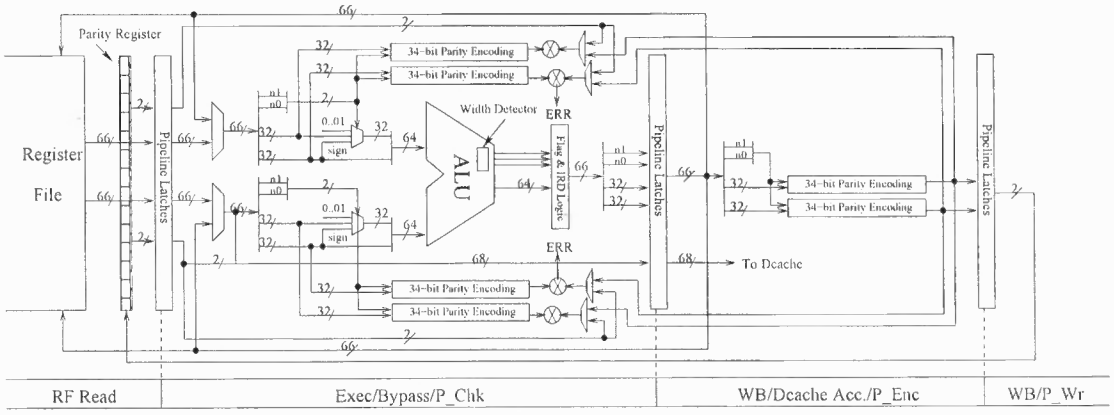


Figure 6.7 The augmented datapath integrating in-register duplication and parity coding to support both error detection and error recovery.

6.4.3 Integrating In-Register Duplication and Parity Coding

In-register duplication itself is expected to be very effective in soft-error detection however, is not capable of recovering from an error. Since in-register duplication already maintains two redundant copies of the value, providing ECC coding (e.g., Hamming coding) for each 32-bit half is either over-designed or not feasible considering the ECC coding/checking latency and power consumption [28]. This work uses simple and fast parity coding to supplement each 32-bit half with an additional parity bit. Notice that the flag bits are

included in the parity coding for both 32-bit halves, thus covered by the same parity bits for the data value. It is assumed that parity encoding/checking takes one clock cycle.

To integrate parity coding with in-register duplication, a separate pipeline stage needs to be added to perform parity encoding after the execution stage. Figure 6.7 shows the modified datapath supporting both error detection and recovery. The parity bit for each 32-bit half (and narrowness flag) is generated in the parity encoding (*P_Enc*) stage. Parity checking (*P_Chk*) for input operands is overlapped with the first cycle of the execution stage such that the branch resolution loose loop [91] is not increased. This also guarantees that detected errors in input operands are signaled before the erroneous result is written back to the register file since many ALU operations take just one cycle to complete. Input operands read from the register file come with the parity bits for the two 32-bit halves (and 2-bit flag). Parity checking basically regenerates the parity bit for each 32-bit half (and 2-bit flag) and compares it against the one with the data value. However, operands retrieved from the first stage of the bypass network do not have parity bits generated yet. In such a case, both parity encoding (*P_Enc* stage) and parity regenerating (in *P_Chk* stage) are performed simultaneously and the parity bits from the *P_Enc* stage are bypassed to the *P_Chk* stage for parity bit checking since the comparison happens in the latter stage of *P_Chk*. If the two parity bits for the lower 32-bit half (and 2-bit flag) match, no error is detected. Otherwise, the lower half has been corrupted by errors and a stall cycle is inserted. Now if the parity bits for the upper 32-bit half (and 2-bit flag) match, then the upper half is copied back to the lower half to recover the corrupted data. The instruction is then replayed with the recovered inputs. However, if the upper half is also corrupted and the error is detected, an exception is raised for the higher level system(s) to solve the problem.

Since parity encoding takes one additional clock cycle, one design issue raised here is when to write back the result value and the parity bits. To avoid increasing the complexity of the register file read/write ports or the bypass network, This work proposes to use a special bit-addressable parity register to hold two parity bits for each entry in the register

file, as shown in Figure 6.7. The parity bits are written into the parity register at *P_Wr* stage.

IRD with parity coding is expected to be very effective in detecting and recovering single-bit errors in narrow-width values. In the presence of multi-bit errors (at a rate of several orders of magnitude lower than single-bit errors), a more aggressive detection scheme combining parity checking and duplicate comparison can be employed. Due to the extremely low possibility that the two copies are corrupted by multi-bit errors at exactly the same bit locations, multi-bit errors in narrow-width values can be effectively detected by duplicate comparison. However, IRD may lack the capability of recovering from detected multi-bit errors.

6.4.4 Protecting Regular Values

As a side benefit of in-register duplication, regular values (those that cannot be represented by 32 bits plus 2 flag bits) are also protected by the 2 parity bits. For a detected regular value, the 2 flag bits $n1n0$ are reset to 00. During the *P_Enc* stage, two parity bits are generated for the two 32-bit halves (and flag bits) in the same way as for narrow-width values. Once a regular value reaches the input of a functional unit, the flags bits $n1n0$ (=00) enforce parity checking for both 32-bit halves to verify the absence of soft errors. If any half fails the parity check, an error signal is raised. However, the hardware itself is not capable of recovering the error-corrupted regular value. Notice that a similar scheme as in [35] can be applied to exploit free registers for duplicating a replica of the regular value, which provides recovery capability. In such a scheme, the mapping information between the original register and the copy register shall be maintained in order to locate the copy register during recovery. Due to significant modifications required in the register renaming logic, the register file, and the issue queue, this idea is not further explored in the following discussion.

6.5 New Models for Register File AVF Estimation

To estimate the register file AVF, the ACE (architectural correct execution) and un-ACE cycles of each register value residing within the register file need to be calculated. Inspired by a previous work [33] that uses lifetime analysis to compute the AVF for address-based structures exemplified by a data cache, a data translation buffer, and a store buffer, this work exploits the register lifetime model as the basis for AVF estimation. As shown in Figure 6.8, the lifetime of a physical register starts with the `Idle` state when it is in the free list. Once the register is allocated to rename a logical (destination) register at the renaming stage, it changes from the `Idle` state to the `Busy` state. The register stays in the `Busy` state till the result value is written into it. The time between the write and last read to the register is referred to as the `Live` phase. After its last read, the register enters its `Dead` state. The physical register is then freed when its unmapping instruction commits. Freeing a physical register puts it back to the free list and returns it to the `Idle` state. The register lifetime model clearly indicates that except the `Live` state, all the other states in a register's lifespan are un-ACE, i.e., the register in these states has no impact on the correctness of the processor architectural state. This is simply because the register either does not contain valid data or its valid value will not be used by any later computation if the register is in the `Idle`, `Busy`, or `Dead` states. Thus the `Live` phase presents an upper bound of the register's ACE cycles. Consequently, a conservative design for a reliable register file would be protecting the register value during its `Live` phase, while most existing proposals [35] allocating copy registers at the renaming stage are clearly over-kill designs.

For more accurate ACE calculation in the register file, a more detailed and comprehensive analysis model of the `Live` register value is required. This work proposes a new register value classification for ACE calculation purposes. In this classification, a register value is either speculative (i.e., produced by a speculative instruction) or non-speculative (i.e., produced by a non-speculative instruction). Obviously, a speculative register value will never be committed and thus it is un-ACE. A non-speculative register value can be

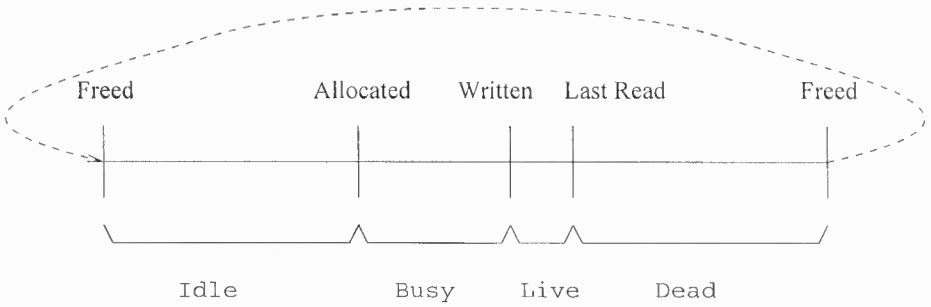


Figure 6.8 The lifetime model of a physical register.

dynamically dead (DD) during execution because it is either first-level dynamically dead (FDD) or transitively dynamically dead (TDD). Different from previous study [92][72] that identifies dynamically dead (both first-level and transitively) instructions for issue queue AVF estimation, this study sees new challenges in determining FDD or TDD register values. A register is first-level dynamically dead if 1) all its consumer instructions are speculative ones, referred to as FDD_S, or 2) it is not read by any instruction before being freed, referred to as FDD_N. Notice that FDD_N registers have a zero `Live` cycle. TDD registers can be further divided into three groups: 1) TDD_S due to all consumers falling into the TDD_S, FDD_S, and/or speculative ones, 2) TDD_N due to FDD_N and TDD_N consumers, and 3) TDD_NS due to a combination of TDD_S and TDD_N consumers. If a register cannot be determined as dynamically dead, it is conservatively assumed to be ACE. The detailed register classification is given in Figure 6.9. All registers in categories other than *ACE Reg* are un-ACE registers.

The question is: does the `Live` time of an ACE register correspond to all ACE cycles? Not necessarily. The reason is quite straightforward: an ACE register can be consumed as the source operand in producing both ACE registers and un-ACE registers. A further breakdown of the `Live` time of an ACE register is given in Figure 6.10. During its `Live` time, a register is to be accessed once or multiple times by its consumers. The last read to the register ends its `Live` phase. If the last ACE read (by an ACE instruction) to the register is different from its last read, then the time between the last ACE read and the

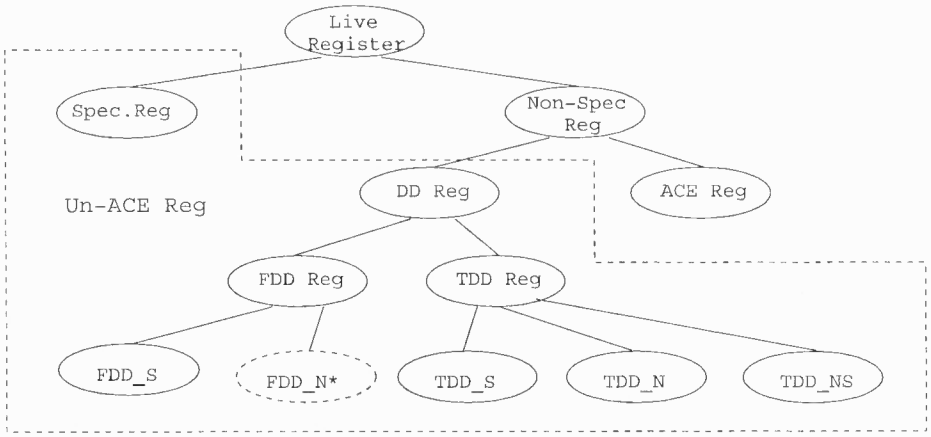


Figure 6.9 Register level ACE analysis and register value classification for Live registers. (DD: dynamically dead, FDD: first-level dynamically dead, TDD: transitively dynamically dead)

last read is considered as un-ACE cycles, while the remaining is ACE cycles independent of possible un-ACE reads before the last ACE read. Thus, the AVF of a register (of this renaming instance) is calculated as the percentage of its ACE cycles over its overall lifetime. Notice that each bit of an ACE register within its ACE cycles is counted as ACE for simplicity without further exploring the masking effects of ALU operations performed on the register value.

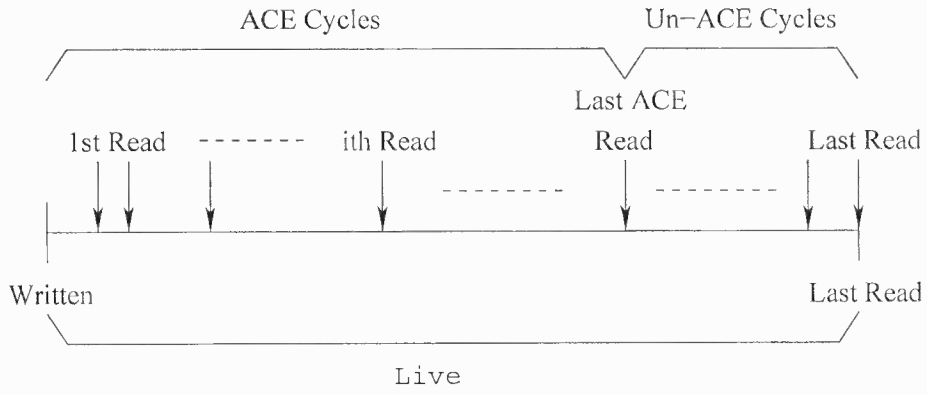


Figure 6.10 Extracting un-ACE cycles from the `Live` phase of an ACE register.

For a base register file without any error detection/protection scheme, errors happening during a register's ACE cycles are likely to result in silent data corruptions (SDCs). Thus, the AVF of the base register file is also its SDC AVF [72][86]. If each register entry

is protected by parity coding and only single-bit soft errors are assumed, then all the errors can be detected by the parity checking logic; however, they cannot be recovered. In other words, parity coding the register file converts all SDC errors into detected unrecoverable errors (DUEs). Consequently, the SDC AVF of a parity protected register file turns into zero and its AVF is now DUE AVF. With the proposed IRD scheme, the ACE cycles of a register holding a narrow-width value are almost halved from the base case, since the upper 32 bits of the register only contains the duplicate of the narrow-width value. The 2-bit narrowness flag should be also considered as ACE bits in this case. Notice that in the IRD register file, any single-bit error to a register storing a narrow-width value will be detected and corrected by the duplicate. Thus those "ACE" cycles will not contribute to the AVF of the IRD register file. Only a regular value (still protected by parity coding) will introduce DUE AVF. Due to the high percentage of narrow-width register values, the IRD scheme will significantly reduce the register file AVF and thus its vulnerability to soft errors.

6.6 Evaluation

6.6.1 Duplication Rates and Performance Impact

The ability to recover register values from detected errors depends on the availability and correctness of duplicate copies. The write-with-duplicate (WWD) rate is used as a first-order estimation to measure the capability of a reliable scheme to duplicate the register values, and use the read-with-duplicate (RWD) rate as a first-order estimation to approximate reliable reads of register values against soft errors. Figure 6.11 shows that by exploiting narrow-width values alone, in-register duplication achieves a WWD rate of 94% and a RWD rate of 95%, an average across SPEC CINT2000 benchmarks, without any performance loss. This RWD rate is significantly improved over the results (78% for CE, and 84% for AG at a 0.2% performance loss) reported in [35]. In the mean time, the in-register

duplication scheme avoids the hardware complexity of the latter schemes.

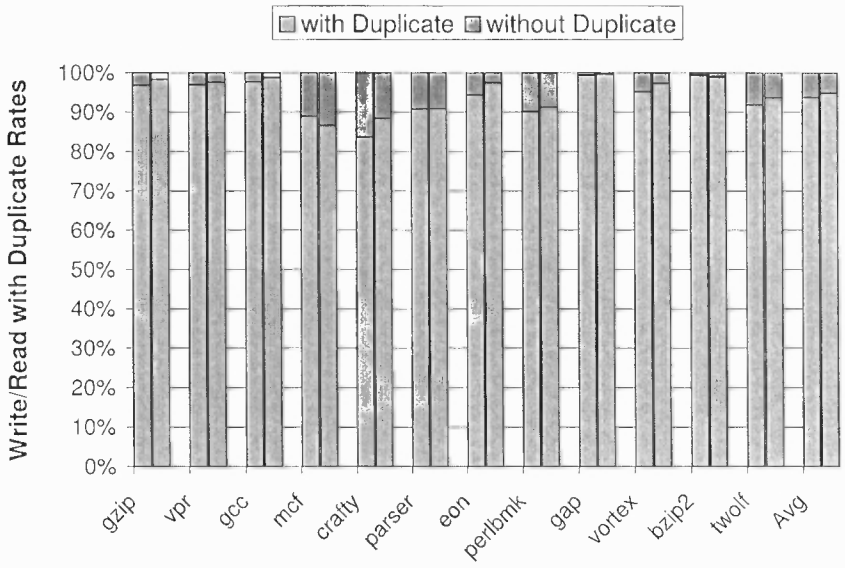


Figure 6.11 Write-with-duplicate rate (left bar) and read-with-duplicate rate (right bar) of the in-register duplication scheme.

Considering a full-duplication scheme (`Full_Dup`) that allocates a copy register for each result register at the register renaming stage, the hardware implementation is much more complexity-effective than the CE/AG schemes [35] since the copy register is implied. The `Full_Dup` scheme achieves a rate of 100% for both WWD and RWD; however, it suffers from a significant performance loss, 7.7% on the average, as shown in Figure 6.12. The in-register duplication `IRD` scheme duplicates the narrow-width value within the same register and requires no additional copy register, thus incurring no performance overhead. Figure 6.11 and Figures 6.12 together show that the proposed schemes are very effective in providing a high error coverage for applications where the performance and cost are highly constrained.

6.6.2 Power Efficiency of the IRD Register File

To evaluate the impact of the IRD scheme on register file power consumption, this work extended the Wattch power model [47] to include power profiling for the physical register

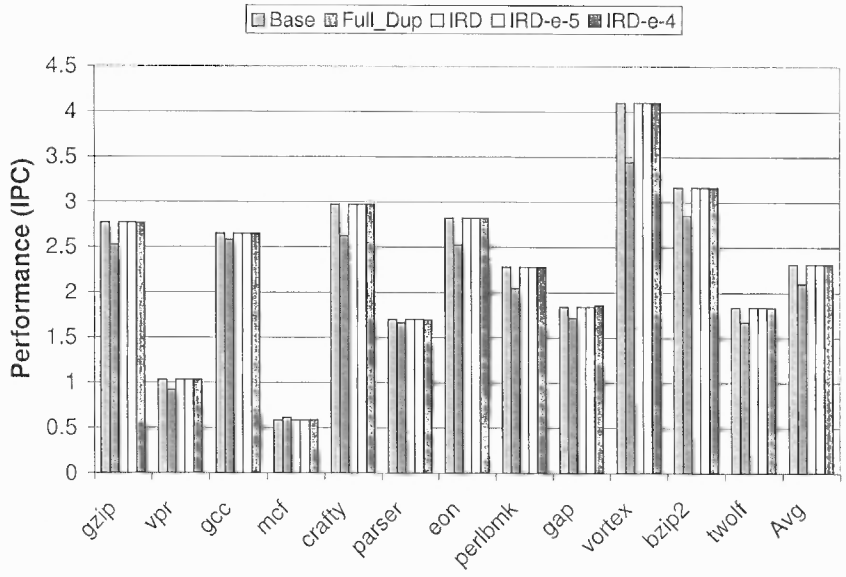


Figure 6.12 Performance comparison of various register file schemes.

file. Compared to the base register file without any protection scheme, IRD requires an additional 2-bit narrowness flag to be transferred and stored with each register value. Integrating parity coding has added two additional parity bits for each register entry. Notice that the parity bits are not stored with the value in the register file for the reason discussed in Chapter 6.4.3. Since this work exploited input narrowness information and functional unit internal signals for simple fast narrow-width detection, the major power overhead due to the IRD scheme will be from transferring this 2-bit narrowness flag on the result bus and writing/reading the flag to/from the register file. Similarly, the parity scheme also introduces power penalties due to parity encoding and decoding (parity check) as well as transferring two parity bits (one for each 32-bit half plus 2-bit narrowness flag) on the result bus. The power models of the register file and result bus are augmented to include the 2-bit flag and two parity bits. The published parity encoding/decoding power numbers are borrowed from [29] to approximate the power consumption of the (34, 1) parity scheme in the IRD register file. All the power numbers are scaled to the 70nm technology for the simulated microprocessor. Figure 6.13 breaks down the power consumption of the IRD register file. On the average, the IRD scheme alone only causes a 4.5% power increase (due to the narrowness

flag in the register file and data bus, `flag_in_RF` and `flag_in_DB`) to the base register file, `Base_RF`, and the parity scheme is responsible for additional 4.3% increase (1.3% for `parity_in_DB` and 3.0% for `parity_Coding`). Overall, the IRD register file is only 8.8% higher in power consumption than the base one, while significantly improving register file reliability.

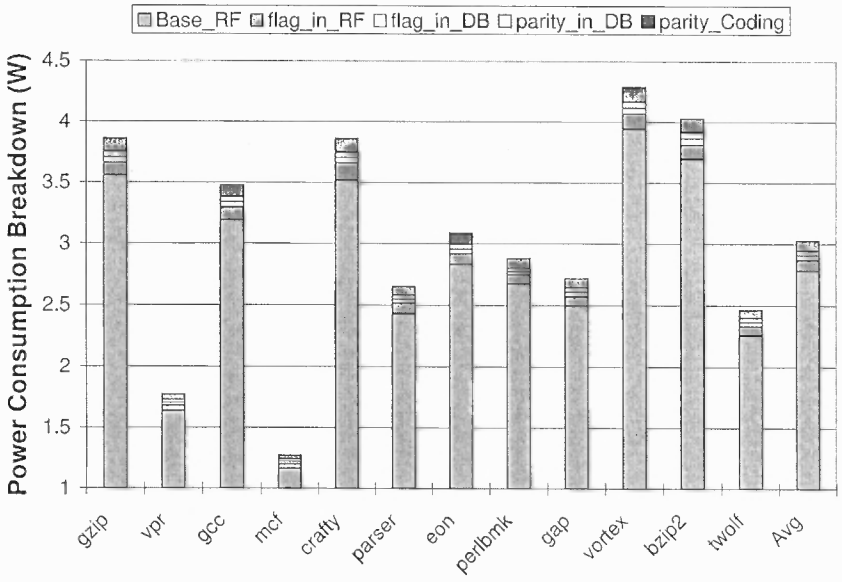


Figure 6.13 A breakdown of the power consumption in the IRD register file.

6.6.3 Register File AVF Estimation

To estimate the AVF of the register file, this work introduces a register AVF analysis window with 50,000 entries to record information (e.g., lifetime information, reads, and source registers, etc., required for ACE calculation) for the past 50,000 register values produced by non-speculative instructions. Notice that whether a non-speculative register value is dynamically dead (un_ACE) or not can only be determined by future instructions on its dependence chains. Following the AVF analysis model proposed in Chapter 6.5, the AVF analysis results of the Base register file are presented in Figure 6.14. Among the register lifetime, on the average, the `Idle`, `Busy`, and `Dead` states account for 32%, 21%, and 37%, respectively. `Unknown` represents those that cannot be determined upon completion

of the simulation, which is a very small portion, less than 1% in the Figure. The remaining 9% is contributed by the Live state, which is the sum of the live time of Spec_Live, FDD_S, TDD_S, TDD_N, TDD_NS, and ACE registers. ACE is further broken down into AVF_ACE and AVF_un-ACE cycles. As shown in the Figure 6.14, components of the Live time other than AVF_ACE form a total of less than 1%. Thus, the AVF of the register file in the simulated microprocessor when running SPEC 2000 CINT benchmarks is 8.4%, the percentage of AVF_ACE in the overall register lifetime. If it is the base register file without any protection scheme, its AVF only consists of SDC AVF, which is 8.4% in this case. While protected by parity coding, the register file converts SDC AVF into DUE AVF. Reliable register file designs shall target at reducing AVF_ACE for register file AVF reduction and reliability improvement. The employment of the proposed IRD scheme successfully eliminates the AVF_ACE cycles of a register holding a narrow-width value, since the error will be detected by the parity logic and corrected using the duplicate stored in the upper half of the same register, under the assumption of the single-bit error model. The eliminated AVF_ACE portion is referred to as AVF_DUP in the IRD register file, which is no longer ACE. The remaining part constitutes the true ACE and contributes to DUE AVF. As shown in Figure 6.15, the IRD scheme removes 98.8% of the original AVF_ACE cycles, which dramatically improves register file reliability by reducing the AVF from 8.4% in the base register files to 0.1% in the IRD register file.

6.6.4 Error Model and Soft Error Injection

To further evaluate the error resilience of the proposed schemes, this work also conducted soft error injection during the execution-driven simulation. Software-based error injection flips one bit or multiple bits in a selected register value. Since the multiple-bit error rate is several orders of magnitude lower than the single-bit error rate [40], a single-bit error model is assumed in this study. The error injection scheme simulates single-event upsets (SEUs) in the register file, the bypass network, and the result writeback bus. At each clock

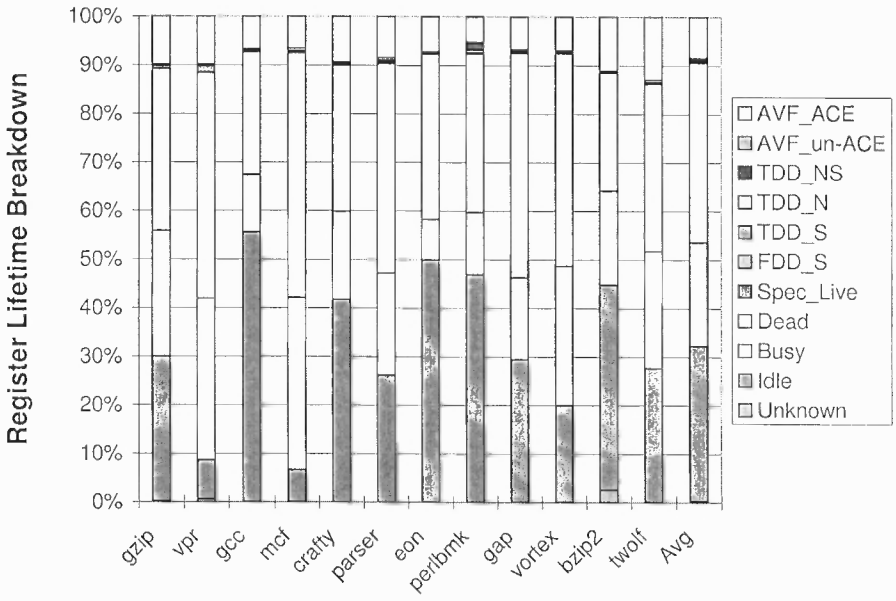


Figure 6.14 Register lifetime breakdown for AVF measurement in a base register file, a zoom-in view of its Live phase breakdown.

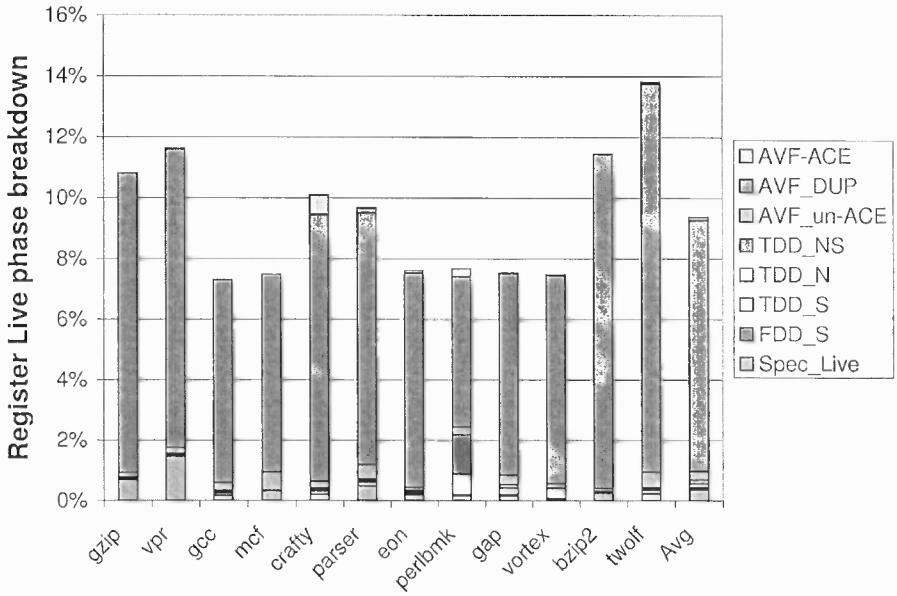


Figure 6.15 Register lifetime breakdown for AVF measurement in the IRD register file, a zoom-in view of its Live phase breakdown.

cycle, a uniformly distributed random function is called to locate a register and a specific bit in that register. Then, an error is injected with a given probability (e.g., 10^{-7} [40]), i.e., single-bit soft error rate. During error injection, if the selected register is receiving a new value which is also being bypassed to the next execution stage, the bit wire in the bypass network is flipped instead of the bit cell in the register file. Thus error detection and recovery can be immediately exercised at the *P_Chk* stage. If the selected value is transmitting over the result bus, error injection also flips the corresponding bit wire in the result bus and the error is propagated to the register entry in the register file once the value is written. Otherwise, a bit cell in the register file is flipped to reflect the error-corrupted bit value. Notice that each register file write clears out the errors previously injected into that particular register entry.

To avoid crashing the simulation, each injected error is logged using a bitmap for each register entry instead of flipping the real bit value and the error history is also recorded. During simulation, the soft error bitmap and error history information of a given register value are used to perform error detection and recovery at the execution/P_Chk stage.

6.6.5 Error Behavior under Soft Error Injection

This work evaluated the IRD schemes with a wide range of error rates (per bit per cycle) from 10^{-7} (suggested in [40]) to 10^{0}. Due to the limited simulation of 100 million instructions, error injection with a rate of 10^{-7} injects very few errors and errors are rarely being read in. At this error rate, only single-bit errors can happen to a register and AVF measurement itself can be a very good tool for analyzing the reliability of the IRD scheme. To exercise the IRD register file's resilience to double- or multi-bit errors as in extremely harsh environments, a higher error rate is required for the injection. For illustration purposes, only results for error rates of 10^{-5} (shown as "e-5") and 10^{-4} ("e-4") are presented. Notice that these again are accelerated rates for very rare single-event upsets (SEUs).

Erroneous input operands can be either read from the register file or retrieved from

the bypass network. This experiment tries to identify the contributions of these two error sources. Notice that erroneous reads are instances of retrieved input operands with errors, which are different from the cumulative bit errors in the input operands. For example, an input value with multiple bits flipped due to soft errors (multiple-bit errors) is only counted as one instance of erroneous read. Table 6.1 shows that the majority of the erroneous reads, for both soft error rates of e-5 and e-4, 1) around 89% on the average, are due to the corrupted value read from the register file (RF), 2) the remaining 11% are due to wire flips when the value is being forwarded by the bypass network (FWD). Thus, the register file is still the major source of erroneous reads. A second breakdown of erroneous reads in Table 6.1 shows that most readin errors are single-bit errors (SE), 99.8% (99.4%) at this very high error rate e-5 (e-4). This is because the live time (between writeback and the last read) of a register value is quite short [88], during which the same register entry is rarely hit by multiple errors.

Table 6.1 A characterization of erroneous reads for input operands.

	Error Sources				Error Types			
	e-5		e-4		e-5		e-4	
	RF	FWD	RF	FWD	SE	ME	SE	ME
gzip	86.3%	13.7%	83.8%	16.2%	100.0%	0.0%	98.9%	1.1%
vpr	97.1%	2.9%	95.2%	4.8%	100.0%	0.0%	100.0%	0.0%
gcc	78.1%	21.9%	89.8%	10.2%	100.0%	0.0%	100.0%	0.0%
mcf	93.0%	7.0%	96.8%	3.2%	99.1%	0.9%	97.6%	2.4%
crafty	90.6%	9.4%	86.6%	13.4%	100.0%	0.0%	99.7%	0.3%
parser	91.3%	8.7%	90.3%	9.7%	98.6%	1.4%	98.2%	1.8%
eon	90.3%	9.7%	86.9%	13.1%	100.0%	0.0%	100.0%	0.0%
perlbmk	87.8%	12.2%	87.5%	12.5%	100.0%	0.0%	100.0%	0.0%
gap	91.5%	8.5%	92.0%	8.0%	100.0%	0.0%	100.0%	0.0%
vortex	86.4%	13.6%	86.4%	13.6%	100.0%	0.0%	100.0%	0.0%
bzip2	85.7%	14.3%	83.4%	16.6%	100.0%	0.0%	98.4%	1.6%
twolf	96.1%	3.9%	93.0%	7.0%	100.0%	0.0%	100.0%	0.0%
Avg	89.5%	10.5%	89.3%	10.7%	99.8%	0.2%	99.4%	0.6%

*RF: Register File, FWD: Forward Network, SE: Sing-bit Error, ME: Multi-bit Error

6.6.6 Error Detection and Recovery from Detected Soft Errors

Integrated with parity coding, IRD checks the parity bits for both 32-bit halves at the first stage of execution. If any half fails this check, erroneous data is detected. This scheme covers both narrow-width values and regular values. However, this parity coding scheme is not capable of detecting an even number of bit errors in a 32-bit half.

Figure 6.16 presents results for IRD using parity checking. `P_H_Detected` and `P_F_Detected` correspond to detected readin errors in narrow-width values and regular values, and `P_H_Fail` and `P_F_Fail` represent undetected readin errors, respectively. IRD using parity checking detects all readin errors in regular values and only fails less than 0.3% of the time for narrow-width values.

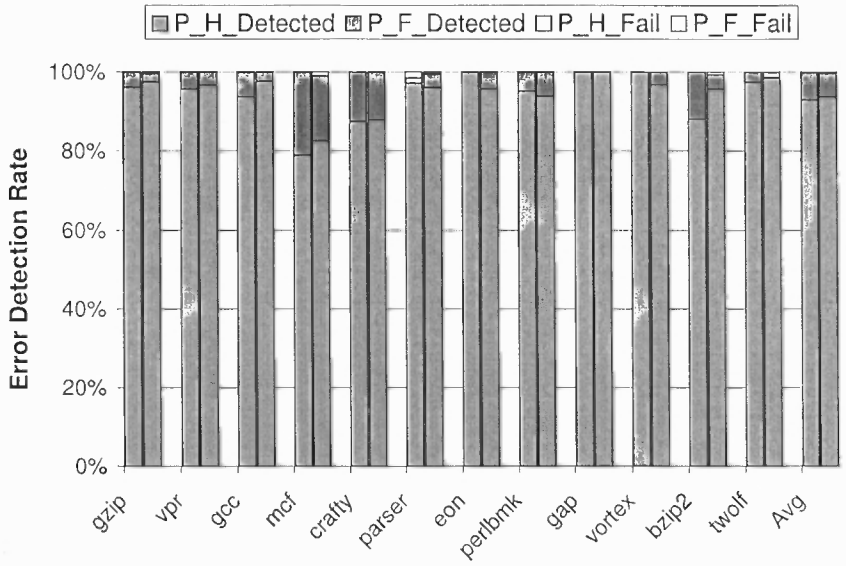


Figure 6.16 Soft error detection in the IRD scheme by parity checking. (Left bar for e-5 and right bar for e-4.)

Notice that the in-register duplication scheme restores the full 64-bit value of a narrow-width input by only using its lower 32-bit half. This is to say that, for narrow-width values the IRD scheme is further tuned to use the parity checking result of the lower 32-bit half to detect soft errors and the parity checking of the upper half to determine whether

it can be used to recover the value once the lower half is detected as error-corrupted. Of these readin erroneous narrow-width values, IRD detects 99.7% of the errors, which is very encouraging. Once errors are detected, IRD makes the following decision: if the duplicate in the upper 32-bit half passes the parity check, IRD uses the duplicate for error recovery; otherwise, IRD generates an ERROR exception and lets the operating system handle error recovery. This work introduces an additional 1000 cycles for this ERROR exception handler. Notice that each detected erroneous regular value will also trigger this ERROR exception. However, during IRD recovery, if the duplicate was also corrupted but yet succeeded in parity checking (even number of bit errors), IRD is forced to perform a false recovery using the corrupted duplicate. Figure 6.17 shows, that, of the detected errors in narrow-width input operands, IRD recovers 99.7% (99.2%) of the errors with non-corrupted duplicates, `IRD_True_Recovery`. The false recovery rate, `IRD_False_Recovery`, is 0% (0.1%) at error rates e-5 (e-4). The operating system takes care of the remaining 0.3% (0.7%) of the detected errors. A performance comparison was shown early in Figure 6.12. The performance overhead due to error recovery is negligible at these two error rates.

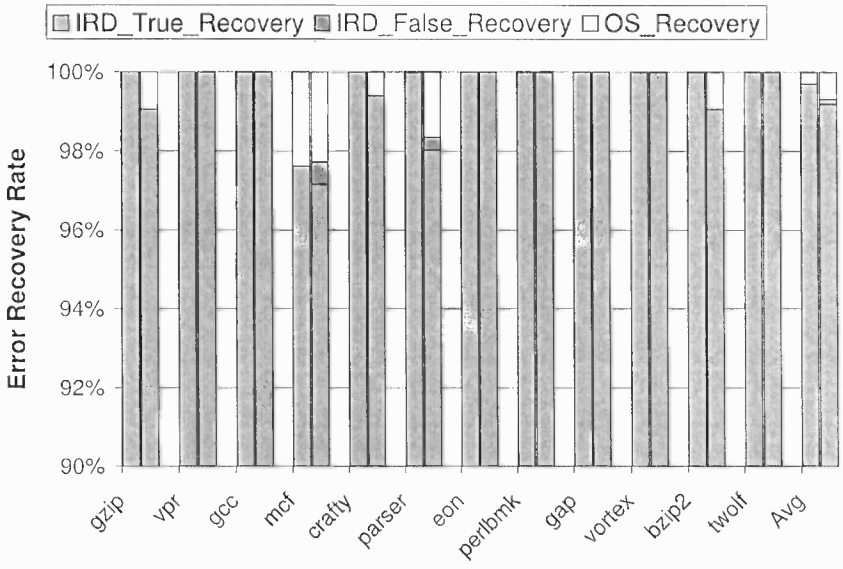


Figure 6.17 Error recovery rate of detected errors in IRD scheme, under error injection rates of 10^{-5} (left bar) and 10^{-4} (right bar) per selected bit per cycle.

Overall, these results confirm that the in-register duplication scheme that exploits

narrow-width values is very effective in detecting and recovering from soft errors occurring in the register file, the bypass network, or the result writeback bus, while only incurring some minor microarchitectural modifications. It is important to notice that this high error detection/recovery rate in the IRD register file is achieved under the extremely high error injection rates that are unlikely to happen in the real world. Thus, for realistic applications experiencing significantly low error incidents, it is of crucial importance that reliable designs only incur minimal cost/overhead in terms of hardware, performance, and power consumption.

6.7 Summary

To design high-performance reliable register files, an in-register duplication (IRD) scheme is proposed in this work by exploiting the narrow-with values. In IRD, the narrow-width register value is duplicated in its upper 32-bit half, which can eliminate the hardware complexity required for acquiring and maintaining copy registers in previous schemes. A new AVF measurement of the register file is studied and the experimental results show that the IRD scheme achieves an extremely low AVF of 0.1% in the register file, a 98.8% reduction over a base one. Software-based error injection evaluation in this work also shows that the IRD scheme demonstrates superior error detection and recovery rates at minimum hardware cost.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

With continuous technology scaling down, the reliability issue becomes of increasing concern and one of the major constraints in microprocessor design. This work focuses on the reliability design of the on-chip memory structures, i.e., L1 data/instruction caches and register files, against soft errors.

Due to their large share of the transistor budget and die area, on-chip caches suffer from a high soft error rate . To improve the reliability of on-chip caches, this work first performs a detailed study on the cache vulnerability to soft errors based on new lifetime models for data and tag arrays; both data and instruction caches are studied. It aims to provide insights into cache vulnerability behavior as well as guidance in designing cost-effective highly reliable caches. First, this work studies the impact of different data cache write policies, early write back schemes and the proposed multiple-dirty-bit (MDB) scheme on reducing the vulnerable `WPL` phase of dirty cachelines. This work proposes a clean cacheline invalidation (CCI) scheme to reduce the time that clean cachelines stay in the vulnerable `RR` phase; it also studies the narrow-width value compression (NWVC) scheme toward reducing the overall vulnerable phases. By combining the DTEWB, MDB, CCI, and NWVC schemes, the data array in the data cache attains a substantially improved reliability. For the data array in an instruction cache, this work proposes a variation of the cacheline scrubbing (CS) scheme to reduce the vulnerable phase. Combined with the CCI scheme, the CS-CCI scheme achieves a lower TVF with minimum performance and energy overheads. This work also develops a new lifetime model for the tag array based on an extended Hamming-distance-one (HDO) analysis. The results with HDO analysis indicate that the tag array has a potentially low TVF, except for the writeback data cache,

and the DTEWB and CCI schemes can substantially improve the reliability of the tag arrays in the cache.

In Chapter 3, a lifetime vulnerability model for the tag array has been proposed and studied. However, only those schemes, such as DTEWB and CCI, which targeted at improving the reliability of the data array, have been evaluated for their benefit on improving tag array reliability. Exploiting the address locality of memory accesses, a Tag Replication Buffer (TRB) is proposed to protect information integrity of the tag array in the data cache with low performance, energy and area overheads. Several optimized schemes, including a selective TRB scheme that protects only the tag entries of dirty cachelines, are subsequently proposed. In order to provide a comprehensive evaluation on the reliability of the cache tag array, the dissertation conducts a cache tag vulnerability factor analysis and proposes a refined cache tag reliability evaluation metric DOR-TVF that combines the TVF and AWR analysis. Based on the DOR-TVF analysis, a new TRB scheme with early write-back (TRB-EWB) triggered by the tag buffer replacement is proposed, which can achieve a 100% AWR rate and a zero DOR-TVF with a minimum performance and energy overhead.

All the reliable designs proposed in Chapters 3 and 4 target fixed operating environments, i.e., characterized by fixed soft error rates. To design reliable systems in changing operating environments, this work proposes a new methodology that chooses the applied reliability scheme for the best match with the current erroneous situation. This methodology is exemplified by presenting the design of a self-adaptive reliable data cache (SA-RDC) that supports three levels of protection targeting at different error rates with different performance and energy impacts. The monitoring component of the adaptive design continuously monitors the error incidents within preset time windows and sends the error information to the control component. The latter decides whether to replace the current reliability scheme or not. The simulation results show that this self-adaptive reliable data cache scheme can effectively take the best advantage of each provided meta scheme while avoiding their de-

ficiencies.

Besides on-chip caches, the register file is another on-chip memory structure that is of critical importance in high-performance reliable microprocessor design. To improve the reliability of the register file, this work proposes to exploit narrow-width register values. Instead of allocating an additional copy register for storing a duplicate; the in-register duplication (IRD) scheme creates a replica of the narrow-width value in the upper 32-bit half, thus eliminating the hardware complexity required for acquiring and maintaining copy registers in previous schemes. AVF measurement based on a new analysis model has shown that the IRD scheme achieves an extremely low AVF compared to the basic one. Evaluation via software-based error injection shows that the IRD scheme demonstrates superior error detection and recovery rates at minimum hardware cost, making it a suitable design choice in high-performance, highly reliable microprocessors.

Enhanced by the proposed schemes in this work, the reliability of the data/instruction caches and register file, as well as of the entire microprocessor, will be dramatically improved. The new lifetime vulnerability model of the data/instruction caches can provide some guidance toward future reliable cache designs. The limited study of the self-adaptive data cache conveys a very encouraging message in the area of self-adaptive reliable system design, and the IRD scheme also demonstrates the possibility of exploiting the narrow-with values for the reliable design of other components in microprocessors.

7.2 Future Work

This dissertation has mainly focused on reliable on-chip memory structures design on the single-core superscalar processors. As computer architecture enters the era of multi-/many-core, improving the reliability of the multi-/many-core systems will become the major challenge in designing next generation microprocessors. Therefore, in the near future, I would like to extend the current reliable system design to the multithreading, such as simultaneously multithreading (SMT), and the multi-/many-core processors.

The similar cache vulnerability analysis presented in this dissertation can be adopted on the caches in the multiprocessor, where cache coherence protocols will further complicate this analysis. Due to the different access patterns of these caches, different vulnerable phases can be identified and the reliability optimizing schemes for multiprocessor caches can be proposed based on the new lifetime vulnerability analysis. In the multiprocessor scenario, the multiple copies of the shared data in private L1 caches, which also is a form of information redundancy, can provide the ability of error recovery. My goal for this future research is to build a reliability measurement model for the caches in the multiprocessors and finally propose a reliable cache coherence protocol that can improve the reliability of the multi-/many-core systems.

Further, the narrow-width value can be exploited to improve the reliability, performance, power efficient and in the multithreading and multiprocessor architectures, as well as the Network-on-Chip (NoC) system. By identifying the major narrow-width values, the reliability of the communication between different cores can be enhanced by duplicating the narrow-width value within itself. The performance and power efficient can also be improved by saving of the bits that need to be transferred for the narrow-width values. Especially for the NoC system, in which the interconnection is among the major design issues as the number of cores is increasing in deep sub-micron, the reliability, performance, and power can be substantially improved by exploiting the narrow-width value designs.

REFERENCES

[1] J. F. Ziegler *et al.*, "IBM experiments in soft fails in computer electronics (1978 - 1994)," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 3–18, January 1996.

[2] R. Blish *et al.*, "Critical reliability challenges for the international technology roadmap for semiconductors (itrs)," *Technical Report, International SEMATECH*, March 2003.

[3] T. J. O'Gorman *et al.*, "Field testing for cosmic ray soft errors in semiconductor memories," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 41–50, January 1996.

[4] P. Shivakumar *et al.*, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. of the International Conference on Dependable Systems and Networks*, June 2002, pp. 389–398.

[5] P. Hazucha and C. 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, December 2000.

[6] S. Hareland *et al.*, "Impact of cmos process scaling and soi on the soft error rates of logic processes," in *Proc. of the Symposium on VLSI Technology Digest of Technical Papers*, 2001, pp. 73–74.

[7] K. X. Zhang, "Soft error immunity in cmos circuits with large shared diffusion areas," US Patent #6087849.

[8] S. Kirkpatrick, "Modeling diffusion and collection of charge from ionizing radiation in silicon devices," *IEEE Trans. Electron Devices*, vol. 26, no. 11, pp. 1742–1753, November 1979.

[9] T. Karnik *et al.*, "Impact of body bias on alpha- and neutron-induced soft error rates of flip-flops," in *Proc. of the Symposium on VLSI Circuits*, 17-19 June 2004, pp. 324–325.

[10] Y. S. Dhillon *et al.*, "Sizing cmos circuits for increased transient error tolerance," in *Proc. of the 10th IEEE International On-Line Testing Symposium (IOLTS'04)*, 2004, pp. 11–16.

[11] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 2007.

[12] K. C. Yager, "The MIPS R10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–40, April 1996.

[13] R. E. Lyons and W. Vanderkulk, "The use of tripple-modular redundancy to improve computer reliability," *IBM Journal*, April 1962.

[14] "Hp nonstop himalaya. http://nonstop.compaq.com/."

[15] T. J. Slegel *et al.*, "IBM's S/390 G5 microprocessor design," *IEEE Micro*, vol. 19, no. 2, pp. 12–23, March/April 1999.

[16] M. Namjoo and E. McCluskey, "Watchdog processors and detection of malfunctions at the system level," CRC, Tech. Rep. 81-17, December 1981.

[17] T. Austin, "Diva: A reliable substrate for deep submicron microarchitecture design," in *Proc. of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture*, November 1999, pp. 196–207.

[18] E. Rotenberg, "Ar-smt: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. of the International Symposium on Fault-Tolerant Computing*, June 1999, pp. 84–91.

[19] S. Reinhardt and S.Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proc. of the 27th Annual International Symposium on Computer Architecture*, June 2000, pp. 25–36.

[20] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multithreading alternatives," in *Proc. of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 99–110.

[21] T. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery via simultaneous multithreading," in *Proc. of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 87–98.

[22] K. Sundaramoorthy, Z. Purser, and E. Rotenburg, "Slipstream processors: Improving both performance and fault tolerance," in *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 257–268.

[23] A. Mendelson and N. Suri, "Designing high-performance and reliable superscalar architectures: The out of order reliable superscalar (o3rs) approach," in *Proc. of the International Conference on Dependable Systems and Networks*, June 2000.

[24] J. Ray, J. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," in *Proc. of the 34th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2001, pp. 214–224.

[25] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam, "A complexity-effective approach to alu bandwidth enhancement for instruction-level temporal redundancy," in *Proc. of the 31st Annual International Symposium on Computer Architecture*, June 2004.

[26] J. Smolens, J. Kim, J. C. Hoe, and B. Falsafi, "Efficient resource sharing in concurrent error detecting superscalar microarchitecture," in *Proc. of the ACM/IEEE International Symposium on Microarchitecture*, December 2004.

[27] M. Gomaa and T. N. Vijaykumar, "Opportunistic transient-fault detection," in *Proc. of the 32nd Annual International Symposium on Computer Architecture*, June 2005.

[28] S. Kim and A. Somani., "Area efficient architectures for information integrity checking in cache memories," in *Proc. of International Symposium on Computer Architecture*, May 1999, pp. 246–255.

[29] R. Phelan, "Addressing soft errors in ARM core-based soc," ARM White Paper, ARM Ltd., Dec 2003.

[30] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam, "Icr: in-cache replication for enhancing data cache reliability," in *Proc. of the International Conference on Dependable Systems and Networks*, 2003.

[31] J.-C. Lo, "Fault-tolerant content addressable memory," in *Proc. of the International Conference on Computer Design*, 1993, pp. 193–196.

[32] F. Salice, M. Sami, and R. Stefanelli, "Fault-tolerant cam architectures: A design framework," in *Proc. of the 17th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, 2002, pp. 233–244.

[33] A. Biswas *et al.*, "Computing architectural vulnerability factors for address-based structures," in *Proc. of the IEEE International Symposium on Computer Architecture*, June 2005.

[34] H. Wang, T. Sun, and Q. Yang, "Cat—caching address tags: a technique for reducing area cost of on-chip caches," in *Proc. of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 381–390.

[35] G. Memik *et al.*, "Increasing register file immunity to transient errors," in *Proc. of Design, Automation, and Test in Europe*, Munich, Germany, May 2005.

[36] G. Memik, M. H. Chowdhury, A. Mallik, and Y. I. Ismail, "Engineering over-clocking: Reliability-performance trade-offs for high-performance register files," in *Proc. of the International Conference on Dependable Systems and Networks*, 2005, pp. 770–779.

[37] J. Yan and W. Zhang, "Compiler-guided register reliability improvement against soft errors," in *Proc. of the 5th ACM International Conference On Embedded Software*, 2005, pp. 203–209.

[38] M. Kandala, W. Zhang, and L. T. Yang, "An area-efficient approach to improving register file reliability against transient errors," in *Proc. of the 21st International Conference on Advanced Information Networking and Applications Workshops*, 2007, pp. 798–803.

[39] P. Montesinos, W. Liu, and J. Torrellas, "Using register lifetime predictions to protect register files against soft errors," in *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007, pp. 286–296.

[40] L. Li *et al.*, "Soft error and energy consumption interactions: A data cache perspective," in *Proc. of the International Symposium on Low Power Electronics and Design*, 2004, pp. 132–137.

[41] W. Zhang, "Computing cache vulnerability to transient errors and its implication," in *Proc. of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 2005.

[42] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. of the 5th International Symposium on High Performance Computer Architecture*, January 1999.

[43] O. Ergin *et al.*, "Register packing: Exploiting narrow-width operands for reducing register file pressure," in *Proc. of the 37th Annual International Symposium on Microarchitecture*, Portland, OR, 2004, pp. 304–315.

[44] D. Burger, A. Kagi, and M. S. Hrishikesh, "Memory hierarchy extensions to simplescalar 3.0," Department of Computer Sciences, The University of Texas at Austin, Tech. Rep. TR99-25, 2000.

[45] P. Bannon, "Alpha 21364: A scalable single-chip smp," *Microprocessor Forum*, Oct. 1998.

[46] P. Shivakumar and N. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," Compaq Western Research Lab, Tech. Rep., 2001.

[47] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. of the International Symposium on High-Performance Computer Architecture*, 2000.

[48] R. P. Preston *et al.*, "Design of an 8-issue superscalar risc microprocessor with simultaneous multithreading," in *Proc. of the IEEE International Solid-State Circuits Conference*, 2002.

[49] "Spec cpu2000 v1.3," http://www.spec.org/cpu2000/.

[50] T. Sherwood *et al.*, "Automatically characterizing large scale program behavior," in *Proc. of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[51] H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Vulnerability analysis of L2 cache elements to single event upsets," in *Proc. of Design, Automation, and Test in Europe*, March 2006.

[52] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proc. of the 40th IEEE/ACM International Symposium on Microarchitecture*, Dec 2007, pp. 197–209.

[53] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, "Soft errors issues in low-power caches," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, no. 10, pp. 1157–1166, Oct. 2005.

[54] N. N. Sadler and D. J. Sorin, "Choosing an error protection scheme for a microprocessor L1 data cache," in *Proc. of the International Conference on Computer Design*, Oct. 2006.

[55] V. Sridharan, H. Asadi, M. B. Tahoori, and D. Kaeli, "Reducing data cache susceptibility to soft errors," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, 2006.

[56] S. Wang, J. Hu, and S. G. Ziavras, "Self-adaptive data caches for soft-error reliability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1503–1507, August 2008.

[57] J. S. Hu, G. M. Link, J. K. John, S. Wang, and S. G. Ziavras, "Resource-driven optimizations for transient-fault detecting superscalar microarchitectures," in *Proc. of Tenth Asia-Pacific Computer Systems Architecture Conference*, Singapore, October 24-26 2005.

[58] S. Wang, J. Hu, and S. G. Ziavras, "On the characterization of data cache vulnerability in high-performance embedded microprocessors," in *Proc. of the 6th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, Samos, Greece, July 17-20 2006, pp. 14 – 20.

[59] G. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Balancing reliability and performance in the memory hierarchy," in *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, March 2005.

[60] J. Yan and W. Zhang, "Evaluating instruction cache vulnerability to transient errors," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 4, pp. 21–28, Sept. 2007.

[61] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. of the International Symposium on Computer Architecture*, 2001.

[62] G. H. Loh, "Exploiting data-width locality to increase superscalar execution bandwidth," in *Proc. of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, 2002.

[63] M. H. Lipasti *et al.*, "Physical register inlining," in *Proc. of the 31st Annual International Symposium on Computer Architecture*, June 2004, pp. 325–335.

[64] S. Wang, H. Yang, J. Hu, and S. G. Ziavras, "Asymmetrically banked value-aware register files," in *Proc. of the IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 363–368.

[65] J. Hu, S. Wang, and S. G. Ziavras, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *Proc. of the International Conference on Dependable Systems and Networks*, Philadelphia, PA, June 25-28 2006, pp. 281–290.

[66] O. Ergin, O. Unsal, X. Vera, and A. Gonzalez, "Exploiting narrow values for soft error tolerance," *IEEE Computer Architecture Letters*, vol. 5, 2007.

[67] J. Hu, S. Wang, and S. G. Ziavras, "On the exploitation of narrow-width values for improving register file reliability," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 7, pp. 953–963, July 2009.

[68] A. Aggarwal and M. Franklin, "Energy efficient asymmetrically ported register files," in *Proc. of the IEEE International Conference on Computer Design*, 2003, pp. 2–7.

[69] M. Kondo and H. Nakamura, "A small, fast and low-power register file by bit-partitioning," in *Proc. of the 11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 40–49.

[70] S. Wang, H. Yang, J. Hu, and S. G. Ziavras, "Asymmetrically banked value-aware register files for low energy and high performance," *Microprocessors and Microsystems*, vol. 32, no. 3, pp. 171–182, May 2008.

[71] O. Ergin, "Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors," in *Proc. of the 16th International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2006, pp. 477–485.

[72] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2003.

[73] P. Pujara and A. Aggarwal, "Increasing the cache efficiency by eliminating noise," in *Proc. of the 12th International Symposium on High-Performance Computer Architecture*, Feb 2006.

[74] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *Proc. of the 33th Annual International Symposium on Microarchitecture*, 2000, pp. 214–220.

[75] N. S. Kim, T. Austin, and T. Mudge, "Low-energy data cache using sign compression and cache line bisection," in *Proc. of the Workshop on Memory Performance Issues*, 2002.

[76] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt, "Cache scrubbing in microprocessors: Myth or necessity," in *Proc. of 10th International Symposium on Pacific Rim Dependable Computing, March 3-5, 2004*, March 3-5 2004.

[77] A. M. Saleh, J. J. Serrano, and J. H. Patel, "Reliability of scrubbing recovery-techniques for memory systems," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 114–122, April 1990.

[78] S. Wang, J. Hu, and S. G. Ziavras, "On the characterization and optimization of on-chip cache reliability against soft errors," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1171–1184, September 2009.

[79] W. Zhang, "Enhancing data cache reliability by the addition of a small fully-associative replication cache," in *Proc. of the International Conference on Supercomputing*, 2004, pp. 12–19.

[80] N. Quach, "High availability and reliability in the itanium processor," *Micro, IEEE*, vol. 20, no. 5, pp. 61–69, Sept.-Oct. 2000.

[81] K. Reick *et al.*, "Fault-tolerant design of the ibm power6 microprocessor," *IEEE Micro*, vol. 278, no. 2, pp. 30–38, March-April 2008.

[82] P. Petrov and A. Orailoglu, "Tag compression for low power in dynamically customizable embedded processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1031–1047, July 2004.

[83] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "Cacti 4.0," HP Laboratories Palo Alto, Tech. Rep., 2006.

[84] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. of the International Symposium on Low Power Electronics and Design*, June 1998, pp. 76–81.

[85] N. J. Wang and S. J. Patel, "Restore: Symptom-based soft error detection in microprocessors," *IEEE Transactions on Dependable and Secure Computing*, July-September 2006.

[86] C. Weaver *et al.*, "Techniques to reduce the soft errors rate in a high-performance microprocessor," in *Proc. of the 31st Annual International Symposium on Computer Architecture*, 2004.

[87] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Adaptive error protection for energy efficiency," in *Proc. of the International Conference on Computer Aided Design*, November 2003, pp. 2–7.

[88] G. S. S. J. Adam Butts, "Use-based register caching with decoupled indexing," in *Proc. of 31st Annual International Symposium on Computer Architecture*, 2004, pp. 302–313.

[89] D. R. Lutz and D. N. Jayasimha, "Early zero detection," in *Proc. of the 1996 International Conference on Computer Design*, 1996, pp. 545–550.

[90] M. S. Hrishikesh, D. Burger, S. W. Keckler, P. Shivakumar, N. P. Jouppi, and K. I. Farkas, "The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays," in *Proc. of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[91] E. Borch *et al.*, "Loose loops sink chips," in *Proc. of the 8th Annual International Symposium on High-Performance Computer Architecture*, February 2002, pp. 270–281.

[92] J. A. Butts and G. Sohi, "Dynamic dead-instruction detection and elimination," in *Proc. of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002, pp. 199–210.