Overcoming Hard-Faults in High-Performance Microprocessors

by

Amin Ansari

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2011

Doctoral Committee:

Associate Professor Scott Mahlke, Chair
Professor Todd M. Austin
Assistant Professor Thomas F. Wenisch
Assistant Professor Zhengya Zhang

To my family

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my advisor, Professor Scott Mahlke, for his support and mentorship in these past years. An enthusiastic researcher and a constant source of ideas, Scott was a great advisor. I also owe thanks to the remaining members of my dissertation committee, Professor Austin, Professor Wenisch, and Professor Zhang. They all donated their time to help shape this research into what it has become today.

I am also indebted to my reliability colleagues, Shantanu Gupta and Shuguang Feng. It has been a pleasure working with them, and I cannot imagine having this thesis in its current form without their support. Shantanu spent a lot of time discussing research ideas with me and helping me to gain a better grasp of research fundamentals. Shuguang has helped with me throughout this process, sitting through our long meetings with Scott, giving excellent research insights, and always being there for any help with writing and proof reading papers.

During my stay in Michigan, I was lucky to work with an amazing set of people in our research lab, Compilers Creating Custom Processors (CCCP). I would like to thank Gaurav Chadha, Hyoun Kyu Cho, Kevin Fan, Shuguang Feng, Shantanu Gupta, Jeff Hao, Amir Hormati, Po-Chun Hsu, Anousheh Jamshidi, Manjunath Kudlur, Yuan Lin, Andrew Lukefahr, Mojtaba Mehrara, Hyunchul Park, Yongjun Park, Mehrzad Samadi, Ankit Sethia,

Mark Woh, and Griffin Wright. You folks made coming to the office more fun, and I would have not made it through without you. I have also learned a great deal about different cultures, beliefs, and cuisines through our countless discussions in the lab. I also want to thank my world-class ping pong buddy, Gaurav Chadha for late-night, smash-intensive games.

Most importantly, my family deserves major gratitude. My parents and my sister provided their unconditional love and support. I feel blessed to have a family who has given me all the opportunities to succeed in life. My dad's dedication and academic excellence have been a constant source of inspiration to me throughout my whole life. His endless knowledge and courageous grand visions still amaze me. My uncles and his wife have made my occasional California vacations truly memorable. Finally, the greatest of thanks go to my cousins, uncles, aunts, and grandparents who enriched moments of my life.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

LIST OF TABLES xiv

ABSTRACT xv

CHAPTER

I. Introduction 1

1.1 Reliability Threats in Deep Submicron Technologies 1
1.1.1 Process Variation 3
1.1.2 Manufacturing defects 3
1.1.3 Wearout 4
1.1.4 Power Consumption 5
1.2 Overcoming Hard-Faults in High-Performance Microprocessors . . 6
1.2.1 Challenges with High-Performance Microprocessors . . 7
1.2.2 Protecting On-Chip Caches 9
1.2.3 Protecting Non-Cache Parts of the Core 11
1.3 Contributions 13
1.4 Organization 15

II. Related Work 16

2.1 Fault-Tolerant Cache Techniques 16
2.1.1 Coding Solutions 16
2.1.2 Circuit-Level and VLSI Solutions 17
2.1.3 Architectural Solutions 18
2.2 Low-Power Cache Techniques 19
2.2.1 Conventional Low-Power Cache Techniques 19

2.2.2 Lowering Power by Tolerating Failures in On-Chip Caches 19
2.2.3 Alternative SRAM Cells 20
2.3 Handling Hard-Faults in the Non-Cache Parts of the Core 21
2.3.1 Coarse-Grained Redundancy and Disabling 21
2.3.2 Fine-Grained Redundancy and Disabling 21
2.3.3 Unconventional Approaches 22

III. Armoring Cache Architectures in High Defect Density Technologies . . 24

3.1 Introduction 24
3.2 ZerehCache 28
3.2.1 ZC Architecture 28
3.2.2 Hard-Fault Detection 35
3.2.3 ZC Configuration 36
3.3 Design Space Exploration 41
3.4 Yield Analysis 49
3.5 Wearout Tolerance 52
3.6 Comparison and Discussion 55
3.6.1 Comparison with Conventional Techniques 55
3.6.2 Comparison with Recently Proposed Techniques 57
3.6.3 Significance 59
3.7 Summary 60

IV. A Polymorphic Cache Design for Enabling Robust Near-Threshold Operation 62

4.1 Introduction 62
4.2 Archipelago 67
4.2.1 Baseline AP Architecture 67
4.2.2 AP with Relaxed Group Formation 71
4.2.3 AP Configuration 75
4.3 Evaluation 81
4.3.1 Methodology 81
4.3.2 Design Space Exploration 83
4.3.3 Results 86
4.4 Quantitative Comparison to Alternative Methods 90
4.5 Summary 92

V. Enhancing System Throughput by Animating Dead Cores 93

5.1 Introduction 93
5.2 Utility of an Undead Core 95
5.2.1 Effect of Hard-Faults on Program Execution 95
5.2.2 Relaxing Correctness Constraints 97
5.2.3 Opportunities for Acceleration 98

5.3 From Traditional Coupling to Animation 100
5.4 NM Architecture 102
5.4.1 High-Level NM System Description 103
5.4.2 Hint Gathering and Distribution 106
5.4.3 Reducing Communication Overheads 109
5.4.4 Hint Disabling Mechanisms 110
5.4.5 Resynchronization 113
5.4.6 NM Design for CMP Systems 114
5.5 Evaluation 116
5.5.1 Experimental Methodology 116
5.5.2 Experimental Results 119
5.6 Throughput Enhancement 126
5.7 Summary 129

VI. Conclusions 130

BIBLIOGRAPHY 134

LIST OF FIGURES

Figure

3.1 Probability of having at least one faulty SRAM cell at different granularities while varying the failure probability of each SRAM cell, P_F 25

3.2 Fraction of non-functional SRAM bit-cells for a 2MB L2 cache over time. Here, the mean time to failure of each SRAM bit-cell is varied from 50 to 200 years. 27

3.3 Two simple scenarios in which the line swapping can preserve the correct functionality of the cache by resolving the occurred collision. A black box shows a faulty chunk of data. 29

3.4 The high-level architecture of the ZC is shown in this figure and the extra modules that are added to the baseline cache are highlighted. Note that the slices of the base address are shown using numbers 1, 2 and 3 (Address Format). The fault map array and spare cache have their own shared decoder to avoid getting their word-line activation signals from the main cache's decoder. For simplicity, the separate sense amps for the fault map and spare cache are not shown. Built-in-self-test (BIST) module is commonly used for fault diagnosis in the embedded memory structures. . 30

3.5 A Benes network is shown which connects the second rows of the four consecutive logical group of rows in the main cache. As an example, a single route from the decoder to the word-lines is also shown. 32

3.6 Mapping between the graph coloring problem and the defect pattern in the main/spare caches. The solid edges stand for the intrinsic conflicts between the word-lines. The dotted edges correspond to the word-line conflicts due to the defect pattern. An "X" indicates a collision using a default grouping. Numbers written in the fault map indicate the corresponding cache word-lines to which the spare units are assigned. (G=Green, B=Blue, P=Purple, O=Orange) 37

3.7 Proper configuration of two BNs that transform the actual cache layout (left) to the virtual one (right) for the given coloring assignment. The upper (lower) BN connects the first (second) rows of the 4 logical groups. The darker 2-input MUXes are configured to output their lower input while the lighter MUXes output their upper input. (G=Green, B=Blue, P=Purple, O=Orange) 40

3.8 The run-time of the IBSC graph coloring solver in *ms* for different edge densities and number of nodes in the graph. In this figure, *p* is the edge density which is defined as the probability of having an edge between an arbitrary pair of nodes in a random graph *G(n,p)*. 43

3.9 P_{op} of L2 ZC for different P_F while fixing two parameters and allowing the third one to vary. 46

3.10 Area, power, and energy overhead of the potential L1/L2 ZCs which are stated in percentage. 48

3.11 Distribution of generated chips by the number of faulty SRAM cells in their L1/L2 caches. A population of 1000 chips is generated by considering the large-area clustering effect, intra-die, inter-die, systematic, and parametric variations. 51

3.12 Results of Monte Carlo lifetime simulation which show the probability of operation for L1/L2 caches protected by different mechanisms. In addition, the shaded region shows the expected number of failures over the life-time. 54

3.13 Area overhead of the different protection mechanisms for tolerating a given P_F. In this figure, Row-Redun stands for the row redundancy protection scheme. ECC and ECC-2 are the 1-bit and 2-bit error correction schemes, respectively. 56

4.1 Bit error rate for an SRAM cell with varying V_{dd} values in $90nm$. For this technology, the write-margin is the dominant factor and limits the operational voltage of the SRAM structure. Here, the Y-axis is logarithmic, highlighting the extremely fast growth in failure rate with decreasing V_{dd}. The two horizontal dotted-lines mark the failure rates at which the mentioned SRAM structures (64KB and 2MB) can operate with at least **99%** manufacturing yield. 63

4.2 Percentage of faulty bits, bytes, words, blocks, columns, and word-lines for a 2MB L2 cache while varying the supply voltage. Here, the Y-axis is logarithmic, highlighting the rapid growth in faulty units when decreasing V_{dd}. The top part of this figure depicts our conceptual division of this V_{dd} range into four different regions based on the protection difficulty. For each region, corresponding bit error rates and also several applicable protection techniques are also shown. In order to operate correctly in the failure-free region, no protection mechanism is required. However, as can be seen, our target is the high failure rate region which causes an avalanche of failures for on-chip caches. 65

4.3 Basic structure of a dual-bank 2-way set-associative Archipelago. Two cache banks with eight lines each are shown. Each block consists of 3 equally sized data chunks. Black boxes in each cache line represent chunks of data that have at least one faulty bit. The memory and fault maps, which are essential components of the proposed scheme, are also shown. 69

4.4 Two special read-access scenarios. A standard read access is illustrated in Figure 4.3. Notice the extra bit that has been added to both the memory map and every fault map entry to handle scenario (b). Since the 4*th* data chunk of semi-sacrificial line is re-allocated, it is marked as **RA** in scenario (b). 72

4.5 A simplified example of the minimum clique covering process for a given distribution of faults in the cache banks. Here, each bank has only 5 lines. The solver disables the 6*th* line since it has many faulty chunks and, is therefore very expensive to repair. Two cliques are formed by the solver and lines 9 and 3 are designated as sacrificial lines for groups 1 and 2, respectively. Moreover, the conceptual partitioning of the cache to distinct islands is also demonstrated. 74

4.6 Distribution of the clique size for different versions of the solver based on Monte Carlo simulation. Note that for 64-cap, the size of all cliques is ≤ 64. Here, the number of non-functional lines is the summation of the number of sacrificial lines and the number of disabled lines. The plot in the insert depicts the average number of non-functional cache lines, the maximum number of non-functional lines, and the number of disabled lines while achieving 99% yield. 78

4.7 Process of determining the minimum achievable V_{dd} for L1 and L2 caches while limiting the fraction of the non-functional cache lines and also the area overhead of the fault map structure to $\leq 10\%$. Moreover, in these 10 sub-plots, vertical dotted lines show the minimum achievable V_{dd} while data chunk size varies from 1 *bit* to 16 *bits*. 84

4.8 Design points for different Maximum Clique Size (MCS) and chunk size pairs are shown that can achieve a 99% yield. For each MCS value, corresponding chunk sizes from $\{2^n \mid n \in \{0, 1, ..., 7\}\}$ for L1 and from $\{2^n \mid n \in \{0, 1, ..., 5\}\}$ for L2 are chosen. The shaded boxes represents the region of interest where both the fault-map overhead and the fraction of non-functional lines is limited to $\leq 10\%$. The black dotted line is the Pareto frontier. 85

4.9 Area, leakage, and dynamic power overheads of our scheme for both L1 and L2 caches. Here, 10T cell is used for protecting fault map, memory map, and tag arrays. 86

4.10 Performance loss break-down for our scheme in low power mode using SPEC-2K benchmarks. As can be seen, since the fraction of non-functional lines is limited to be less than 10%, the access latency overhead is the dominant factor in performance penalty. 88

4.11 Low-power mode benefits and also overheads of an Alpha 21364 microprocessor system (Table 4.1) augmented with Archipelago. Here, we account for the dynamic power overhead of accessing the second bank in low power mode for handling failures. 89

5.1 Distribution of injected hard-faults that manifest as architectural state mismatches across different latencies – in terms of the number of committed instructions (*CI*). 96

5.2 Number of instructions that are committed (*CI*) before an injected hard-fault results in a violation of a pre-specified similarity index threshold. For this purpose, 5K hard-faults were injected while considering three different similarity index thresholds (90%, 60%, and 30%). 97

5.3 IPC of different DEC Alpha microprocessors, normalized to EV4's IPC. In most cases, by providing perfect hints for the simpler cores (EV4, EV5, and EV4 (OoO)), these cores can achieve a performance comparable to that achieved by a 6-issue OoO EV6. 99

5.4 The high-level architecture of NM is shown in this figure and modules that are modified or added to the underlying cores are highlighted (not drawn to scale). 104

5.5 Port activity breakdown for local caches of the animator core. Here, we show the percentage of cycles that each cache port is either busy or free. For our animator core, the data cache has 2 ports while the instruction cache has a single port. 107

5.6 A code example in which the NM BP performs poorly and switching to the original BP of the animator core is required. The code simply calculates the summation of a 2D-array elements which are stored in a row-based format. It should be noted that the branch prediction release window size is normally set so that the branch prediction accuracy for the entire execution gets maximized. As can be seen, hints are received by the animator core at improper times, resulting in low branch prediction accuracy. 109

5.7 Two high-level examples of cache and branch prediction hint disabling mechanisms. Here, values on the X-axes of the plots correspond to eight entries of the cache disabling table. 112

5.8 The high-level NM design for a large CMP system with 16 cores, modeled after the Sun Rock processor, which has 4 cores per cluster. The details of NM core coupling can be found in Figure 5.4. 115

5.9 Effect of the NM D-cache release window size on the data cache miss rate of the animator core. 120

5.10 Effect of the branch history table size of the NM BP on the overall branch prediction accuracy of the animator core. 120

5.11 Effect of CAM size that are used for reducing the number of D-cache hints – generated in the undead core – on the data cache miss rate of the animator core. Here, the lines show the number of data cache hints should be sent to the animator core per cycle, normalized to the the case without any CAM. 121

5.12 Number of instructions committed in the animator core before the branch prediction hint is disabled for different pre-specified branch prediction hint disabling thresholds (i.e., 50%, 70%, and 90% similarities). 122

5.13 Effect of different resynchronization policies on the overall speed-up of the NM coupled cores normalized to the performance of the baseline animator core. 122

5.14 Effect of communication queue size on the overall speed-up of the NM coupled cores normalized to the performance of the baseline animator core. 123

5.15 Variations in the speed-up of the animator core for different hard-fault locations across SPEC-CPU-2K benchmarks. To only highlight the impact of hard-fault locations, in each row, results are normalized to the average speed-up that can be achieved by the NM coupled cores for that particular benchmark. 124

5.16 Performance of the baseline animator core, NM coupled cores, and a live core normalized to the average performance of a baseline animator core. Due to the higher heterogeneity across the benchmarks for a CMP system with more cores, NM can achieve a higher overall speed-up. 125

5.17 Break-down of NM area and power overheads for CMP systems with different numbers of cores. As can be seen, the overheads that are imposed by the the baseline animator core is typically the major component, which gets amortized as the number of cores grows. 126

5.18 Throughput enhancement for a population of manufactured chips with different number of cores. Here, we consider two baselines, CMP system without and with proper protection for on-chip caches, and show the yield improvement for these two cases (shaded regions) when applying NM. Each line presents the achievable yields for different expected throughput values. 128

LIST OF TABLES

Table

3.1 The target system configuration 41

3.2 Comparison with recently proposed cache protection schemes 59

4.1 The target system configuration 82

4.2 Comparison of different protection schemes 90

5.1 The target NM system configuration 116

5.2 Fault injection locations and their corresponding pipeline stages along with stage-level area break-down for EV6. 117

ABSTRACT

Overcoming Hard-Faults in High-Performance Microprocessors

by

Amin Ansari

Chair: Scott Mahlke

As device density grows, each transistor gets smaller and more fragile leading to an overall higher susceptibility to hard-faults. These hard-faults result in permanent silicon defects and impact manufacturing yield, performance, and lifetime of semiconductor devices. In this thesis, we propose comprehensive, low-cost solutions to tackle reliability problems in high-performance microprocessors. These microprocessors mainly consist of on-chip caches and core pipeline. We first present two flexible cache architectures, ZerehCache and Archipelago, to protect regular SRAM structures against high failure rates. ZerehCache virtually reorganizes the cache data array using a permutation network to provide higher degrees of freedom for spare allocation. In order to study the impact of fault patterns on the redundancy requirements in a cache, we propose a methodology to model the collision patterns in caches as a graph problem. Given this model, a graph coloring scheme is employed to minimize the amount of additional redundancy required for protecting the cache.

Archipelago targets failures in near-threshold region. It resizes the cache to provide redundancy for repairing faulty cells. Furthermore, a near optimal minimum clique covering configuration algorithm is introduced to minimizes the cache capacity loss.

With proper solutions in place for caches, a robust and heterogeneous core coupling execution scheme, Necromancer, is presented to protect the general core area against hard-faults. Although a faulty core cannot be trusted, we observe that for most defects, execution traces on a defective core coarsely resemble those of fault-free executions. Necromancer exploits a functionally dead core to improve system throughput by supplying hints regarding high-level program behavior. We partition the cores into multiple groups. Each group shares a lightweight core that can be substantially accelerated. However, due to the presence of defects, a perfect data or instruction stream cannot be provided by the dead core. This necessitates employing low-cost recovery mechanism and generic hints that are more resilient to local abnormalities.

CHAPTER I

Introduction

The rapid growth of the silicon process over the last decade has substantially improved semiconductor integration levels. However, as device density grows, each transistor gets smaller and more fragile leading to an overall higher susceptibility of chips to failures. This aggressive technology scaling has lead to a host of reliability challenges such as manufacturing defects, wear-out, and parametric variations [23, 19]. These threats can affect correct program execution, perhaps, the most significant aspect of any computer system [13].

1.1 Reliability Threats in Deep Submicron Technologies

Technological trends into the nanometer regime have lead to a host of manufacturing and process issues such as sub-wavelength lithography (e.g., exposure tool optimization), cleaning technology, resist process optimization, higher sensitivity of materials, line edge roughness, random particles attaching to the wafer surface, and random dopant fluctuation [45]. These factors result in a wide distribution of transistor characteristics which translates into an increasing vulnerability of manufactured parts. As the device vulnera-

bility increases gradually, more fault incidences can be observed in modern chips. At the same time, this growth in the fault incidences translates into a larger number of user visible failures in computer systems. Furthermore, propagation of non-masked faults through the system and their manifestation as systematic failures, in many applications (e.g., automotive electronic systems, financial services, and space shuttle management systems), imposes a life-threatening and a significant economic impact.

The sources of computer system failures are widespread, ranging from soft-faults (i.e., transient faults) to hard-faults (i.e., permanent faults). Soft-faults or single event upsets (SEU) are happening due to electrical noise and energetic particle strikes such as neutrons from cosmic rays and alpha particles from packaging material [108]. Such particle strikes can flip a state bit and change the value being computed by a logic element. In recent years, industry designers and researchers have invested significant effort in building architectures resistant to soft-faults [88, 101]. In soft-faults the damage to a chip is never permanent, and a replay of instructions is typically sufficient for recovery.

In contrast to soft-faults, dealing with hard-faults is significantly more involved, and relatively little research has been conducted to efficiently tolerate the same. There are numerous sources of hard-faults, ranging from manufacturing defects, process variation induced failures, to in-field wearout phenomenas such as ElectroMigration (EM) [32], time dependent dielectric breakdown (TDDB) [104], negative bias temperature instability (NBTI) [107], and hot carrier injection (HCI) [97]. These hard faults result in permanent silicon defects, impact the manufacturing yield, performance, lifetime throughput, and dependability of semiconductor parts (e.g., reliability, availability, and maintainability) [23].

1.1.1 Process Variation

Process variation [82], caused by the inability to precisely control the fabrication process at small-feature technologies, introduces significant deviation of circuit parameters (channel length, threshold voltage, wire spacing) from the design. This wide distribution of transistor characteristics directly translates into lower parametric yield [20]. Process variation is encountered at manufacturing time, and influences almost every manufactured chip. The variations can be systematic (e.g., lithographic lens aberrations) or random (e.g., dopant density fluctuations), and can manifest at different levels – wafer-to-wafer (W2W), die-to-die (D2D) and within-die (WID). Traditionally, D2D has been the most visible form of variation, and was tackled by introducing the notion of speed-binning (chips are partitioned based on their frequency and sold accordingly). However, the increasing levels of WID variations [82, 64] have created newer challenges. This significant divergence of process parameters from their nominal specification limits the achievable frequency and also significantly hurts the leakage power of modern high performance processors [95]. A conventional approach to deal with process variation is to introduce large voltage/frequency guard-bands which considerably impacts the power consumption.

1.1.2 Manufacturing defects

Manufacturing defects is one of the main challenges for the semiconductor industry, which have a direct impact on yield. From each process generation to the next, microprocessors become more susceptible to manufacturing defects due to higher sensitivity of materials, random particles attaching to the wafer surface, and sub-wavelength lithography

issues such as exposure tool optimization, cleaning technology, and resist process optimization [45]. For instance, based on the latest ITRS report [48], for current and near future CMOS technology, one manufacturing defect per five $100mm^2$ dies can be expected. Thus, in order to maintain an acceptable level of manufacturing yield, a substantial investment is required [87]. Traditionally, modern high-performance processors are declared as functional if all parts of the design are fault-free, or if they can operate correctly by tolerating failures. However, since manufacturing defects can cause a significant yield loss, semiconductor companies have recently started to manufacture parts that have been over-designed to hedge against defects. For instance, to improve yield, IBM did this with the Cell Broadband Engine that sometimes only had 7 out of the 8 processing elements activated [90].

1.1.3 Wearout

Apart from these fabrication challenges, as circuit density grows, each transistor gets smaller, hotter, and more fragile. This leads to an overall higher susceptibility of chips to in-field wearout induced hard-faults [23, 91]. For many computer systems such as embedded systems, data center processors, and space equipments, the device lifetime and throughput is crucial. However, these wearout failures, can impact the performance guarantees offered by a semiconductor chip, and limit their useful lifetime. Wearout or aging can be tackled by either proactive or reactive methods. In proactive methods, the operation of the device should be altered such that it avoids phenomenas that are causing aging (e.g., EM, TDDB, and NBTI). On the other hand, in reactive methods, the wearout induced failures will be addressed without slowing the aging process. In general, wearout avoidance techniques (i.e., proactive approaches) have a very limited scope and mostly try to avoid one or two

of the aforementioned wearout mechanisms. Therefore, in order to develop a more general solution for hard-faults, in this thesis, we focus on dealing with the hard-faults after they occur (i.e., a reactive approach).

1.1.4 Power Consumption

With aggressive silicon integration and clock frequency increase, power consumption and heat dissipation have become key challenges in the design of high performance processors. Growing power consumption reduces device lifetimes and expedites early stage failures [91]. It also affects the cost of thermal packaging, cooling, electricity, and data center air conditioning [59]. Dynamic voltage scaling (DVS) is a widely used technique to reduce the power consumption of microprocessors, exploiting the fact that dynamic power quadratically scales with voltage and linearly with frequency. However, the supply voltage of a microprocessor cannot be reduced below a certain threshold without drastically sacrificing clock frequency. Lowering this minimum achievable voltage can dramatically improve the lifetime, energy consumption, and battery life of medical devices, laptops, and handheld products.

The minimum achievable voltage for DVS is set such that under the worst-case process variation, the processor operates correctly [35]. Large SRAM structures are limiting the extent to which operational voltages can be reduced in modern processors. This is because SRAM delay increases at a higher rate than CMOS logic delay as the supply voltage is decreased [94]. Furthermore, with increasing systematic and random process variation in deep sub-micron technologies, the failure rate of SRAM structures rapidly increases in the near-threshold regime. Ultimately, the minimum sustainable V_{dd} of the entire cache

structure – and consequently the core as a whole – is determined by the one SRAM bit-cell within the entire system with the highest required operational voltage. This forces designers to appropriate a large voltage margin in order to avoid on-chip cache failures.

Efficiency of CMOS technology is questionable in the face of such challenges. Current projections indicate that future microprocessors will be composed of billions of transistors, many of which will be unusable at manufacture time, and many more which will degrade in performance (or even fail) over the expected lifetime of the processor [23]. These issues are detrimental to the semiconductor industry's economic model. Loss of compelling performance gains reduces the incentive to regularly upgrade machines, loss in yield directly translates to loss in sales and in-field defects could necessitate conservative designs to avoid substantial performance degradation. To address these reliability concerns, designers must armor their designs to tolerate and operate properly in the presence of faults.

1.2 Overcoming Hard-Faults in High-Performance Microprocessors

Traditionally, hardware reliability was only a concern for high-end systems (e.g., HP Tandem Nonstop and IBM eServer zSeries) for which applying high-cost redundancy solutions such as triple modular redundancy (*TMR*) was acceptable. Nevertheless, hardware reliability has already become a major issue for mainstream computing, where the usage of high-cost reliability solutions is not acceptable [63]. Therefore, there is a need for low-cost reliably solutions to maintain the correctness and enhance the efficiency of modern microprocessors.

Traditionally, modern high-performance processors are declared as functional if all

parts of the design are fault-free, or if they can operate correctly by tolerating failures. However, since manufacturing defects can cause a significant yield loss, semiconductor companies have recently started to manufacture parts that have been over-designed to hedge against defects. For instance, to improve yield, IBM did this with the Cell Broadband Engine that sometimes only had 7 out of the 8 processing elements activated [90].

1.2.1 Challenges with High-Performance Microprocessors

Protecting high-performance modern microprocessors against hard-faults is more challenging compared to other CMOS devices. Here, we explain some of these difficulties which do not exist to the same degree in other circuit designs. First, high-performance microprocessors contain on the order of several hundred million transistors to allow a more aggressive extraction of instruction level parallelism. Failure of any of these transistors can potentially impact the correct operation of the microprocessor. Therefore, since each core contains a large number of transistors, simple reliability solutions like disabling the faulty core (i.e., core disabling) is not cost effective. Furthermore, this growth in the number of transistors per core, increases the chance of having more faulty transistors in a given core.

More importantly, the design complexity of the modern high-performance microprocessors increases everyday. Complexity in the connectivity between different stages, and also having a larger number of stages, do not allow us to use techniques like Core Cannibalization [79] and StageNet [41] which suggest breaking each core into pipeline stages and allowing one core to borrow stages from other cores through interconnection networks. These techniques designed for simple in-order cores with a regular, straightforward connectivity between stages. Furthermore, the operational clock frequency of these micro-

processors is relatively high. This tight delay budget and the inherent complexity in the connectivity, makes it almost impossible to use fine-grained spares (with the same set of connectivity) for every structures.

In addition, in order to achieve a better performance, these high-performance microprocessors mostly operate at a higher clock frequency, voltage, and temperature. Since higher operational stress (e.g, temperature, current, and voltage) accelerates the aging process, there is much higher chance that in-field wearout failures occur in these processors. Therefore, in order to combat the impact of different wearout mechanisms (e.g., NBTI, HCI, or EM), proper reliability solutions are even more necessary for high-performance microprocessors.

Inserting SCAN chains everywhere in a high-performance microprocessor imposes a high performance, area, and power cost. Therefore, they can only be applied in a limited set of locations to enhance the observability and countability of test process. This simply translates to a state explosion during test procedure. Therefore, it is hard (if not impossible) to pinpoint the fault location in most cases (especially for in-field failures). This introduces another challenge with protecting the high-performance microprocessors against faults. This implies that the reliability solutions which are not relying on exact fault location would be more appealing in the future. However, this is contradictory with the conventional method of designing fault-tolerant systems which emphasizes the fault diagnosis to allow a finer-grained replacement/reconfiguration.

Above all, given a particular technology node, high-performance microprocessors mostly operate on the most aggressive voltage vs clock frequency curve. This means operating at the highest possible frequency at a given supply voltage. There is set of simple reliability

techniques that suggest using high voltage and frequency guard-bands or slowing down the processor to avoid wearout and process variation induced failures. However, these simple techniques eliminate most of the achievable performance of these microprocessors and/or introduce high power overhead.

Another challenge with high performance microprocessors, is the introduction of large on-chip caches. These multi-level caches allow the processor to achieve a better performance by effectively hiding the main-memory latency. However, protecting these large and delay sensitive structures against reliability issues with conventional approaches is not practical. The main reason behind this is that the voltage and frequency of the SRAM array should be set based on the bit-cell with worst timing characteristics. Process variation causes different SRAM cells to show different timing characteristics. Therefore, as the population of the cells grow (larger cache), the chance of having a cell with a very poor timing characteristics increases. This means as the SRAM array size increases, in order to cope with process variation, power and timing efficiency of the array should to be sacrificed.

1.2.2 Protecting On-Chip Caches

On-chip memory arrays in high-performance processors are critical for chip reliability as more than 70% of the transistors can be devoted to caches. Moreover, as the technology scaling continues, in order to deal with the power budget, without power gating a large fraction of the chip, even a larger fraction of the chip is expected to be devoted to on-chip caches. These SRAM structures, however, are particularly vulnerable to the process variation due to their minimum-geometry transistors, sensitive differential circuit, and area efficient semi-custom layout. Therefore, these large on-chip caches in the high-performance

microprocessors need to be protected against different sources of hard-fault (e.g., process variation, wearout, and manufacturing defects).

To efficiently scale to higher defect densities and handle arising hard-faults, a more flexible and configurable cache design is necessary. As a solution, we present and evaluate ZerehCache that is a high-failure rate tolerant cache design for both L1 and L2 on-chip caches. ZerehCache is an adaptive, dynamically reconfigurable solution for tackling the high defect rates of future technologies. It also provides a wide range of cache design options based on the primary design concerns such as delay, power, and area overhead. The cache data array is divided into equal sized groups. Lines within each of these groups share a single spare line in the spare cache. In order to tolerate many defects, logical groups are formed by carefully shuffling together cache lines using an interconnection network. The functionality of the interconnection network is to swap the lines in a manner that resolves the existing collisions. We model collisions as a graph coloring problem that can be solved during the manufacturing test time to minimizes the amount of redundancy required for protecting the cache. In this thesis, we leverage ZerehCache architecture to tolerate process variation in 45*nm* technology. ZerehCache takes advantage of its interlaced usage of redundancies in multiple ways to substantially cut the overheads of protecting on-chip caches. Current microprocessors have already been equipped with ECC and row-redundancy to protect the caches [80]. ZerehCache can substitute these conventional protection mechanisms, while providing the same level of robustness, for a considerably lower overhead.

Apart from wearout and manufacturing issues, power consumption is a major concern for semiconductor industry. As mentioned earlier, DVS is a widely used technique to reduce the power consumption of microprocessors. However, the supply voltage of a mi-

croprocessor cannot be reduced below a certain threshold without drastically sacrificing clock frequency. Therefore, the minimum achievable voltage for DVS is set such that under the worst-case process variation, the processor operates correctly. Since large SRAM structures are limiting the extent to which operational voltages can be reduced in modern processors, in order to enable DVS to push the core/processor operating voltage down to the near-threshold region, correct functionality of on-chip caches should be preserved. For this purpose, we propose Archipelago, a cache capable of reconfiguring its internal organization to efficiently tolerate the large number of SRAM failures that arise when operating in the near-threshold region. Since low-power operation is optional, in order to minimize the overheads, Archipelago does not rely on an separate spare cache, instead, it resizes the original cache to provide spare elements. Archipelago allows fault-free operation by partitioning the cache into multiple autonomous islands with various sizes. Each island is a group of physical cache word-lines that can operate correctly without using any word-line outside of their group. Each group has a sacrificial word-line which is divided up to multiple redundancy units. These spare units are directly/indirectly employed to achieve fault-free operation of the other word-lines in the same group. Furthermore, an adapted version of the minimum clique covering algorithm is used to partition the cache to the least number of islands to minimize the number of sacrificial word-lines required for guaranteeing the fault-free operation of the cache.

1.2.3 Protecting Non-Cache Parts of the Core

With appropriate protection mechanisms in place for caches, the processing cores become the major source of defect vulnerability on the die. Consequently, in the second half

of this thesis, we try to address hard-faults in the non-cache parts of the processing core. Due to the inherent irregularity of the general core area, it is well-known that handling defects in the non-cache parts is challenging [75]. The industry is currently dominated by Chip Multi-Processor systems with only a modest number of high-performance cores (e.g., Intel Core 2), systems which cannot afford to lose a core due to manufacturing defects. Therefore, these common solutions like core disabling or isolation are not a cost-effective. The other extreme of the solution spectrum lies fine-grained micro-architectural redundancy. Here, broken microarchitectural structures, such as ALUs, are isolated or replaced to maintain the functionality. Unfortunately, since the majority of the core logic is non-redundant, the fault coverage from these approaches is very limited [75].

To enhances overall system throughput and mitigates the performance loss caused by defects in the non-cache parts of the core, we preset Necromancer. Necromancer relaxes the correct execution constraint on a faulty core since it cannot be trusted to faithfully execute programs. However, we observed for most defect instances, the execution flow of the program on the faulty core coarsely resembles the fault-free program execution on the animator core when starting from the same architectural state. Therefore, Necromancer leverages high level execution information (hint) from the faulty core to accelerate the execution of a lightweight core. This lightweight core is an additional core, introduced by Necromancer, that is an older generation of the baseline cores in the CMP with less resources and the same instruction set architecture. Moreover, in the lightweight core, these hints are only treated as performance enhancers and do not influence execution correctness. We partition the cores in a conventional CMP system into multiple groups in which each group shares one of these lightweight cores that can be substantially accelerated.

To prevent the faulty core from wandering too far from the correct path of execution, we dynamically resynchronize architectural state with the lightweight core. However, due to the presence of defects, a perfect data or instruction stream cannot be provided by the faulty core. This necessitates employing generic hints that are more resilient to local abnormalities. Given the variation in the usefulness of the execution information, in order to enhance the efficiency of the lightweight core, we introduce several fine-grained hint disabling mechanisms. Besides, when the faulty core gets completely off the correct execution path, hints become useless, and it needs to be brought back to a valid execution point. Therefore, the architectural state of the lightweight core can be copied over to the faulty core. For this purpose, we leverage coarse-grained online monitoring of the effectiveness of the hints over a large time period to decide whether the faulty core should be resynchronized with the lightweight core.

1.3 Contributions

Before enumerating the contributions, we go over some of the common attributes of proposed fault-tolerant architectures in this thesis. In order to achieve low-cost reliability solutions for commodity high-performance microprocessors, we realized the future computer systems need to have several characteristics: runtime adaptability, high degree of reconfigurability, fine-grained flexibility in spare substitution, and ability to exploit approximate execution. These characteristics allow a system to tolerate failures with minimum costs by dynamically reorganizing itself. In this thesis, we make the following contributions:

- We demonstrate a comprehensive, low-cost approach for protecting high-performance modern microprocessors against different sources of hard-faults in deep submicron technology nodes.

- A flexible cache architecture, ZerehCache, is presented to protect regular SRAM structures against high degree of process variation, wearout induced failures, and manufacturing defects. Furthermore, we propose a methodology to model the collision pattern in the cache as a graph problem. Given this model, a graph coloring scheme is employed to minimize the amount of additional redundancy.

- To efficiently tolerate the large number of SRAM failures that arise, in the large on-chip caches, when operating in the near-threshold region, a highly reconfigurable cache design, Archipelago, is presented. Since low-power operation is optional, instead of relying on redundancy, Archipelago resizes the cache to provide spare elements. Furthermore, a near optimal minimum clique covering configuration algorithm is introduced.

- A robust and heterogeneous core coupling execution scheme, Necromancer, is presented to tackle hard-faults in the non-cache parts of the core. Although a faulty core cannot be trusted to correctly execute programs, we observe that for most defects, when starting from a valid architectural state, execution traces on a defective core actually coarsely resemble those of fault-free executions. In light of this insight, Necromancer exploits a functionally dead core to improve system throughput by providing high-level hints to accelerate the execution of a fault-free core.

1.4 Organization

The rest of this thesis is organized as follows. First, Chapter III presents ZerehCache, a high-defect tolerant on-chip cache architecture that combines redundant data array elements with a permutation network for providing a higher degree of freedom on replacement. In Chapter IV, we introduce Archipelago, a highly flexible fault-tolerant cache design that by reconfiguring its internal organization can efficiently tolerate the large number of SRAM failures that arise when operating in the near-threshold region. Next, in order to tackle hard-faults in the non-cache parts of the core, in Chapter V, we propose a robust and heterogeneous core coupling execution scheme, Necromancer, that exploits a functionally dead core to improve system throughput by supplying hints regarding high-level program behavior. Finally, Chapter VI presents our directions for future work.

CHAPTER II

Related Work

A significant amount of literature targets the microprocessor reliability concerns (e.g., transient faults, manufacturing defects, process variation, wearout avoidance/tolerance, and stability in near/sub-threshold operation). In this section, based on the aforementioned scope of this thesis (Chapter I), we divide the prior work into three major categories: fault-tolerant cache techniques, low-power cache techniques, and techniques for handling hard-faults in the non-cache parts of the core. Here, SRAM cell stability efforts are included in the low-power category since they try to proactively avoid failures.

2.1 Fault-Tolerant Cache Techniques

The proposed solutions in this domain can be divided into three major categories:

2.1.1 Coding Solutions

Simple error detection codes (EDC) and parity can be applied for the detection of the faults in caches [80]. Single error correction double error detection (SECDED) is a widely used technique for protecting the memory structures against soft-errors. However, in a

high-failure rate situation, these solutions are not practical because of the strict bound on the number of tolerable faults in each protected data chunk (Section 3.6). A 2D error correction coding scheme is presented in [54] that uses two sets of EDCs on the rows and columns of the data array. As the failure rate sensitivity analysis results show in [54], this scheme is not appropriate for tolerating large number of randomly distributed failures. Further, the overhead of updating all the column codes for *each cache write* is high. Multiple bit error correcting codes (ECCs) like Hamming codes are capable of tolerating high failure rates, but are inefficient in terms of the coding delay, area, and power overheads for on-chip caches [54]. In summary, the coding solutions are best applied to memory structures under low failure-rate scenarios or where transient faults are the main concern.

2.1.2 Circuit-Level and VLSI Solutions

Many solutions have been proposed that employ dynamic voltage/frequency scaling to improve the cache reliability [74]. These methods try to identify the most vulnerable SRAM cell in each line and scale the access time/voltage to a level that guarantees proper operation for all the cells in that word-line. There are two major drawbacks of this scheme, 1) a mechanism is needed to dynamically determine the weakest cell in each row, and 2) the working conditions of the cache must be adjusted to the weakest cell, resulting in a considerable performance penalty (access latency). A 3T1D DRAM cell can be substituted for the conventional 6T SRAM cell to improve the reliability [64]. However, the 3TD1 cell cannot retain the value for a long period and each word-line must be refreshed periodically. Moreover, since on-chip DRAM is not normally used in current technologies, it adds to the process/manufacturing complexity and effort. Another alternative is to size up the SRAM

cells or use a different structure for them (e.g. 8T, 10T, or ST) [58]. Unfortunately, these methods incur a large area overhead (Section 3.6) and they are mostly employed for power reduction by allowing the near/sub-threshold operation.

2.1.3 Architectural Solutions

Dual modular redundancy (DMR) schemes are used in many designs for providing memory structure reliability, but they are highly inefficient in terms of the overhead [83]. A popular architectural solution is to use redundant rows and/or columns [62]. However, as it will be discussed in Section 4.1 and 3.1, for our target failure rate, almost all word-lines/columns can be expected to be faulty from the start (Figure 4.2 and 3.1b). This results in a poor utilization of the provisioned redundancy. Moreover, since the redundant row replacement is based on a decoder modification and using hard-wired fuses, it is generally not applicable for more than 10 extra rows [46]. A similar set of methods are based on the cache block/row/way disabling that are also suitable for the low-failure rate situations [74]. Wilkerson et. al. have suggested several layers of shifters for merging multiple defective word-lines to form a single functional word-line [103]. To achieve operation in the presence of faults, their Word-Disable method sacrifices half of the cache area and their Bit-Fix method adds three cycles of latency to the cache access time. Both of which result in considerable performance drop-off. There are other groups of work that use a re-mapping table to map a faulty block onto one of neighboring functional blocks [47]. These methods impose a high pressure on the L1-L2 communication bus by increasing the L1 miss rate substantially. Furthermore, these methods have two major applicability issues: they are properly applicable only to *direct-mapped* caches [4]; and, they cannot be applied to L2

caches since a read from a faulty block results in a miss that gets its value from main memory with several hundred cycles latency.

2.2 Low-Power Cache Techniques

The proposed solutions in this domain can be divided into three major categories:

2.2.1 Conventional Low-Power Cache Techniques

The usage of V_{dd} gating for leakage power reduction by turning off cache lines is described in [51]. This approach reduces the leakage power of the cache by turning off the cache lines that are not likely to be accessed in the near future. Meng et. al. [68] proposed a method for minimizing leakage overhead in the presence of manufacturing variations. In this scheme, they artificially prioritize cache ways with smaller leakage and resize the cache by avoiding sub-arrays that have higher leakage factors. Instead of turning off blocks, drowsy cache [36] is a state preserving approach that has two different supply voltage modes. In order to save power, recently inactive cache blocks periodically fall into a low power mode in which they cannot be read or written.

2.2.2 Lowering Power by Tolerating Failures in On-Chip Caches

However, for low V_{dd} values (e.g., $\leq 651mV$ in $90nm$), the amount of power saving for these methods is restricted due to failures in SRAM structures [78]. In contrast, as we discussed earlier, our objective is to enable DVS to push the processor/core operating voltage down to the near-threshold region while preserving correct functionality of on-chip caches. Wilkerson et. al. [103] proposed two different cache protection schemes for L1

and L2 caches that use several levels of decoding/shifting to take the faulty data chunks out and replace them using ECC protected patches. Due to the strict binding between data and redundancy, their schemes need to disable 50% of L1 and 25% of L2 caches which results into a considerable performance drop-off in low power mode. Recently, Abella et. al. [1] proposed a cache protection scheme based on sub-block disabling which can provide a better performance predictability than [103]. However, since this scheme relies on disabling finer granularities than a cache block, it losses its efficiency when applied to caches other than L1-Data. Chishti et. al. recently proposed another technique [31] that employs multi-bit segmented ECC to also allow soft and hard-error resilience in lower voltages by sacrificing 50% of cache capacity. Although an interesting approach, it can only achieve 30% reduction in the MinV_{dd}.

2.2.3 Alternative SRAM Cells

On the other hand, many variations of SRAM cells such as 8T [70], 10T [27], 11T [69], and ST [58] have also been proposed. These larger SRAM cells are more stable against different sources of parameter variations compared to the conventional 6T cell and allow the SRAM structures to operate at lower voltages while preserving its correct functionality. Most of these cells have a large area overhead which is a significant shortcoming since the extra area does not translate into any performance gains when operating in high power mode.

2.3 Handling Hard-Faults in the Non-Cache Parts of the Core

The proposed solutions in this domain can be divided into three major categories:

2.3.1 Coarse-Grained Redundancy and Disabling

Manufacturing defects can cause transistors in different parts of a microprocessor to get corrupted. Prior work on defect tolerance mostly focused on on-chip caches since there is less homogeneity in the non-cache parts of a core, making defect tolerance a more challenging issue. Typically, for high-end server systems designed with reliability as a first-order design constraint (e.g., HP Tandem NonStop [16], Teramac [34], and the IBM eServer zSeries [16]), coarse-grained replication has been employed [18, 89]. Configurable Isolation [6] is a high availability chip multiprocessor architecture for partitioning cores to multiple fault domains which allows independent redundant executions. However, dual and triple modular redundant systems incur significant overheads in terms of area and power which is not generally acceptable for mainstream computing. An easy solution is to disable the faulty cores – to avoid yield loss – which clearly causes a significant reduction in the system throughput and sale price [6]. This simple core disabling approach has been taken by microprocessor vendors, such as IBM, Intel, AMD, and Sun Microsystems, to maintain an acceptable level of manufacturing yield.

2.3.2 Fine-Grained Redundancy and Disabling

Core Cannibalization [79] and StageNet [41, 42, 43] suggest breaking each core into pipeline stages and allowing one core to borrow stages from other cores through inter-

connection networks. Introduction of these interconnection networks in the processor pipeline presents performance, power consumption, and design complexity challenges. Finer-grained redundancy maintenance has been used by Bulletproof [33] and sparing of array structures [24]. In the same vein, Shivakumar et. al. [87] proposed a method to disable non-functional microarchitectural components (e.g., execution units) and faulty entries in small array structures (e.g., register file). Rescue is mainly a microarchitectural design-for-test (*DFT*) technique which can map out faulty pipeline units that have spares [84]. However, as shown in [75], these schemes have a limited applicability due to the small amount of microarchitectural redundancy that exists in a modern high-performance processor.

2.3.3 Unconventional Approaches

Architectural Core Salvaging [75] is a high-level low-cost architectural proposals which uses thread migration between the cores to guarantee the correct execution. To avoid incorrect execution, for each instruction, it assesses whether the fault location might be exercised by the corresponding *opcode*. Thus, without using extra redundancy, it is only applicable to defects in about 10% of core area. DIVA [13] was proposed for dynamic verification of complex high-performance microprocessors. It employs a checker pipeline that re-runs the same instruction stream for ensuring correct program execution. Given the fact that DIVA is not a defect tolerant scheme, as shown in [13], a "catastrophic" core processor failure results in about 10X slow-down. Detour [67] is a completely software-based approach which leverages binary translation for handling defects in execution units and register files. Apart from limited defect types that can be handled, a binary translation layer cannot typically be

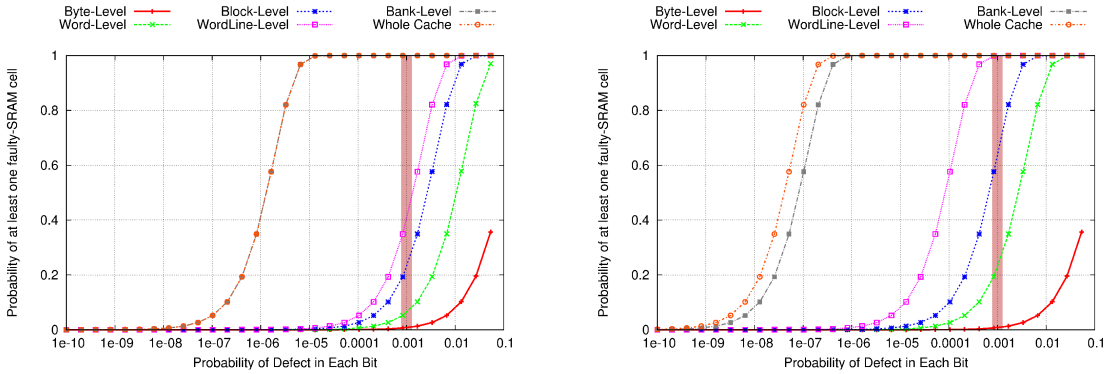
applied to high-performance x86 cores [75].

CHAPTER III

Armoring Cache Architectures in High Defect Density Technologies

3.1 Introduction

On-chip memory arrays in high performance processors are critical for chip reliability as more than 70% of the transistors can be devoted to caches. These SRAM structures, however, are particularly vulnerable to the process variation due to their minimum-geometry transistors, sensitive differential circuit, and area efficient semi-custom layout. The yield of an *unprotected* cache in the 45*nm* technology can be as low as 33%, implying the necessity of proper protection [4, 5]. Under process variation, a single SRAM cell can fail because of the following reasons that are sorted based on the frequency of occurrence [4]: **1.** Access Time Failure: It occurs when the differential read voltage between bit-lines is not enough for the sense amplifier to extract the correct stored value. **2.** Write Stability Failure: This case arises when the cell contents cannot be replaced with a new value. This happens due to stronger pull-up of the storage node with input *0* compared to the access transistor. **3.** Read Stability Failure: If the voltage of a storage node with a stored *0* value



(a) A 1-bank 64KB L1 cache with 32B block size and 64b word size

(b) A 2-bank 2MB L2 cache with 128B block size and 256b word size

Figure 3.1: Probability of having at least one faulty SRAM cell at different granularities while varying the failure probability of each SRAM cell, P_F

during the read operation is higher than the trip-point of an inverter that has an output value of 1, the read value from the SRAM cell flips. **4.** Hold Failure: If the supply voltage drops below a minimum level while an SRAM cell is not being accessed, the SRAM cell can lose its stored value.

To illustrate the reliability implications on L1/L2 caches, Figure 3.1 presents the probabilities of having at least one faulty SRAM cell considering different granularities of storage. This trend is shown for a wide range of single cell failure probabilities (P_F), assuming a uniform failure distribution. As the failure probability increases in these graphs, the granularity of fault manifestation decreases in size. For instance, the L2 cache configuration demonstrates a modest number of block-level failures at $P_F \sim 10^{-5}$. Thus, a block-level redundancy solution would be satisfactory for fault tolerance in this case. However, at $P_F \sim 10^{-3}$, the L2 cache is certain to contain at least one faulty cell in each cache word-line with a very high chance of fault in each cache block. This makes the use of word-line/block level redundancy impractical. Hence, with the increasing failure probability, a smaller granularity of redundancy will be necessary to guarantee robustness. The same

trends can be seen for the L1 cache, but it is favorably shifted towards the right due to the smaller block and word-line sizes of the L1 cache in comparison to the L2 cache. The primary challenge in this scenario is to design a cache architecture that can maintain and optimally utilize the smaller levels of redundancy for defect tolerance. At a 45*nm* technology node, an SRAM cell is expected to have a 30*mV* standard deviation in the threshold voltage (V_{th}) resulting in a P_F as high as 10^{-3} [5]. Thus, there is a real need to devise solutions for this reliability challenge.

Apart from fabrication challenges, as circuit density grows, each transistor gets smaller, hotter, and more fragile. This leads to an overall higher susceptibility of chips to *permanent faults* [23, 91]. These wearout failures, can impact the performance guarantees offered by a semiconductor chip, and limit their useful lifetime. To combat such a scenario, on-chip caches need to be equipped with cost-effective mechanisms to tolerate in-field silicon defects. In this chapter, in addition to process variation tolerance, we expand the functionality of our proposed cache architecture to tackle wearout failures over time. This will extend the effective lifetime of the on-chip caches and prevent early lifetime failures. Assuming no manufacturing defects, Figure 3.2 depicts the fraction of non-functional SRAM bit-cells for a 2MB L2 cache over time. This plot was generated for a range of mean time to failure (*MTTF*) values from 50 to 200 years. Here, it is notable that even for an MTTF of 200 years, a considerable number of failures need to be addressed in early lifetime. Moreover, a comparison of our scheme with conventional wearout tolerance methods and also the experimental methodology, for generating this plot, will be discussed in section 3.5.

To efficiently scale to higher defect densities, a more flexible and configurable solution is necessary. To this end, we introduce the ZerehCache (ZC, Zereh in Farsi means body ar-

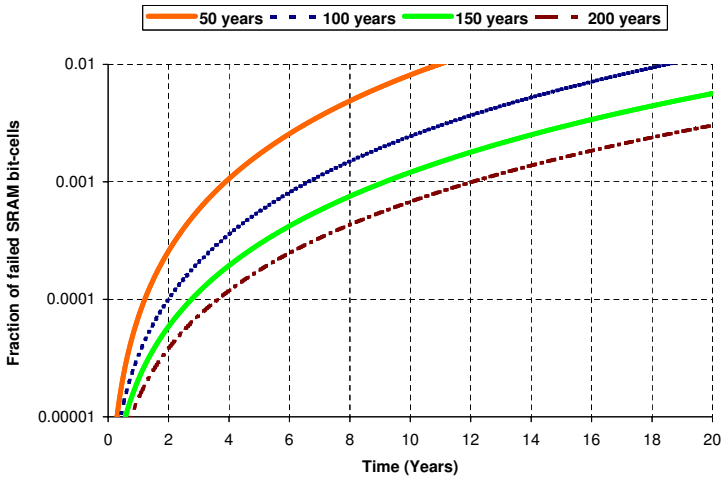


Figure 3.2: Fraction of non-functional SRAM bit-cells for a 2MB L2 cache over time. Here, the mean time to failure of each SRAM bit-cell is varied from 50 to 200 years.

mor) [11, 12], a high-failure rate tolerant solution for both L1 and L2 on-chip caches. ZC is an adaptive, dynamically reconfigurable solution for tackling the high defect rates of future technologies. It also provides a wide range of cache design options based on the primary design concerns such as delay, power, and area overhead. In this chapter, the ZC architecture is leveraged to tolerate process variation in 45*nm* technology. ZC takes advantage of its intelligent interlaced usage of redundancies in multiple ways to substantially cut the overheads of protecting on-chip caches. To our knowledge, ZC has the capability of achieving the highest degree of the fault tolerance, among the previously proposed approaches, for a given area budget. Current microprocessors have already been equipped with ECC and row-redundancy to protect the caches [80]. ZC can substitute these conventional protection mechanisms, while providing the same level of robustness, for a considerably lower overhead (Section 3.6). We believe our scheme provides a solid foundation for the cache designers to take advantage of the higher clock frequency and transistor density in the deeper technology nodes while preserving the correct functionality and timing constraints

of their design with less overhead.

The primary contributions of this chapter are: 1) A flexible, dynamically reconfigurable architecture that can be leveraged to protect regular SRAM structures against high defect density nanometer technology nodes; 2) Minimizing the amount of redundancy required for protecting the cache by modeling the collision pattern in the main/spare cache with a well studied graph coloring problem and taking advantage of the existing rich approximation methods; 3) A design space exploration in 45*nm* to show the actual process of fixing the architecture parameters; and 4) Derivation and modeling of the manufacturing yield and evaluating the proposed method under process variation conditions.

3.2 ZerehCache

In this section, the ZC architecture is first described that adaptively reconfigures itself to absorb failing SRAM cells. Next, an effective graph-coloring algorithm to configure the underlying architecture is presented.

3.2.1 ZC Architecture

The key idea behind the ZC architecture is to use redundant units in multiple ways to increase their potential utilization. ZC partitions the complete cache array into sets of equally sized logical groups, where each logical group is allocated one spare cache word-line. Here onwards, we use the term *line* when referring to a *word-line*. The logical groups are formed by carefully shuffling together physical cache lines in order to optimize the utilization of a single spare cache line. Each cache/spare line is divided up to equally sized

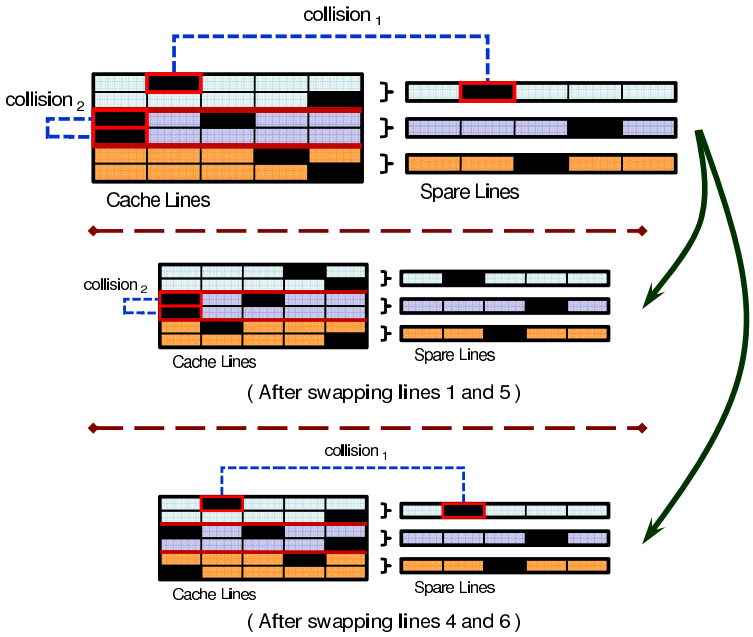


Figure 3.3: Two simple scenarios in which the line swapping can preserve the correct functionality of the cache by resolving the occurred collision. A black box shows a faulty chunk of data.

data chunks to allow smaller granularities of spare substitution. For instance, the fourth data chunk of the second spare line can be used to substitute the fourth data chunk of the third/fourth cache line in the case of failure. Flexibility for this line shuffling is provided by adding a network into the cache that allows swapping of cache lines to eliminate conflicting failures. Conflicting failures occur when two lines that share a spare line have a failure in the same chunk, or when a cache line and its corresponding spare have failures in the same chunk. While there may be sufficient redundancy, conflicting failures arise and render the cache non-operational.

To illustrate this issue, Figure 3.3 shows two simple scenarios where ZC can preserve the correct functionality of the underlying cache while it is not possible for conventional redundancy methods to do so. In this figure, each line contains five units of data and every two consecutive lines in the main cache form a logical group. Each logical group is

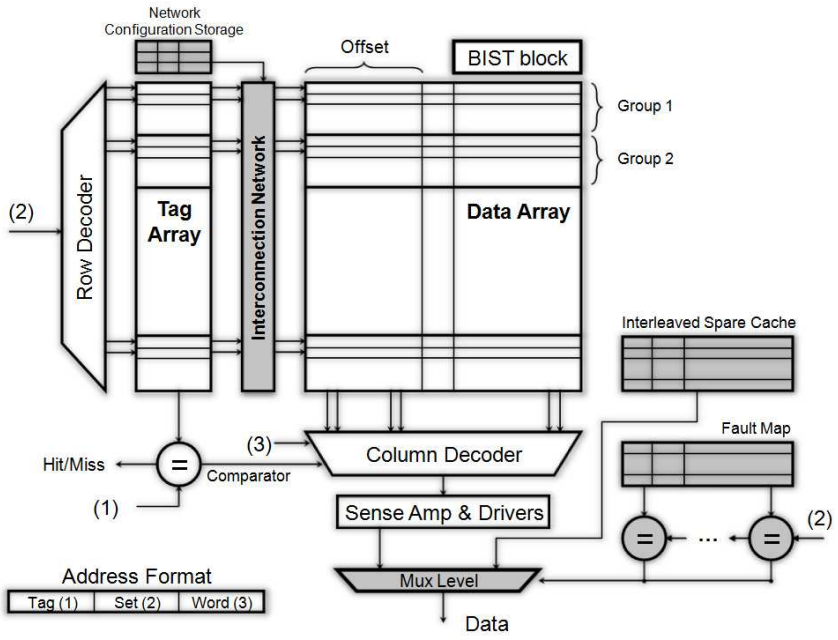


Figure 3.4: The high-level architecture of the ZC is shown in this figure and the extra modules that are added to the baseline cache are highlighted. Note that the slices of the base address are shown using numbers 1, 2 and 3 (Address Format). The fault map array and spare cache have their own shared decoder to avoid getting their word-line activation signals from the main cache's decoder. For simplicity, the separate sense amps for the fault map and spare cache are not shown. Built-in-self-test (BIST) module is commonly used for fault diagnosis in the embedded memory structures.

assigned one line in the spare cache. The first line in the main cache has a failure in the same place as the first line of the spare cache. Swapping the first and fifth lines in the main cache can resolve this collision ($collision_1$). After swapping, the second and fifth rows will form the first logical group which utilizes the first row of the spare cache. The second conflict situation, $collision_2$, is between two lines in the same logical group. Swapping the fourth and the sixth lines is one possible way to resolve this conflict. Swapping eliminates collisions and increases the chance of having a functional cache for a given area overhead budget.

A high-level architecture of a set-associative ZC is shown in Figure 3.4. The cache data

array is divided into equal sized groups. Lines within each of these groups share a single spare line in the spare cache (static multiplexing of spares). For each access to the cache array, in a high speed design, the spare cache and the fault map arrays are also accessed in parallel. The result of the fault map access determines whether the spare data chunk should be routed to the output instead of the main cache content. In order to tolerate many defects, logical groups are formed by carefully shuffling together cache lines using an interconnection network. As Figure 3.3 demonstrates, the functionality of the interconnection network is to swap the lines in a manner that resolves the existing collisions. The configuration for this network is computed once and saved in the non-volatile network configuration storage. By using static multiplexing and the interconnection network for overcoming the limitations of static binding, ZC can maximize the utilization of the spare units. The remainder of this section provides a detailed description for each of the architectural modules.

Spare Cache: Each row in the spare cache corresponds to a logical group of lines in the main cache. A single row of the spare cache is further broken up into smaller redundancy units of fixed size. Each of these redundancy units in the spare cache keeps the valid content of the corresponding corrupted element in the main cache (if any exists). In order to avoid high fan-in ORs required when using the main cache row decoder, the spare cache and fault map arrays use a separate shared decoder. This decoder uses the top n most significant bits from the set segment of the memory address, where n is based on the number of rows in the spare cache.

Interconnection Network: In order to shuffle around the cache lines and form logical groups, we use an interconnection network. This network is placed between row decoder of the main cache and the cache word-lines. A unidirectional Benes network (BN) [73] is

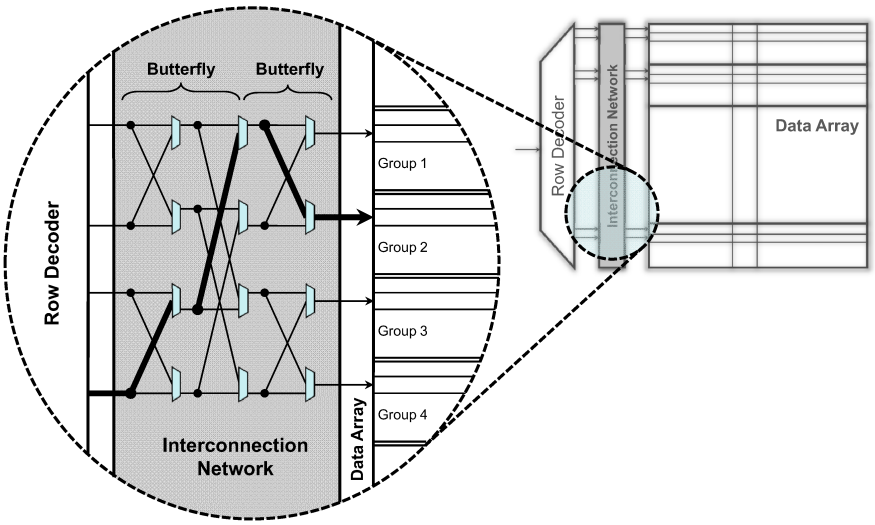


Figure 3.5: A Benes network is shown which connects the second rows of the four consecutive logical group of rows in the main cache. As an example, a single route from the decoder to the word-lines is also shown.

used to provide a non-blocking routing and full permutation mapping between the inputs and outputs. As Figure 3.5 shows, a BN consists of two back-to-back connected butterfly networks. The main reasons for selecting a BN for this work are: 1) Full permutation and non-blocking properties allow routing any permutation from inputs to outputs in a conflict-free fashion. 2) Logarithmic depth of the net can minimize the imposed delay overhead of the interconnection network. For connecting 2^n nodes to each other, $2n - 1$ stages are required. We call this a BN with *n* swapping levels. 3) The BN delay/power/area scaling characteristics are superior in comparison to most other interconnection networks like full-crossbar or omega network.

The network consists of multiple local BNs. Each local BN is used to connect the word-lines with the same relative positions in different groups. For instance in Figure 3.5, all of the four groups have their 2^{nd} lines connected by a local BN. There have to be as many interleaved local BNs as there are lines in a single group. The set of groups con-

nected by a single local BN is called the swapping set, and the size of this swapping set ($2^{num.\ of\ swapping\ levels} = 2^2$ in the example shown) is determined by the depth of the network. Given the full permutation and non-blocking properties of the chosen interconnection network, lines in the same relative position can be swapped between the different logical groups. Increasing the depth of the BN widens the scope of line swapping, however, it also imposes higher overheads on the underlying cache. In order to minimize these overheads and reach to a network with higher depth, we employ an efficient circuit-level implementation of a BN which is presented in [86]. A memory hash-table can also be used as an alternative here. However, since this network is in the critical path of the cache access, we employ a BN which provides inherent flexibility, lower delay, and lower power consumption.

Network Configuration Storage: The interconnection network configuration is kept in the network configuration storage. According to our evaluations in the next section, a small fraction of the manufacturing test time can be used to solve the configuration and mapping problems. For the network configuration storage, we use a low-voltage *on-chip* NOR-flash described in [93]. However, since this structure is extremely small (mostly less than 400 bytes), employing other non-volatile memories (e.g. fuse, or EEPROM) has negligible impact on our results.

Fault Map Array: The fault map array has the same number of rows as the spare cache. For each redundancy unit in a spare cache row, the fault map array stores the row number in the corresponding logical group which utilizes that redundancy. For example, if a broken data chunk in the 5th row of an eight-rows logical group should be replaced by its corresponding spare unit, the fault map saves 101 for that redundancy unit. This implies,

that for a very small granularity of redundancy, the length of the word-lines in the fault map array can be significantly longer than the main cache. The access time of the fault map array is comparable to the L1 cache. Hence, this structure should be accessed in parallel with the tag array access for the L1. Conversely, for the L2 cache, the access to this structure happens after the hit resolution from the tag side, resulting in a significant reduction in the dynamic access energy. In contrast to the network configuration storage, which should be filled during the manufacturing test time, the fault map gets its contents directly from the built-in self test (BIST) module during the first boot of the system. Further, its content can be saved on the hard-disk and retrieved during the machine boot up. This mechanism works properly for the fault map since the BN routes have already been fixed. And during the testing operation by BIST, the effect of line swapping will be automatically accounted for.

Comparison Stage: This stage compares the least several significant bits of the set segment of the address[1] with the returned content of the fault map array to determine whether that unit of redundancy replaces the data chunk from the main cache.

MUXing Level: At the end of the access critical path, based on the results of the comparison stage, the MUXing level determines for each redundant unit whether the main cache or the spare cache data is valid and drives that onto the cache output. Word-lines in the main/spare cache are divided into equal units of redundancy. The size of these redundancy units specifies the MUXing granularity. On the other hand, since the read and write are symmetric operations, the only modification in the implementation would be to replace the MUXes with pass transistors.

[1]The number of bits depends on the number of word-lines in each cache logical group.

In order to guarantee the proper operation of the on-chip caches, we assume all the main SRAM structures in the ZC architecture (i.e. main cache, spare cache, tag array, and fault map) would be affected by the process variation. In our design, the main and spare caches are the major contributors to the ZC area and potential failures in these structures are directly handled by our scheme. In order to protect the fault map and tag array, we employ a process variation tolerant 8T SRAM cell which is more area efficient than simple transistor sizing [30, 29, 98]. However, it should be noted that the 8T cell comes with around 36% area overhead and is not cost effective for protecting the entire cache (Section 3.6).

3.2.2 Hard-Fault Detection

In this work, we use ZC to tackle process variation as well as wearout induced failures. Here, we separately discuss the mechanisms required to detect these two type of hard-faults. First, we focus on the detection process that is needed to detect process variation induced failures. Since these hard-faults are present at manufacturing time, the standard manufacturing testing process can be employed. In order to test on-chip caches, normally, a combination of automatic test pattern generation (*ATPG*) and the BIST-based testing is used to generate the test vectors, apply them, and produce the compact signature from the test results. Different versions of the MARCH test, introduced by the research community and industry, are the most common type of test patterns for testing memory structures and are widely used for testing SRAM structures against stuck-at and bridging faults.

However, wearout induced failures manifest during the lifetime of the system and needs special treatment. Two type of approaches have been proposed for this purpose. The conventional approach is periodic testing in which the system periodically suspends its normal

operation and allows the BIST module to test the on-chip caches. In order to guarantee the correct operation of the system, the architectural or microarchitectural state of the system needs to be checkpointed right after each testing interval. Moreover, to reduce the fault detection latency and this checkpointing cost, mostly in terms of main memory or cache usage, this type of testing needs to be done frequently. The other alternative is the continuous testing which is the subject of many current research works. In this type of testing, as soon as a faulty cell gets used, the detection mechanism informs the system of the location of the failure. Continuous testing can have many forms. One of the simplest is error detection codes which are widely used in memory structures. Another well-known proposal is the redundant execution that needs to continuously check the consistency among multiple copies of the same data. Another means for continuous detection is through sensors that can estimate the amount of device level wearout.

3.2.3 ZC Configuration

Proper configuration of the ZC is crucial for achieving higher utilization of the spare elements. The first step toward this is to determine the input/output mapping for the BNs. In other words, logically group together the cache lines that share a single spare line. We model this as a graph coloring problem that can be solved during the manufacturing test time. The solution to the coloring problem provides the required BN configuration information which is saved in the network configuration storage. On the first boot up of the machine, the BIST module takes advantage of the already configured BNs to find the faulty SRAM cells. The fault map array is then populated by the BIST module based on the location of the faulty cells in the main/spare caches. In order to achieve an effective

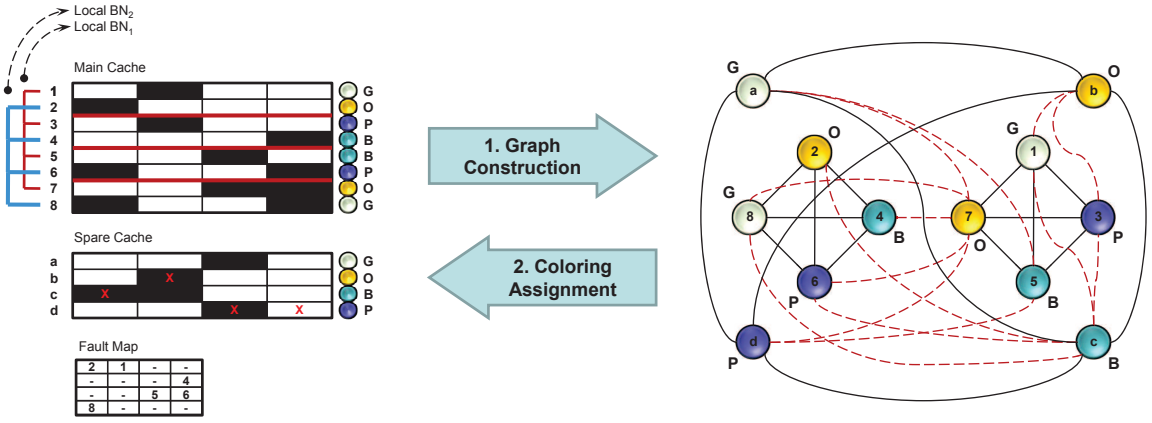


Figure 3.6: Mapping between the graph coloring problem and the defect pattern in the main/spare caches. The solid edges stand for the intrinsic conflicts between the word-lines. The dotted edges correspond to the word-line conflicts due to the defect pattern. An "X" indicates a collision using a default grouping. Numbers written in the fault map indicate the corresponding cache word-lines to which the spare units are assigned. (G=Green, B=Blue, P=Purple, O=Orange)

line swapping capability in ZC architecture, two major algorithmic problems need to be addressed here: 1) Effective group formation, 2) Benes network configuration.

3.2.3.1 Effective Group Formation

The problem of determining the logical groups that share a single spare line in the ZC architecture is modeled as a graph coloring problem. Figure 3.6 is an example that illustrates the process of mapping the defects in the main/spare cache to a graph. In the cache arrays of Figure 3.6, each black box stands for a faulty cell. An 8-line cache is divided into 4 logical groups and a spare line is assigned to each logical group. For example, lines 1 and 2 in the main cache form a logical group that utilizes line 'a' in the spare cache. Two local BNs are required to do the proper shuffling. The first (second) lines from different logical groups can swap their positions using the corresponding local BN (e.g., lines 1, 3,

5, and 7 can swap their positions).

The graph on the right hand side of the figure is constructed based on the defect pattern in the main/spare caches. Each node in this graph represents a line in the main/spare cache. Whereas, the edges represent a conflict between a pair of lines, i.e., the two nodes connected by an edge represent lines that cannot be in the same logical group. A graph coloring algorithm can now be applied to this graph to find a solution such that neighboring nodes are not assigned the same color. Thus, after coloring, nodes with the same color are guaranteed to have no edges between them implying that the corresponding cache lines have no conflicts between them. Cache lines with the same color thereby form a logical group. The graph edges that represent conflicts between the lines can be broadly divided into two categories:

1. Intrinsic Edges: Each of the lines in the spare cache is dedicated to a single logical group in the main cache. This implies that spare lines cannot be in the same logical group. As a result, 4 nodes (a, b, c, d) construct a complete sub-graph (Figure 3.6). Moreover, the structure of the BN forces the lines connected to a local BN into different logical groups. For example, lines 1, 3, 5, and 7 can not be in the same group. Consequently, these 4 word-lines also form a complete sub-graph.

2. Defect Edges: The defect pattern in the main/spare cache introduces other edges in this graph. These defect edges connect the pair of lines that have at least one conflict (for the same data chunk). For instance, there is a defect edge between the nodes *3* and *c*, because both have their second data chunks faulty.

A graph coloring problem is solvable for a graph G and an integer $K \geq 0$, if the nodes of G can be colored with K colors such that no edge exists between the same colored

nodes. For our problem instance, we want to show that if there are *h* logical groups in the cache, and the nodes can be colored with at most *h* colors, then there would be a feasible configuration for the BNs such that the ZC works properly. But since there is always a complete sub-graph in the problem graph with *h* nodes (due to the intrinsic edges), the chromatic number is at least *h*. On the other hand, our problem constraint dictates that we can use at most *h* colors for the graph coloring problem. Hence, the graph coloring problem for the ZC configuration should have a solution with exactly *h* colors. A valid coloring assignment indicates no collision between the lines within each logical group and replacement of the defective data chunks can be properly handled.

Graph coloring is widely recognized as NP-complete. Thus, for solving it, we use an approximate algorithm called Incomplete Backtracking Sequential Coloring (IBSC) [55]. IBSC is a heavily optimized version of the full backtracking solution. It restricts the branching factor on each level to expedite the process of finding the approximate chromatic number. On an average, IBSC only increases the chromatic number of the graph by 5.2%, which is considerably better than the theoretical upper bound that we used for the analysis in Section 3.3. The complexity of the IBSC algorithm is $O(|V|^4)$ and the actual runtime is discussed in the next section. Furthermore, this algorithm can easily be converted to the exact solution by eliminating the branching heuristic. It is especially useful in the case of small graphs or when more computational power/time can be devoted to the solver.

The graph coloring solution determines the assignment of lines to logical groups. All the lines in the main/spare cache with the same color form a single logical group. For example, all the lines with the orange color are bound to the orange spare row using the corresponding local BNs. Figure 3.6 illustrates a valid coloring assignment. With this col-

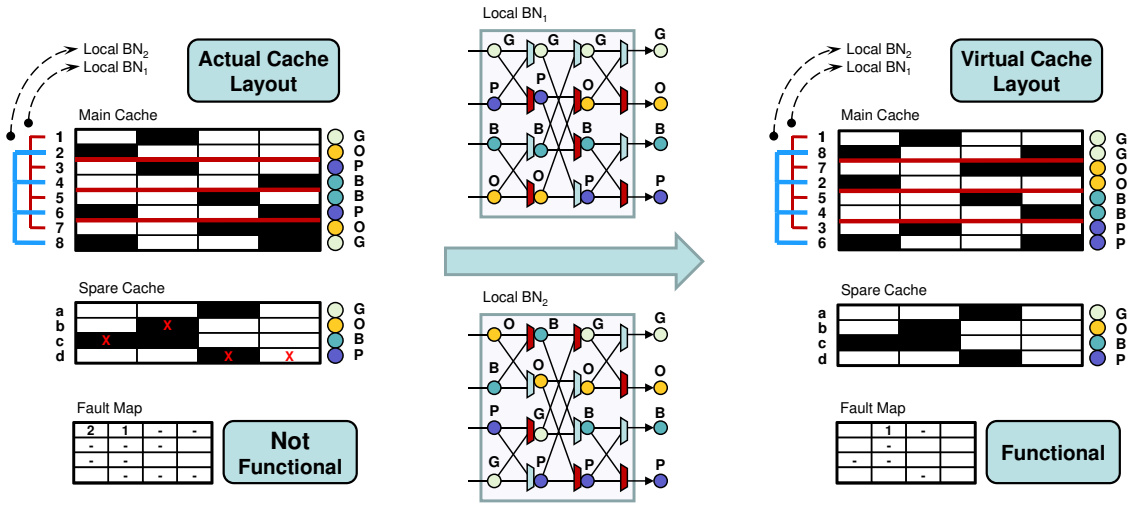


Figure 3.7: Proper configuration of two BNs that transform the actual cache layout (left) to the virtual one (right) for the given coloring assignment. The upper (lower) BN connects the first (second) rows of the 4 logical groups. The darker 2-input MUXes are configured to output their lower input while the lighter MUXes output their upper input. (G=Green, B=Blue, P=Purple, O=Orange)

oring assignment in place, the logical groups formation is complete for the main cache. The next step is to solve the BN configuration problem in order to make the cache functional.

3.2.3.2 Benes Network Configuration

The BN is non-blocking and also allows any permutation of the inputs to be mapped to the outputs. In Figure 3.7, the left cache structure shows the physical cache layout with a solution for the graph coloring problem. As described, the color of each line determines the logical group to which the line is assigned. As a result, the position of each particular line after the line swapping is apparent. For instance, in the Figure 3.7, the last cache row has a green color which corresponds to the first row of the spare cache. This denotes that the last cache row should be mapped to the second row of the first logical group. The line ordering for the virtual cache layout on the right hand side can be obtained from the

Table 3.1: The target system configuration

Parameters	Value
Frequency	4 GHz
L1 Caches	64KB data and 64KB instruction, 2-way set associative, 2 cycles hit latency, 32B block size
L2 Cache	2 banks 2MB Unified, 16-way set associative, 12 cycles hit latency, 128B block size
Registers	128 integer, 128 floating point
ROB (re-ordering buffer)	128 entries
LSQ (load/store queue)	64 entries
Instruction fetch buffer	32 instructions
Issue width	4
FU (functional unit)	4 int ALU, 1 int mult/div, 2 memory system ports
FPU (floating point unit)	4 FP ALU, 1 FP mult/div
Main memory	250 cycle latency, 16 bytes with 10-cycle latency
Branch predictor	combined (bimodal and 2-level)
BHT (branch history table)	4096 entries
RAS (return address stack)	32 entries
BTB (branch target buffer)	512 entries, 8-way associative

physical layout by employing two 3-layer deep local BNs. The BNs have to be properly configured, by determining the select signals for the MUXes within the BN, to achieve this re-ordering. Having the desirable permutation between the inputs/outputs of the BNs, we employ the recursive method described in [105] to configure the network. Since an n-input BN is constructed from two identical $\frac{n}{2}$-input sub-networks, the configuration can be computed recursively in $O(n^2)$.

3.3 Design Space Exploration

The process of finding suitable design points for L1/L2 ZCs involves fixing the architectural parameters. The high level architectural parameters used for this exploration are listed in Table 3.1. In addition, there are three main parameters specific to the ZC design: a) size of the spare cache, b) depth of the BN, and c) the MUXing granularity for the redundant data chunks. In this section, we sweep a wide range of values for these parameters

and study the overhead of each design point. The number of spare cache lines was taken from the set $\{2^i \mid i \in \{0, 1, ..., 7\}\}$. Note that the length of the word-lines is the same for the main and spare caches. The depth of the BN is selected from the set $\{1, 3, 5, ..., 19\}$ and the MUXing granularity is selected from the set $\{2^i \mid i \in \{0, 1, ..., 10\}\}$ bits. In total, considering both the L1 and L2 caches, there are 1760 points in the design space. In order to prune this design space, a number of practical design constraints were considered. For instance, designs with more than 128 spare lines were not studied due to their significantly high area/power/delay overhead. Detailed discussions of these practical constraints and their impact on the design space follows. Finally, we pick suitable configurations for L1/L2 ZCs.

1) Graph Coloring Solver Time: A deeper BN can provide a wider range of cache line swapping, thereby improving ZC defect tolerance. However, a deeper network also increases the complexity of graph coloring and BN configuration. Out of these two, the runtime of the graph coloring problem is by far the dominating factor. Figure 3.8 depicts the relationship between the size of the graph-coloring problem and the time required to solve it using the IBSC algorithm (Section 3.2.3). We ran the solver on a single core Pentium-4 processor with 1*GB* of memory capacity and 2*GHz* clock rate. The Y-axis in this plot is logarithmic. It demonstrates the fast growth in the runtime of the solver with the increase in the problem size.

The total manufacturing test time for a high-end processor (considering the functional, structural, wafer, and packaging tests) is around a few minutes [96, 61]. Using this as a reference, we limit the graph solver time to use a maximum of 10 seconds for all four on-chip cache structures (L1-D, L1-I and two banks of L2). For the case when the cache has 1

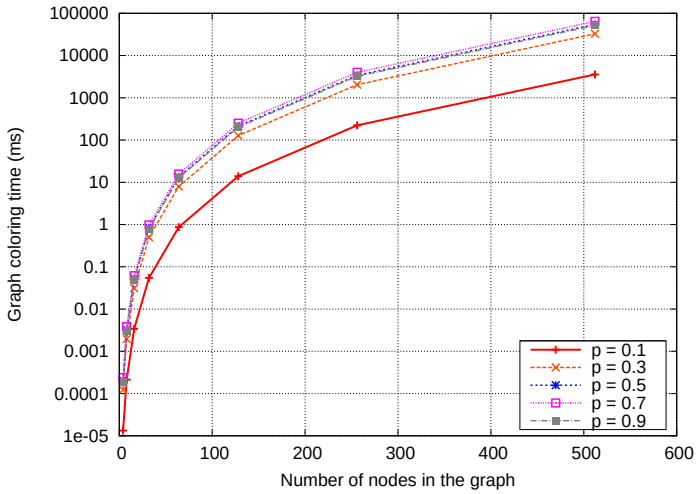


Figure 3.8: The run-time of the IBSC graph coloring solver in *ms* for different edge densities and number of nodes in the graph. In this figure, *p* is the edge density which is defined as the probability of having an edge between an arbitrary pair of nodes in a random graph *G(n,p)*.

spare word-line for every k word-lines and the depth of the BN is $2b-1$, the total number of nodes in the graph coloring problem would be $(k+1)2^b$. According to Figure 3.8 and based on our solver time budget, ZCs can use BNs that are up to 9 levels deep and can connect 32 logical groups together. For instance, if each logical group consists of eight word-lines, there would be $32 \times (8+1) = 288$ nodes in the graph coloring problem. There are numerous works [39] on the parallel graph coloring algorithms that can be used for decreasing the solver runtime and potentially increasing the allowable depth of the ZC. However, scrutinizing them is beyond the scope of this thesis. As a side note, in order to get a feeling about size of the network, we look at the transistor count here. A 9-level deep BN has less than 37K transistors while a 64K L1 cache, which is much smaller than L2, has more than 3M transistors.

There is a less than 4% chance that the solver does not find a feasible coloring assignment due to limitations on the time budget or the inherent complexity of the collision

pattern (Section 3.4). Using a deeper BN, longer time budget for the solver, finer granularity of MUXing, or a larger spare cache can further reduce this small chance. Nevertheless, if such a situation does arise, we can either resize the cache or simply reject it. Block/way disabling techniques [74] can also be applied at the position of the faulty cell to preserve correct functionality. Note that the scenarios where ZCs need to resort to such methods are very rare.

2) Probability of Operation : The probability of operation (P_{op}) is a definitive metric for a reliable system. We calculate the probability that a specific ZC architecture can properly operate for a given P_F and use the results to further prune our design space. The graph, which was generated in Section 3.2.3, represents an instance of a defective cache. For the sake of this study, defective caches are modeled as random graphs *G(n,p)* since SRAM cell defects occur as random events. These random defects are due to the major contribution of the random dopant fluctuation to the process variation [5]. Here, *n* is the number of nodes and *p* is the probability of having an edge between an arbitrary pair of nodes.

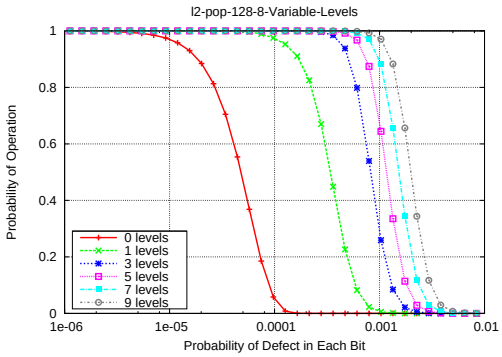
The next step is to estimate the graph coloring solution for these graphs. Calculating the average upper bound of the chromatic number for a random graph is a challenging problem in graph theory [17]. We use two different proposed upper bounds to evaluate the P_{op} of ZCs for a given number of failures. The same set of input conditions are required for both of these upper bounds, which were derived by Achlioptas [2, 3] and Bollobas [22]. The proposed upper bound by Bollobas (B) works better for smaller values of *p* and the Achlipotas bound (A) is mostly applicable for the larger values of *p*. Thus, we used the weighted average of these two bounds based on the *p* value (e.g., $pA + (1 - p)B$). Approximation algorithms used to derive these upper bounds, have a significantly poorer approximation

factor compared to the IBSC algorithm used in Section 3.2.3 [65, 102]. The edge probability factor *p* is defined as the ratio of the expected number of edges in the graph to the number of edges in K_n (a complete graph with *n* nodes). The expected number of edges in a randomly constructed graph can be calculated by accounting for the intrinsic and fault edges. In other words, $E_{edges} = E_{Intrinsic} + E_{Fault}$ where:

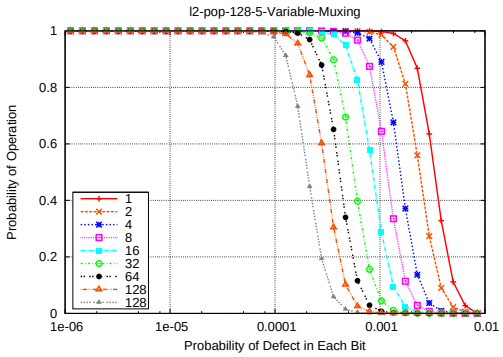
$$\begin{aligned} E_{Intrinsic} &= m \binom{u}{2} + \binom{u}{2} \qquad \text{and} \\ E_{Fault} &= m \times u^2 \times \left[1 - (1 - \alpha_1\alpha_2)^t\right] \\ &+ \left[\binom{u \times m}{2} - m \binom{u}{2}\right] \times \left[1 - (1 - \alpha_2^2)^t\right] \end{aligned}$$

Here, *m* is the number of word-lines in each logical group, *u* is the number of logical groups in a swapping set, *b* is the MUXing granularity, *t* is the number of redundancy units in each word-line, *n* is number of swapping sets, p_1 is P_F for the main cache, and p_2 is P_F for the spare cache. Here, $\alpha_i = 1 - (1 - p_i)^b$ shows the probability of having at least one failure in *b* bits.

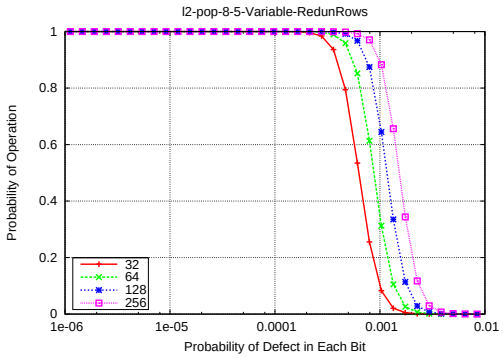
Figure 3.9 shows the P_{op} of an L2 ZC with 128 spare word-lines, MUXing granularity of 8 bits, and 5 levels of swapping. In each of the sub-figures, two of the parameters are fixed and the third one gets the values from the original sweeping set. Notice that in Figure 3.9a, adding the first few levels of line swapping significantly increases the robustness of the cache, but beyond 3 levels, adding more levels has a diminishing return. Since the weighted average of the two bounds is not an integer number, we employed a semi-sigmoid function, which is a sigmoid function fitted to the shifted step function (*Number*



(a) The effect of changing the number of swapping levels while using 128 spare word-lines and MUXing granularity of 8 bits.



(b) The effect of changing the MUXing granularity while using 128 spare word-lines and five levels of swapping.



(c) The effect of changing the number of redundant rows while using MUXing granularity of 8 bits and five levels of swapping.

Figure 3.9: P_{op} of L2 ZC for different P_F while fixing two parameters and allowing the third one to vary.

of Logical Groups + 0.5), for mapping the calculated chromatic number to P_{op}. Using this semi-sigmoid function, if the calculated chromatic number is smaller than the number of available logical groups in the swapping set, the graph is colorable with a probability close to one and if the derived chromatic number is one unit larger than the number of logical groups, the probability would be close to 0. As shown in [4], P_F in 45*nm* can be as high as 10^{-3}. Based on this fact, we pick the design points from our design space that have $P_{op} > 90\%$ for $P_F = 10^{-3}$.

3) Area and Power Overheads: Based on the limiting factors that we have proposed,

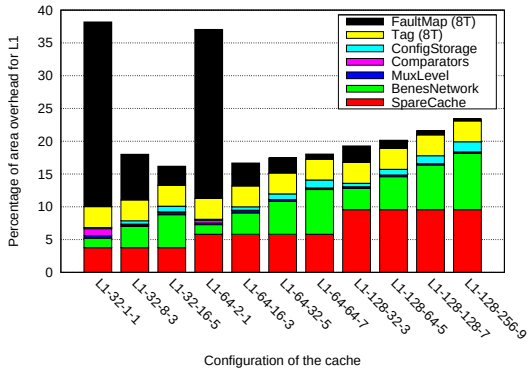
the size of the design space shrinks from the 1760 starting points down to 103 points. The next factor for eliminating the points is a one-by-one comparison. Given a design point (L_1, M_1, D_1) with L_1 spare word-lines, MUXing granularity of M_1, and a D_1 deep BN and another design point (L_2, M_2, D_2), we can exclude the first point from the design space if it is inferior in all dimensions:

$$L_1 \geq L_2 \ , \ M_1 \leq M_2 \ , \ D_1 \geq D_2$$

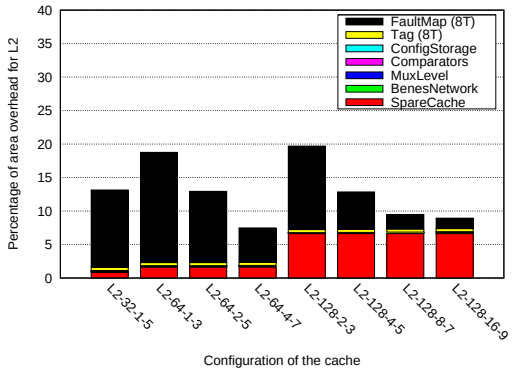
This is equivalent to removing dominated points in the Pareto space with dimensions L_1, M_1, D_1. This step reduces the design space to 11 points for L1 and 8 points for L2.

To evaluate our designs, we used CACTI 6.0 [72] for evaluating the area, leakage power, and the dynamic energy for the SRAM structures. The Synopsys tool-chain was employed for evaluating area, timing, leakage power, and dynamic energy of the non-SRAM parts. All designs are evaluated in 45*nm*.

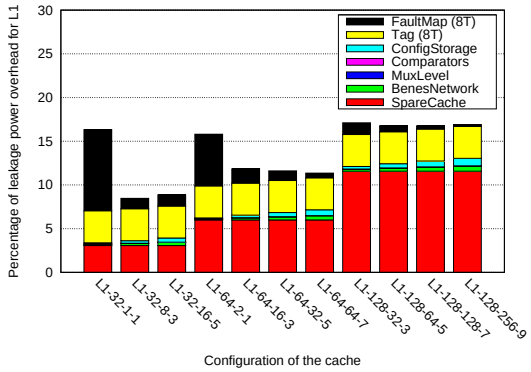
Figure 3.10 shows the area, leakage power, and dynamic energy overhead of the selected points in the design space. For instance, L1-32-8-3 stands for an L1 design point with 32 redundant rows, MUXing granularity of 8 bits, and BNs of depth 3. It is notable that increasing the size of the spare cache does not always lead to an increase in the area of the L2 ZC because the fault map size is reduced. However, due to the longer L2 word-line, a finer MUXing resolution is required which results in a relatively larger fault map array for L2 compared to L1. The dynamic energy overhead for the L1 ZC was mostly higher compared to the L2 ZC. There are two reason behind this: 1) The L1 cache accesses the fault map/spare cache in parallel to the main cache and 2) The L1 cache reads the entire set



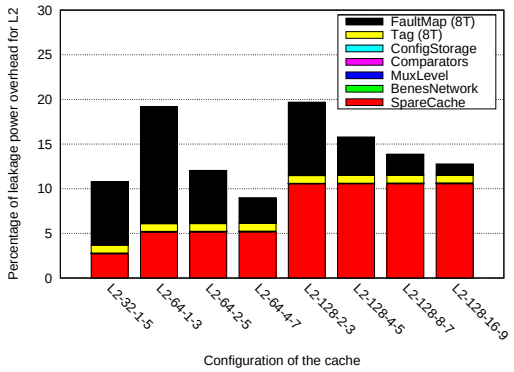
(a) Percentage of area overhead (L1s)



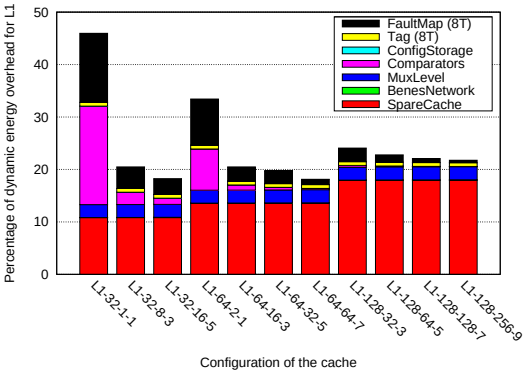
(b) Percentage of area overhead (L2s)



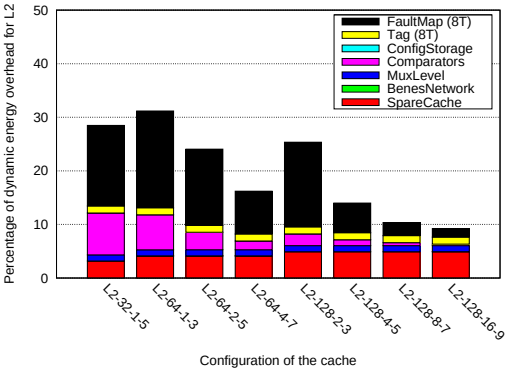
(c) Percentage of static power overhead (L1s)



(d) Percentage of static power overhead (L2s)



(e) Percentage of dynamic energy overhead (L1s)



(f) Percentage of dynamic energy overhead (L2s)

Figure 3.10: Area, power, and energy overhead of the potential L1/L2 ZCs which are stated in percentage.

for every access whereas the L2 cache is able to read just the right cache block because the tag and data are accessed sequentially.

4) Cache Access Latency: The increase in the BN depth also has a direct impact on

the cache access time. Since on-chip caches are essential for the performance of modern processors, we assume no slack is available on the access time of the caches. Therefore, any minor modification in the base caches results in at least one extra cycle access penalty. Nonetheless, in the case that considerable slack is available, a design with narrow BN can be leveraged for avoiding any additional cycle latency. In our design, the MUXing level and BN are on the critical path of cache accesses. Based on the timing analysis of our design, in Figure 3.10, design points with BN depth less than or equal to 7 need one extra cycle latency for the cache access while others (i.e. BN depth = 9) require 2 extra cycles. In Section 3.6, we evaluate the performance drop-off due to the additional access latency of the L1/L2 ZCs.

Considering the design points in Figure 3.10, we select L1-32-16-5 as the L1 ZC which imposes 16% area, 9% static power, and 19% dynamic energy overhead over the baseline L1 cache. For the L2 ZC, L2-64-4-7 is selected which imposes 8% area, 9% static power, and 16% dynamic energy overhead compared to the baseline L2 cache. These two selected configurations represent a good trade-off between all the design objectives. However, based on a particular optimization criteria, another design point might work better. For instance, if static power is the main concern, the optimal design point for L1 switches to L1-32-8-3.

3.4 Yield Analysis

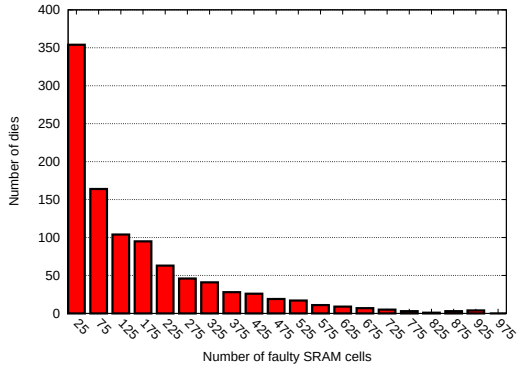
In this section, we go through the process of manufacturing yield calculation for a population of ZC enabled chips. A population of 1000 chips was generated from the selected ZC configurations for this purpose. We account for both, inter-die (die to die (D2D))

and intra-die (within die (WID)), components of the process variation. VARIUS [82] is leveraged to model systematic, D2D, and module level intra-die variations. Each chip is considered as a composition of 8 SRAM structures: L1-Data, L1-Inst, two L2 banks, and the corresponding spare caches. The $\frac{\delta_{V_{th}}}{V_{th}}$ is set to 12.5% which is the projected Systematic + D2D variation for 45*nm* technology [5].

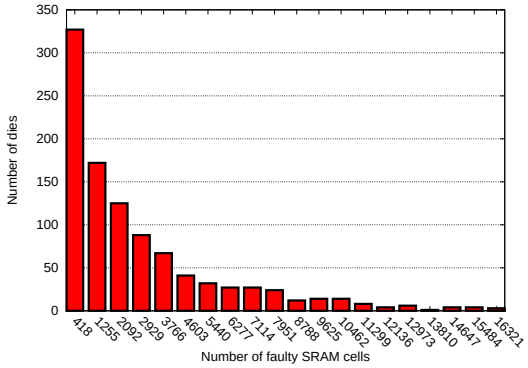
Having all the high level variation models in place, a two-step approach is used to derive the number of faulty cells in each SRAM array for an arbitrary chip in the population: 1) We take the intra-module variation model from [5] with $\delta_{V_{th}} = 30mV$. Using this model, the nominal value of P_F across each module is derived from the data provided in [71] based on the average shift in V_{th} for that module. 2) The clustering effect, which determines the degree of defect dispersal in the cache structures, is also modeled. Due to the high density of SRAM structures, the clustering effect has a significant impact on the arrangement of the defects in the corresponding SRAM arrays. We account for it by employing the large-area clustering negative binomial model [56] which is based on the well-known negative binomial yield formula.

Figure 3.11 illustrates the distributions of the 1000 generated chips based on the number of faulty SRAM cells in their L1/L2 caches. For instance, as Figure 3.11a shows, around 100 of the chips have 100 to 150 faulty SRAM cells in their L1 cache. These derived distributions are consistent with the ones in [4]. It is interesting to note that, in the case with no protection scheme for the cache, the yield for 45*nm* technology could be as low as 33%.

Manufacturing yield is defined as the fraction of fully functional chips to the total number of manufactured ones. This value can be interpreted as the probability of operation for a



(a) Distribution of the generated chips based on the number of faulty SRAM cells in their L1 cache



(b) Distribution of the generated chips based on the number of faulty SRAM cells in their L2 cache

Figure 3.11: Distribution of generated chips by the number of faulty SRAM cells in their L1/L2 caches. A population of 1000 chips is generated by considering the large-area clustering effect, intra-die, inter-die, systematic, and parametric variations.

particular chip after the manufacturing process (Y_{chip}). We define *CW* and C_i as events that express the proper functionality of a manufactured chip and the existence of *i* faulty cells in a chip, respectively. In the following equations, N_{tot} is the total number of manufactured chips, N_i is the number of the chips with *i* faulty cells, and N_{cells} is the total number of SRAM cells. Based on the rules of probability:

$$\begin{aligned} Pr(CW) &= \sum_{i=0}^{N_{cells}} Pr(CW \cap C_i) \\ &= \sum_{i=0}^{N_{cells}} Pr(CW|C_i) \times Pr(C_i) \\ &= \frac{1}{N_{tot}} \sum_{i=0}^{N_{cells}} Pr(CW|C_i) \times N_i \end{aligned}$$

Since we consider an independence between the P_F of L1 and L2 caches, as shown in [47], the yield of a chip can be written as:

$$Yield_{chip} = \prod_{i \in chip\ modules} Yield_i \tag{3.1}$$

As a result, $Pr(CW)$ can be written for each cache separately. Equation 3.1 is used to calculate the chip yield in each case. Here, $Pr(CW|C_i)$ is the probability of having a functional cache given that it contains *i* faulty cells and can be written as:

$$Pr(T, FM, MC, SC|T_{i_1}, FM_{i_2}, MC_{i_3}, SC_{i_4}) = Pr(T|T_{i_1})$$
$$\times Pr(FM|FM_{i_2}) \times Pr(MC, SC|T, FM, MC_{i_3}, SC_{i_4})$$

where $i_1 + i_2 + i_3 + i_4 = i$. In this equation, ***T/FM/MC/SC*** are the events that the tag/fault map/main cache/spare cache arrays work properly. Similar to the previous equations, T_{i_1} is the event of having i_1 faulty cells in the tag array. For the fault map and tag array, we assume 8T cells guarantee the fault-free operation of these relatively small structures (i.e. $Pr(FM|FM_{i_2}) = 1$ and $Pr(T|T_{i_1}) = 1$). Finally, calculation of the last term is discussed in Section 3.3.

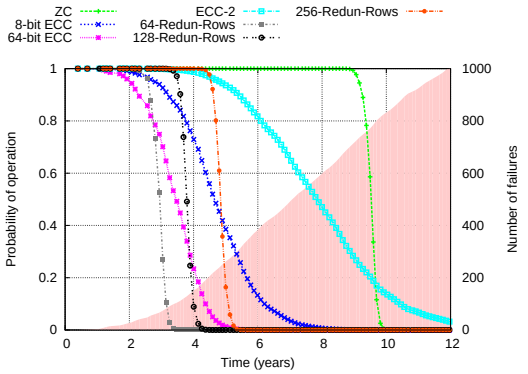
Given the population of 1000 (N_{tot}) generated chips, N_i for each of the cache structures is known using the mentioned modeling. Yields of the L1 cache and each L2 bank are calculated through the described methodology. The derived yield for the L1 ZC and each bank of L2 ZC are 98.8% and 98.2%, respectively. This implies 96.4% yield for the L2 ZC.

3.5 Wearout Tolerance

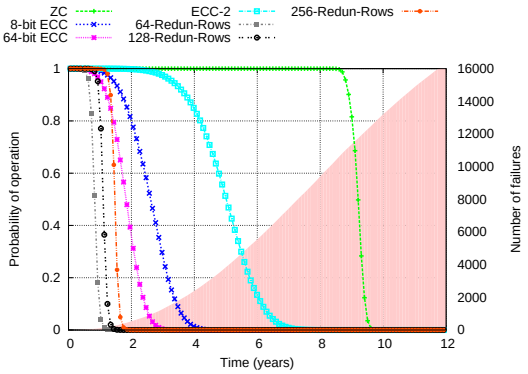
In this section, we expand the potential functionality of our proposed cache architecture which is mainly designed to tackle the process variation concerns in deep sub-micron

technologies. Wearout, in contrast to process variation, is a gradual process. Supposing that the lifetime of a cache can be extended to 10 years and it will experience twenty thousand cell failures during this period, the mean time between failure (MTBF) would be 43.8 hours – worst case scenario. Given the fact that the required time for solving the graph coloring and BN configuration problems is only several seconds, our method can be easily applied to reconfigure the cache after detection of each failure. As another notable point, on-chip flash could also be replaced with latches in this case since the configuration problems would not need to be solved during the manufacturing test time. Nonetheless, even if on-chip flash was deployed, the write-cycle limitations of a flash cell ($\sim$100K) is still higher than the maximum number of required reconfigurations. As a result, it allows us to reconfigure the BN more than 16000 times – that is the average upper-bound for the number of tolerable faults based on the L1/L2 ZCs selected in Section 3.3. We measured the area/power overhead of replacing on-chip flash with latches. Although negligible for L2 cache, the area overhead of the BN configuration storage for L1 is still noticeable and increases the overhead of our selected L1 ZC from 16% to 18%. Moreover, in this scenario, the configuration information could be stored in the hard-disk after it is obtained. When a system is powered up this information is retrieved from the hard-disk and moved to the network configuration storage.

Upon the detection of a new fault, since that is the only outstanding faulty cell, block / way [74] disabling techniques can be applied based on the position of the faulty cell to preserve correct functionality of the underlying cache. Removing this faulty cell from the functional space of the cache enables ZC to use the rest of the available cache space to solve the configuration problems. A simple cache disabling mechanism is to only use the



(a) Monte Carlo life-time simulation results for L1 cache



(b) Monte Carlo life-time simulation results for L2 cache

Figure 3.12: Results of Monte Carlo lifetime simulation which show the probability of operation for L1/L2 caches protected by different mechanisms. In addition, the shaded region shows the expected number of failures over the life-time.

half part of the cache which does not contain the faulty cell and disable the other half. After ZC reconfiguration is performed, the whole cache space would be functional.

A Monte Carlo engine is employed to study the P_{op} for ZCs over their life-time. In each iteration of the Monte Carlo simulation, time to failure (TTF) for each SRAM cell in various array structures is calculated using a Weibull distribution with a nominal mean of 100 years – as the expected lifetime of an individual cell [91]. Hundreds of such iterations are run during the entire simulation. The simulation results are shown in Figure 3.12 for the L1/L2 ZCs selected in Section 3.3 and several other conventional protection mechanisms.

One advantage of ZC over the conventional protection mechanisms is its ability to equalize the lifetimes of L1 and L2 caches. This implies the proper relative provisioning of the caches against hard faults. As a result, ZCs maximize the utilization of the entire provisioned spare elements. The shaded region in these figures depicts the cumulative distribution function (CDF) of the combined MTTF Weibull distributions for the main/spare caches. For instance, in Figure 3.12a, there would be $\sim$400 faulty cells in the L1 related

SRAM structures after 6 years. As can be seen, the selected ZC architectures prolong the functional lifetime of the caches up to 10 years. Furthermore, P_{op} has a graceful degradation for the ECC methods compared to the sharp drop-off for ZC and row/column redundancy. Consequently, there is a significant chance for the ECC protected cache to break early in the life-time. This makes them an inappropriate choice even when a long life is not expected. Two bit correction ECC (DECTED), on the other hand, needs 14 extra bits for each 64-bit of data which is $\sim$22% overhead only for keeping the error correction bits. In terms of the energy overhead imposed by the encoder/decoder per access, as shown in [53], around 50% should be expected.

3.6 Comparison and Discussion

To demonstrate the efficiency of our design, we compare ZC with conventional and recently proposed methods in this section. As representatives for the ZC architecture, we pick the L1-32-16-5 configuration as the L1 ZC (16% area overhead and 99% yield) and L2-64-4-7 configuration as the L2 ZC (8% area overhead and 96% yield).

3.6.1 Comparison with Conventional Techniques

Figure 3.13 demonstrates the amount of area overhead required to protect the L1/L2 caches using different protection schemes. For a given probability of failure, we started with the least possible overhead for every mechanism and gradually increased the area overhead until the P_{op} reaches 90%. An infinity symbol (∞) on the top of a bar indicates that achieving $P_{op} > 90\%$ is not possible for the corresponding protection mechanism. This

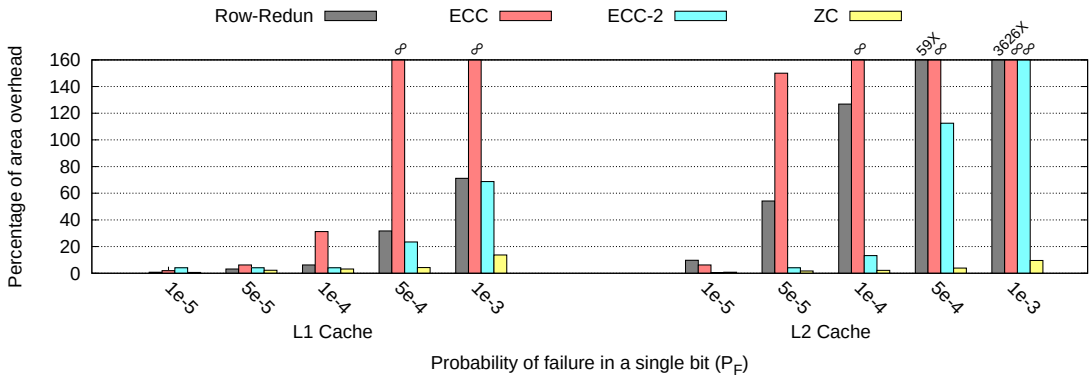


Figure 3.13: Area overhead of the different protection mechanisms for tolerating a given P_F. In this figure, Row-Redun stands for the row redundancy protection scheme. ECC and ECC-2 are the 1-bit and 2-bit error correction schemes, respectively.

figure only accounts for the amount of *redundancy* required by SECDED (ECC), DECTED (ECC-2), and row-redundancy methods while considering the complete overheads for ZC modifications. In other words, hardware overhead for encoder/decoder is not considered for ECC/ECC-2. Similarly, the decoder augmentation is not included in the area overhead of the row-redundancy protection method.

Row-redundancy can protect any cache with inefficient usage of the redundant elements. Nevertheless, as it is shown in [46], row-redundancy with more than 10 extra rows is not efficient due to the considerable increase in the row decoder latency. As shown in this figure, the area overhead of ZC is significantly smaller compared to even the 2 bit error correction scheme (ECC-2) which has a significant power and area overhead for decoding/encoding. Going beyond 2 bit correction using ECC codes is extremely expensive in terms of the code storage area, decoding/encoding power and delay [54]. On the other hand, single bit correction ECC cannot even protect the cache structures with $P_F > 10^{-4}$. For L2, the difference between ZC and other protection mechanisms is even more noticeable because of the longer word-line and larger cache size that challenge the other protection

mechanisms. In terms of the energy consumption, ECC and ECC-2 impose around 25% and 50% overheads, respectively [54]. Whereas, both of the selected L1/L2 ZCs have less than 20% energy overhead (Section 3.3). Hence, it should be clear that the conventional soft-error cache protection schemes cannot deal with the high degree of process-variation in deep nanometer technologies.

3.6.2 Comparison with Recently Proposed Techniques

More recent proposals target high defect density scenarios that are challenging if not impossible for conventional schemes. Here, we compare ZC with three of these recently proposed cache reliability schemes that target failure rates close to ours. For the purpose of comparison, we measure the performance drop-off for a system (Table 3.1) equipped with the selected L1/L2 ZCs in Section 3.3. A performance loss is expected due to the extra cycle of latency added to both L1 and L2 ZC designs. We used the SimpleScalar [14] out-of-order simulator along with the SPEC-FP-2000 (171.swim, 172.mgrid, 173.applu, 177.mesa, 179.art, 183.equake, 188.ammp) and the SPEC-INT-2000 (164.gzip, 175.vpr, 176.gcc, 181.mcf, 197.parser, 255.vortex, 256.bzip2) benchmarks. On average, a 3.2% performance drop-off is observed, with maximum of 6.9% for 197.parser and minimum of 0.1% for 176.gcc.

Agarwal [5] proposed a fault-tolerant *direct-mapped L1* cache that uses cache block remapping to preserve correct functionality under the process variation in 45*nm*. As Figure 3.1a depicts, around 23% of the cache blocks expected to be faulty in this technology. This method maps faulty blocks to the neighboring functional blocks in the same word-line, which forces the L1 to access L2 for getting the values of these blocks. This method is

only applicable to direct-mapped caches and cannot be efficiently applied to L2. As shown in Figure 3.1, around 64% of the L2 cache blocks are faulty and the value of these blocks must be retrieved from the main memory. For our system configuration (Table 3.1), this results in an effective access time of 164 cycles for L2 which hurts the performance drastically. Nevertheless, considering only L1, they achieved 94% yield compared to 99% yield for our scheme.

Wilkerson *et. al* [103] proposed two cache protection schemes that use several layers of shifting to merge multiple defective lines into a single functional line. Their method was originally designed to reduce the operational voltage of the on-chip caches for power saving. Reducing operational voltage of a cache causes the SRAM cells to start failing and they tried to tolerate these unwanted failures. Alternatively, in order to improve the stability of an SRAM cell, Chang *et. al* [29] proposed an 8T SRAM cell, which has been studied and compared with the other alternatives in a more detailed manner by Chen [30] and Verma [98]. These works show that 8T is more effective than simple transistor up-sizing for improving the stability of a bitcell. An 8T cell is more robust against read upset failures compared to a conventional 6T cell due to the isolation of the read and write paths [30].

Table 3.2 summarizes the comparison with these two schemes. As can be seen, Wilkerson's method has a notably higher performance drop-off than ZC. This behavior is due to two reasons: three additional cycles of latency for L2 accesses (compared to 1 cycle for ZC); and, the L1 and L2 capacities are reduced by 50% and 25%, respectively, to provide spares for fixing failures. Wilkerson did not report any power overheads, thus we do our best to provide an estimate in Table 3.2. We ignore overhead due to ECC correction of repair patterns and the shifting layers along with their corresponding decoders. Wilkerson's

Table 3.2: Comparison with recently proposed cache protection schemes

Protection scheme	L1 Cache			L2 Cache			Norm. IPC (SPEC-2K)
	Area over. (%)	Disabled (%)	Power over. (%)	Area over. (%)	Disabled (%)	Power over. (%)	
Wilkerson [103]	15	50	61	7	25	27	0.89
8T [30, 29, 98]	36	0	16	36	0	22	1.0
ZerehCache	16	0	15	8	0	12	0.97

method has a significant power overhead because parallel access to both banks is necessary in the L1 and L2 caches (parallel access only occurs to the L1 ZC), and there is a high leakage power for the ST cells used for the tag array. Lastly, the area overhead of Wilkerson's method is modest, with ZC slightly higher. It should be noted that the area of L2 is around 41 times larger than L1. Consequently, *area* overhead of a protection scheme for the chip is mostly determined by the area overhead for the L2 cache. The 8T cell provides superior performance to either scheme, but at a cost of significant area overhead. The power overhead of the 8T L2 cache is also notably higher than the ZC design. Overall, ZCs can tolerate high defect densities while resulting in a modest amount of performance loss and providing area/power overheads competitive with the best alternatives.

3.6.3 Significance

As we mentioned earlier, large on-chip caches are the major bottlenecks for enhancing process variation tolerance. In our work, we showed the process variation characteristics for a $45nm$ technology. Since the end of the free ride from clock scaling has already arrived, semiconductor companies need to use extremely conservative guard-bands for supply voltage and clock frequency to avoid significant manufacturing yield loss. This has a major impact on the power consumption and operational frequency of modern microprocessors. In order to mitigate these effects, current microprocessors have already been equipped with

ECC and row-redundancy to protect the caches to the first order. An article by Hampson, reported that about 40% yield loss was observed when all forms of redundancy were removed from an Intel die. This was primarily due to the absence of redundancy from on-chip caches. A more detailed study has been conducted on Sun UltraSPARC T1 [106]. In summary, ZC can be leveraged to allow full functionality while imposing overheads competitive with the best known alternatives. This simply translates into higher manufacturing yield and better power/frequency characteristics for high-performance microprocessors.

3.7 Summary

Nanoscale CMOS technologies bring demanding reliability challenges to designers due to high degrees of process variation. In particular, SRAM structures are highly vulnerable to parametric alteration, thus the design of large on-chip caches that are both reliable and efficient is an important problem. In this work, we present ZerehCache, a flexible and dynamically reconfigurable cache architecture that efficiently protects on-chip caches in high failure rate situations. Our solution takes advantage of static multiplexing of the rows along with the added capability of dynamic word-line swapping to maximize the utilization of spare elements. Cache fault patterns are mapped to a graph coloring problem to configure the ZC architecture. We explored a large design space and came up with two suitable architecture configurations for L1/L2 ZCs such that they minimize the area and power overheads while achieving a desired level of robustness. An L1 ZC with 16% and an L2 ZC with 8% area overhead achieve yields of 99% and 96%, respectively. Finally, we compared our scheme with several conventional and state-of-the-art methods to illustrate

its efficiency and effectiveness.

CHAPTER IV

A Polymorphic Cache Design for Enabling Robust Near-Threshold Operation

4.1 Introduction

As mentioned earlier, in Chapter I, Dynamic voltage scaling (DVS) is a widely used technique to reduce the power consumption of microprocessors. However, the supply voltage of a microprocessor cannot be reduced below a certain threshold without drastically sacrificing clock frequency. The minimum achievable voltage for DVS is set such that under the worst-case process variation, the processor operates correctly [35]. The motivation for this work comes from the observation that large SRAM structures are limiting the extent to which operational voltages can be reduced in modern processors. This is because SRAM delay increases at a higher rate than CMOS logic delay as the supply voltage is decreased [94].

An SRAM cell can fail due to the following reasons: a read stability failure, a write stability failure, an access time failure, or a hold failure [5]. Figure 4.1 depicts the bit error rate (BER) of an SRAM cell based on the operational voltage in a $90nm$ technology

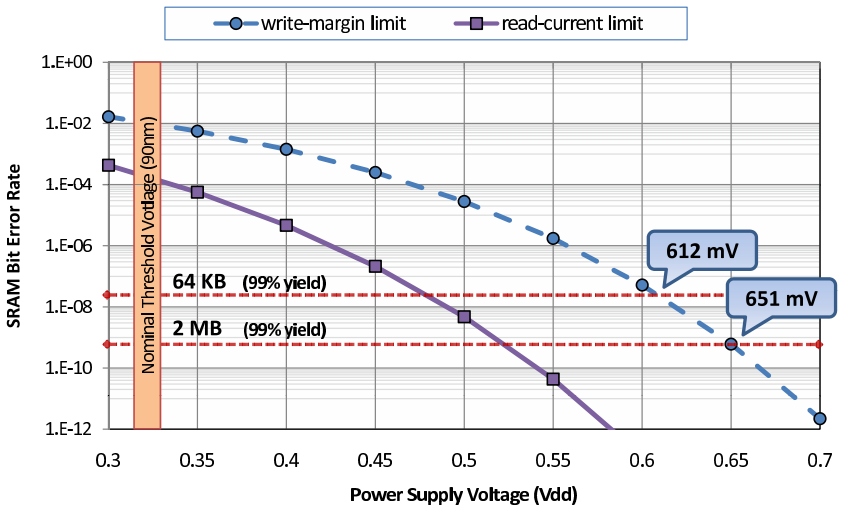


Figure 4.1: Bit error rate for an SRAM cell with varying V_{dd} values in $90nm$. For this technology, the write-margin is the dominant factor and limits the operational voltage of the SRAM structure. Here, the Y-axis is logarithmic, highlighting the extremely fast growth in failure rate with decreasing V_{dd}. The two horizontal dotted-lines mark the failure rates at which the mentioned SRAM structures (64KB and 2MB) can operate with at least **99%** manufacturing yield.

node [70]. The minimum operational voltage of 64KB L1 and 2MB L2 caches is selected to ensure a high expected yield, 99% in Figure 4.1. Due to the higher sensitivity of a bit-cell to parametric variations at lower $V_{dd}s$, the failure rate of an SRAM cell increases exponentially when V_{dd} gets decreased. This exponential increase in the number of faulty cells makes the protection of the on-chip caches much more difficult when operating in the near-threshold region. As can be seen in this figure, the write margin mostly dictates the minimum V_{dd} and it is expected to operate with $V_{dd} \geq 651mV$ due to the dominating failure rate of the 2MB L2 cache. This minimum operational voltage is consistent with predicted and measured values ($\sim 0.7V$) reported in [27].

In the literature, several techniques have been proposed to improve dynamic and/or leakage power of on-chip caches as well as the entire processor [94]. Section 2.2 summarizes some of these low-power design techniques. Most of these methods exploit the

structure specific sleep modes and/or power-aware resource allocation to avoid facing with the failures. Consequently, for low V_{dd} values (e.g., $\leq 651mV$), the amount of power saving for these methods is restricted due to the arising failures in the SRAM structures.

In contrast, the objective of our work is to enable DVS to push the core/processor operating voltage down to the near-threshold region (e.g., low power mode) while preserving correct functionality of on-chip caches. An alternative to this approach is dual-V_{dd}s where core and caches operate at different voltages. However, dual-V_{dd} imposes a high cost in terms of area and design complexity. Voltage-level converters must be added to allow signal sharing between different voltage islands. Furthermore, high voltage memory elements generate noise that victimizes the neighboring low voltage logic circuits, necessitating either shielding or extra noise margins [103].

In this chapter, we target ultra-low voltage operation ($V_{dd} \leq 400mV$) in the near-threshold region which causes an extreme bit-cell failure rate ($> 10^{-3}$) for the on-chip caches – presented as the high failure rate region in Figure 4.2. This figure shows the percentage of *faulty* (i.e., containing at least one faulty bit-cell) bits, bytes, words, blocks, columns, and word-lines for a 2MB L2 cache for different V_{dd} values. Figure 4.2 is generated assuming a uniform failure distribution based on the relation between bit-cell failure rate and V_{dd} in Figure 4.1. In this figure, at $V_{dd} = 350mV$, almost all blocks are faulty while 30% of the words are faulty for the 2MB L2 cache. As can be seen and also discussed earlier for $V_{dd} \geq 651mV$, almost no failure is expected (i.e., *failure-free region*). As can be seen, for the $V_{dd} \leq 400mV$, finer granularities of redundancy are required to allow a high utilization of the spare elements since a large fraction of word-lines, columns, blocks, and even words would be faulty. This increases the complexity of the design since

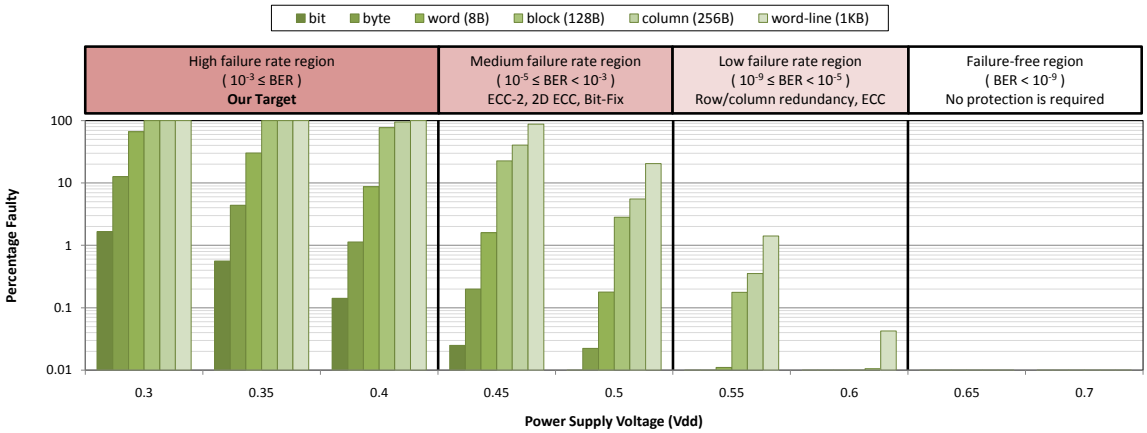


Figure 4.2: Percentage of faulty bits, bytes, words, blocks, columns, and word-lines for a 2MB L2 cache while varying the supply voltage. Here, the Y-axis is logarithmic, highlighting the rapid growth in faulty units when decreasing V_{dd}. The top part of this figure depicts our conceptual division of this V_{dd} range into four different regions based on the protection difficulty. For each region, corresponding bit error rates and also several applicable protection techniques are also shown. In order to operate correctly in the failure-free region, no protection mechanism is required. However, as can be seen, our target is the high failure rate region which causes an avalanche of failures for on-chip caches.

simple disabling (e.g., block or way disabling) or coarse grain redundancy (e.g., row or column redundancy) techniques cannot be efficiently applied.

In this chapter, we propose Archipelago (*AP*) [10, 8], a cache capable of reconfiguring its internal organization to efficiently tolerate the large number of SRAM failures that arise when operating in the near-threshold region. AP allows fault-free operation by partitioning the cache into multiple autonomous *islands* with various sizes. Each island is a group of physical cache word-lines that can operate correctly without using any word-line outside of their group. Each group has a sacrificial word-line which is divided up to multiple redundancy units. These spare units are *directly/indirectly* employed to achieve fault-free operation of the other word-lines in the same group. Sacrificial word-lines do not contribute to the effective cache capacity since they do not store any independent data. The clustering

of the cache to these autonomous islands is done by a configuration algorithm which is described in Section 4.2.3. This adapted version of the minimum clique covering algorithm tries to partition the cache to the least number of islands/*groups* to minimize the number of sacrificial word-lines required for guaranteeing the fault-free operation of the cache. AP enables greater power savings than prior approaches and requires only a single power supply. The overhead of the approach is a small performance penalty (4.6%) when operating in near-threshold mode and a negligible area overhead (2%) for the microprocessor. We apply AP to L1-D, L1-I, and L2 caches to evaluate the achievable power reduction for the overall microprocessor system. A thorough comparison of AP with several well-known proposals is done in Section 4.4.

The primary contributions of this chapter are: **1)** A flexible and highly reconfigurable architecture that can be leveraged to protect regular SRAM structures against high failure rates; **2)** In order to minimize the amount of redundancy required for protecting the cache, we model the cache fault pattern with a proper graph structure to guide a minimum clique covering configuration algorithm for near optimal group formation; **3)** To our knowledge, Archipelago is the first low overhead, fault-tolerant architectural technique which allows the cache to operate correctly when $V_{dd} \leq 400mV$; and **4)** A design space exploration in $90nm$ to show the actual process of selecting the architecture parameters for both L1 and L2 caches.

4.2 Archipelago

In this section, we first describe the AP architecture that adaptively reconfigures itself to absorb failing SRAMs. Next, we present a configuration algorithm that exploits the intrinsic flexibility of the AP architecture to perform a near optimal redundancy assignment. The coalescence of highly flexible hardware and intelligent configuration enables our proposed solution to minimize the impact of operating at near-threshold region on cache characteristics (e.g., size and latency) with minimal overhead.

4.2.1 Baseline AP Architecture

Entering low-power mode causes many bit-cells within a cache to fail. AP provides the appearance of a fully functional cache by tolerating these failures. In order to clarify the operation of AP for set-associative caches, we briefly describe the existing on-chip cache design approaches. For high speed caches, e.g., L1, all the blocks in a set are located in the same physical word-line and read in parallel with the access to the tag array (i.e., *fast-access* design). In case of a cache hit, based on the tag comparison result, the column decoder selects the matched block from the corresponding word-line. For the caches that are not so sensitive to latency, e.g., L2, accesses to the data and tag arrays happen sequentially. Only in the case of a hit, the matched block will be accessed in the data array (i.e., *energy-efficient* design). Although having higher latency for a cache access, the second method has several notable advantages: 1) It allows blocks to be longer since there is no physical constraint for placing multiple blocks in the same word-line. 2) It achieves a better power saving in the case of a miss. 3) Finally, it makes higher degrees of associativity more affordable by

providing a higher efficiency of cache access.

As will be discussed, AP can be easily adapted to both *fast-access* and *energy-efficient* designs. To this end, we partition the cache into several autonomous islands with various sizes. Each island is a group of physical cache word-lines that can operate correctly without using any word-line outside of their group. Each group has a sacrificial word-line which is divided up into multiple redundancy units. These spare units are *directly/indirectly* employed to allow a fault-free operation of the other word-lines in the same group. Put simply, considering one of these groups with $n + 1$ partially functional cache word-lines, AP allows this group to behave as a set of n fully functional cache word-lines.

In the remainder of this chapter, we refer to every physical cache *word-line*, which may contain multiple blocks as a *line*. In our approach, each line is divided into multiple data chunks. Each chunk is labeled faulty if it has at least one faulty bit. Two lines have a collision if they have at least one faulty chunk in the same position. For example, if the second data chunk of the 3*rd* and 6*th* lines is faulty, lines 3 and 6 have a collision. Similarly, in Figure 4.3, lines 10 and 15 are collision-free since they do not have a faulty block in the same position. The primary objective of AP is to form groups such that there are no collisions between any two lines within a group. By definition, it should be clear that collisions are independent of workload running on the system and data stored in the cache.

Figure 4.3 is a toy example that depicts a 2-way set-associative *fast-access* cache with two banks. Each bank has 8 lines and each line consists of two blocks of data which are divided into 3 equally sized data chunks. In this figure, lines 4, 10, and 15 form the 3*rd* group (*G3*) in the cache. As can be seen, there is no collision between any pair of lines

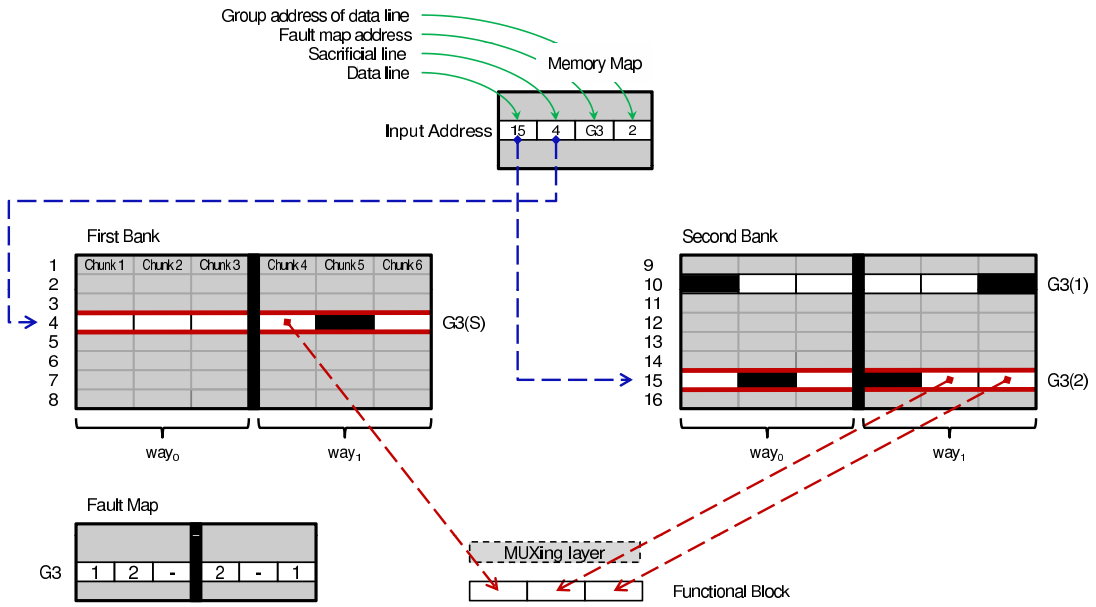


Figure 4.3: Basic structure of a dual-bank 2-way set-associative Archipelago. Two cache banks with eight lines each are shown. Each block consists of 3 equally sized data chunks. Black boxes in each cache line represent chunks of data that have at least one faulty bit. The memory and fault maps, which are essential components of the proposed scheme, are also shown.

within this group – considering both ways simultaneously. Here, line 4 (labeled G3(S)) is the sacrificial line that furnishes the redundancy needed to accommodate the faulty chunks in lines 10 and 15. In order to minimize the access latency overhead, the sacrificial line (4) and the data lines (10 or 15) should be in different banks[1] so that the sacrificial line can be accessed in parallel to the original data line.

In AP, a memory map is used to provide a level of indirection in cache accesses. Each cache access first indexes into this memory map, which supplies the location of the data line and its corresponding sacrificial line. After these two lines have been accessed from their respective banks (different ones), a MUXing layer[2] is used to compose a fault-free

[1] In this work, our approach is described for protecting caches with only two banks. However, it should be clear that our scheme can be naturally extended to caches with any number of banks ≥ 2 without loss of generality.

[2] As a side note, since read and write are symmetric operations, the only modification in the hardware implementation would be to replace the MUXes in the MUXing layer with pass transistors [103].

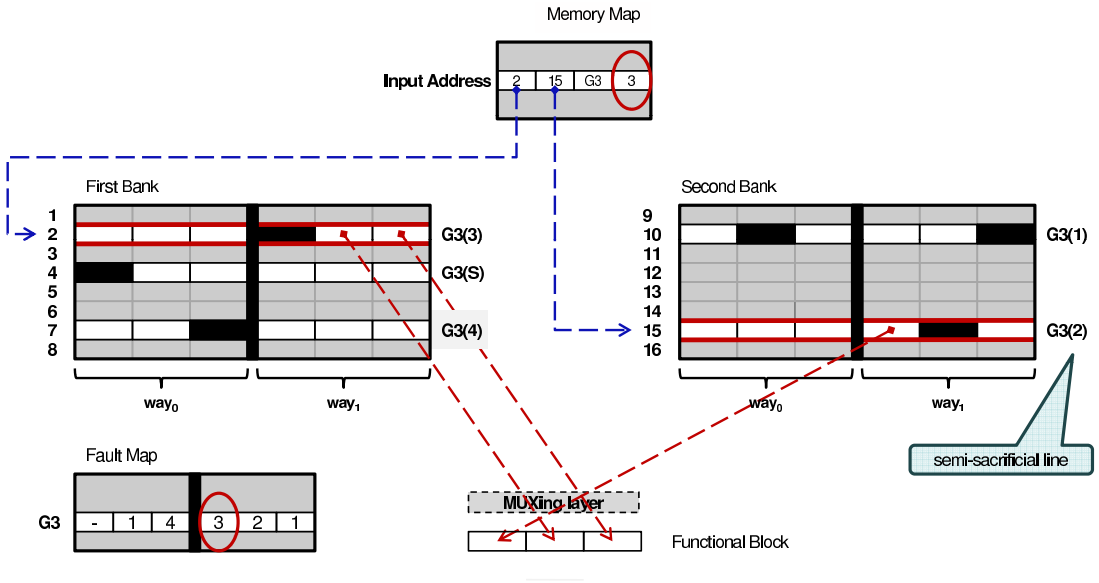
block by selecting the appropriate chunks from each line. Considering the basic design of a *fast-access* cache, based on the tag comparison results, column decoders – which are placed before our MUXing layer – MUX the corresponding blocks out of read word-lines. On the other hand, it should be clear that for *energy-efficient* caches design each word-line might only contain a single block and access to the tag and data array is sequential. As a result, indexing into the memory map will be done after resolving the tag part of the address.

The MUXing layer receives its inputs indirectly from the *fault map*. For a given data line, the fault map determines which chunks are faulty and should be replaced with corresponding chunks from the sacrificial line. To aid in the encoding and decoding of this information, a unique address is assigned to all lines within a group (*group address*). For instance, in Figure 4.3, lines 10 and 15 are the first and second lines of *G3*, respectively. For each data chunk in the sacrificial line, the fault map stores the group address of the line to which that data chunk is assigned. In general, for a given group, if the sacrificial line consists of k data chunks, the fault map requires to store $(i_1, i_2, ..., i_k)$. In this notation, i_j is the group address of the line to which the j*th* data chunk of the sacrificial line is assigned. In our example, the entry which is assigned to the third group (*G3*) in the fault map, contains (1,2,-,2,-,1) for way_0 and way_1. Considering only way_1, this indicates that the 4*th* chunk of the corresponding sacrificial line (*G3(S)*) is devoted to *G3(2)*, the 6*th* chunk is dedicated to *G3(1)*, and the 5*th* chunk is not assigned to any line. Finally, the MUXing layer gets its input from a set of comparators that compare the group address of a data line with fault map entries for the same group. For example, group address of line 15 is 2 – read from memory map – which gets compared with *G3*'s fault map entries.

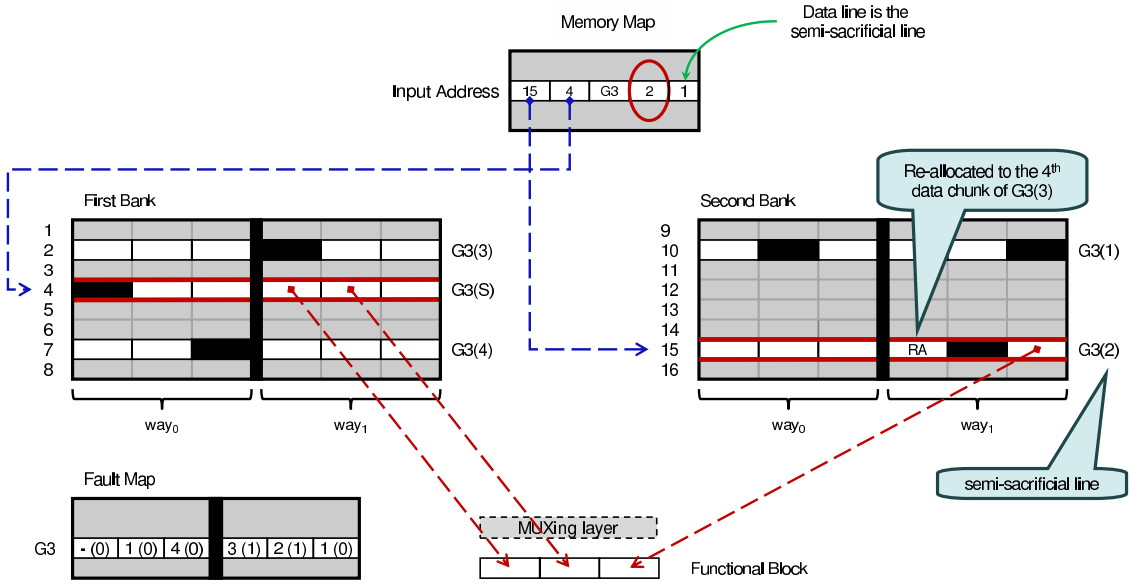
4.2.2 AP with Relaxed Group Formation

Since every group requires a sacrificial line be dedicated solely for redundancy, AP strives to minimize the total number of groups that must be formed. Given that the number of lines is fixed within a cache, achieving this objective implies that larger groups are preferred over smaller ones. In order to improve the likelihood of forming large groups, we remove the constraint that forces the sacrificial line, from a particular group, to be in a different bank than all the other lines. This allows any set of lines from the two banks to form a group. However, in order to minimize the access latency, we do not allow a group to derive all its lines from the same bank. In other words, each group should have at one line in each bank to allow a parallel access to the original and spare data. Relaxing the mentioned constraint, gives rise to two new read access scenarios in addition to the standard case described in Figure 4.3.

Handling Different Types of Accesses: In Figure 4.4(a), lines 2, 4, 7, 10 and 15 are in the same group, with line 4 serving as the sacrificial line. However, when line 2 or 7 is accessed as a data line, a parallel access to line 4 cannot be performed since they are in the same bank. To handle such a scenario, a small and transparent modification to the AP architecture is needed. We arbitrarily select a line from the bank not containing the sacrificial line (but still from the same group) and label it as a *semi-sacrificial* line (line 15 in Figure 4.4(a)). This semi-sacrificial line can be used to replace faulty chunks from cache lines which are in the same bank as the sacrificial line. However, *in contrast to the sacrificial line, the semi-sacrificial line still contributes to the effective cache space*. In other words, a semi-sacrificial line only acts as a level of indirection for redundancy substitution by bor-



(a) Reading a line (G3(3)) from the same bank as the sacrificial line (G3(S))



(b) Reading from the semi-sacrificial line (G3(2))

Figure 4.4: Two special read-access scenarios. A standard read access is illustrated in Figure 4.3. Notice the extra bit that has been added to both the memory map and every fault map entry to handle scenario (b). Since the 4*th* data chunk of semi-sacrificial line is re-allocated, it is marked as **RA** in scenario (b).

rowing redundancy from its corresponding sacrificial line. Moreover, semi-sacrificial lines guarantee the parallel access to the original and spare data in all possible scenarios. Consequently, the faulty chunks of the lines 2 and 7 are replaced using *G3(2)* instead of the

G3(S). With the addition of the semi-sacrificial line, accesses to the cache can be divided into three categories:

1) *Accesses to data and sacrificial lines that reside in different banks*. This is the base case which is demonstrated in Figure 4.3 and does not require any special consideration beyond what is described earlier.

2) *Accesses to data lines that are the in the same bank as the corresponding sacrificial line*. This case is illustrated in Figure 4.4(a). Line *G3(3)* is the data line and line *G3(2)* is the semi-sacrificial line which supplies the replacement chunks for the faulty ones in *G3(3)*. Instead of accessing *G3(S)*, the memory map remaps the address of the sacrificial line to the address of *G3(2)*. This case is particularly interesting, since no other parts of the procedure must be adjusted to support this access scenario. The fault map is still used, unmodified, to indicate that the 4*th* data chunk from *G3(3)* is faulty and should be substituted. Therefore, AP replaces this faulty chunk of *G3(3)* by the semi-sacrificial line's 4*th* chunk instead of sacrificial line's 4*th* chunk.

3) *Accesses to a semi-sacrificial line*. This case is demonstrated in Figure 4.4(b) for which two small modifications to the access procedure are necessary. An additional bit is added to memory map entries indicating whether the accessed data line is the semi-sacrificial line. As can be seen, the 4*th* data chunk of *G3(2)* has been re-allocated to *G3(3)*. Consequently, we artificially mark the 4*th* chunk of the semi-sacrificial line as "re-allocated" (**RA** in Figure 4.4(b)). When *G3(2)* is accessed, its faulty and also re-allocated data chunks (i.e., the 4*th* and 5*th* chunks) are supplied as expected by the corresponding chunks from the sacrificial line (*G3(S)*). However, this cannot be easily done using the un-modified fault-map since the 4*th* entry of the fault map points to the faulty chunks of data

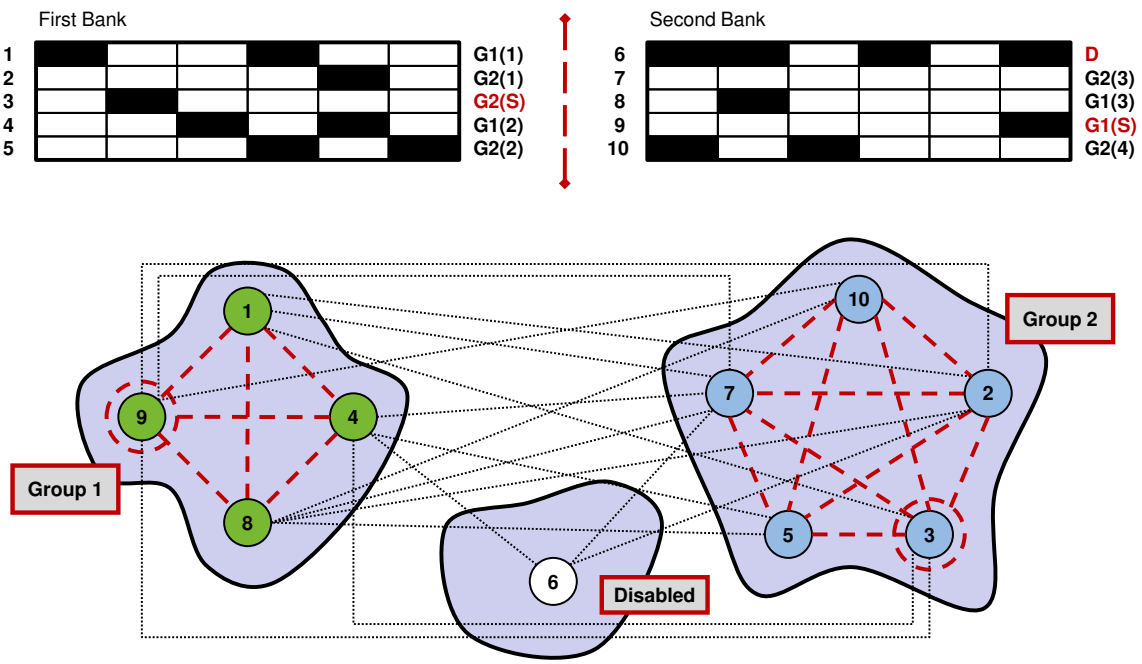


Figure 4.5: A simplified example of the minimum clique covering process for a given distribution of faults in the cache banks. Here, each bank has only 5 lines. The solver disables the 6*th* line since it has many faulty chunks and, is therefore very expensive to repair. Two cliques are formed by the solver and lines 9 and 3 are designated as sacrificial lines for groups 1 and 2, respectively. Moreover, the conceptual partitioning of the cache to distinct islands is also demonstrated.

from *G3(3)*. As a result, the system cannot identify that the 4*th* chunk of the semi-sacrificial line should be replaced during an access. In order to tackle this problem, we add an extra bit to every fault map entry which indicates whether the corresponding data chunk should be replaced when accessing the semi-sacrificial line. For instance, in Figure 4.4(b), since the 4*th* and 5*th* chunks should be replaced, their corresponding bits in the fault map have been set to "1".

4.2.3 AP Configuration

To maximize the number of functional lines in the cache, we need to minimize the number of sacrificial lines required to enable fault-free operation. As previously discussed, there is a single sacrificial line devoted to every group of lines. This sacrificial line is not addressable as a data line since it does not store any independent data. In other words, sacrificial lines do not contribute to the usable capacity of the cache. Depending on the number of collision-free groups that are formed, the effective capacity of the cache can vary dramatically. This motivates the need to minimize the total number of groups required.

Problem Modeling: Here, we model the problem as a graph in which every group corresponds to a clique. To minimize the number of groups in a given faulty cache – upper part of Figure 4.5 with two banks, we model the problem as a minimum clique cover (*MCC*) problem [38]. Figure 4.5 shows the process of forming the groups given a fault pattern for the cache. Each node in the constructed graph, ten in all, is a cache line. There is an edge between two nodes if and only if there is no collision between the corresponding lines. Therefore, it is possible to assign connected nodes to the same group. For example, there is an edge between lines 2 and 3 but no edge between lines 1 and 10.

As mentioned earlier, a group is a set of lines for which there is no collision between any pair of lines. By constructing the graph this way, a collision free group forms a clique (i.e., there is an edge between every pair of nodes). As a result, the task of forming groups can be represented as finding the cliques in the constructed graph. However, since we are interested to minimize the number of sacrificial lines, this problem turns to finding the minimum number of cliques that cover the entire graph. In Figure 4.5, lines 1, 4, 8, and 9

form the first group (*G1*, clique with size 4) while lines 2, 3, 5, 7, and 10 form the second group (*G2*, clique with size 5). Line 6 is disabled (*D*) since it contains 4 faulty chunks and repairing it is not cost effective. In general, a line gets disabled, if its corresponding node in the graph does not belong to any clique with size ≥ 2. Here, the fault map has 2 lines which correspond to the sacrificial lines 9 and 3. As can be seen in this figure, there are 16 faulty chunks out of 60, therefore, at most $10 - (\lceil \frac{16}{6} \rceil) = 7$ cache lines can be kept functional after the grouping. Ultimately, our configuration algorithm can achieve this best case, highlighting the effectiveness of our approach.

MCC Solver: We use the transformation described in [50] to convert the MCC problem to a minimum chromatic number problem. After applying this transformation, the final graph is passed to the DSATUR solver [55] which uses a well-known and efficient approximation algorithm. As shown in [55], the approximation factor for the DSATUR solver is $\leq 6.1\%$ for various graph densities. For the problem size that we target in this work, the run time of the DSATUR algorithm is always less than 5*ms*. In contrast, the full backtrack-based solver (e.g., optimal solver) takes several days to solve *one instance* of our configuration problem which makes it infeasible to be used in practice.

Nevertheless, the original solver's answer is not directly applicable to our problem. Since the solver is free to form a group in any possible way that minimizes the number of cliques, it is possible to have a clique with nodes in only one bank. This latter case is not a feasible solution since it blocks the parallel access to the spare elements. As a result, we apply a set of modifications to the original solver for making it suitable to our application:

1) We force the solver to pick the second line of the group from a different bank from the first line of the group. This small modification assures us that at least one line is selected

from each bank and allows us to take care of the parallel access problem discussed above.

2) An artifact in the DSATUR solver algorithm can sometimes cause it to disable a large fraction of cache lines. More specifically, it picks the nodes for coloring only based on the degree of saturation which is proportional to the reciprocal of the node degree in our constructed graph [55]. Because of this bias, all the lines from one bank might be selected while there are many unassigned lines in the other bank. In such a scenario, the unassigned lines have to be disabled which results in a large fraction of disabled lines. In order to solve this problem, we modify the original DSATUR algorithm to pick the lines from both banks more evenly. This was done by giving artificial priority to the lines in a bank that has more *unassigned lines* at the beginning of each assignment phase.

3) As will be discussed in Section 4.3, minimizing the area overhead of the fault map carries a major significance for our scheme. The size of each entry in the fault map is proportional to the $\log_2(max\{clique\ size\})$. Therefore, to reduce the size of the fault map, an upper bound (e.g., 64) can be placed on the maximum clique size (*MCS*). By adding this feature, all the cliques which are found by the DSATUR solver can be forced to have a size smaller than or equal to *MCS*.

The main plot in Figure 4.6 depicts the distribution of clique sizes for different versions of the solver: base (base DSATUR solver), 2nd non-fair (base solver + modification (1)), 2nd fair (base solver + modifications (1,2)), and 64-cap (base solver + modifications(1,2,3)). This data is generated using 1000 iterations of a Monte Carlo simulation for a 2MB L2 cache with 2048 lines. It should be clear that the size of the fault map is proportional to the $(Num.\ of\ Cliques) \times \log_2(MCS)$ which implies a small number of large cliques is preferable. Furthermore, since $(Num.\ of\ Cliques) \times (Average\ Clique\ Size)$ is

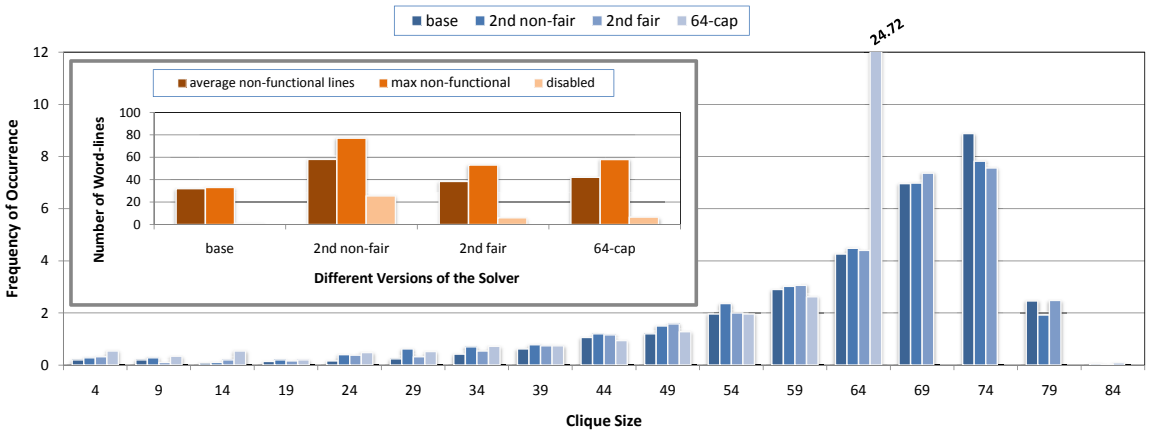


Figure 4.6: Distribution of the clique size for different versions of the solver based on Monte Carlo simulation. Note that for 64-cap, the size of all cliques is ≤ 64. Here, the number of non-functional lines is the summation of the number of sacrificial lines and the number of disabled lines. The plot in the insert depicts the average number of non-functional cache lines, the maximum number of non-functional lines, and the number of disabled lines while achieving 99% yield.

equal to the total number of word-lines, a constant value, *MCS* should be as close as possible to the $Average\ Clique\ Size$. As a result, the most desirable distribution of clique sizes would be a tight distribution around large clique sizes. As can be seen in Figure 4.6, a majority of cliques fall into the narrow region of 60 to 80 nodes. This tight distribution shows the efficiency and proper balancing of the group sizes in the process of group formation by the different versions of the solver. The smaller plot in this figure demonstrates different characteristics of these 4 versions of the solver. In this plot, the number of non-functional lines is the summation of the number of sacrificial lines and the number of disabled lines. As can be seen, the second modification (i.e., fairness) can effectively reduce the average number of disabled lines from 25.5 to 6.1.

Another observation is that the constraint $MCS = 64$ increases the maximum number of non-functional lines by 9% while it reduces size of each fault map entry by 17% – from

7 bits to 6 bits. For each cache instance, the number of lines in the fault map array is equal to the number of sacrificial lines (e.g., 2 in Figure 4.5). However, due to the presence of process variation in a large population of fabricated chips, different fault patterns should be expected. As we described earlier, in our evaluation, we employ a Monte Carlo simulation to generate a population of 1000 cache instances and the total number of fault map lines is determined based on the maximum number of sacrificial lines while achieving a 99% yield.

Hardware Configuration: In order to configure AP, the memory and fault maps need to be filled. The initial step involves solving the MCC configuration problem for a given cache fault-pattern. Given the MCC solution, each line – a node in the graph – can be classified as: **1)** *Data line:* For each data line, a new memory map entry should be allocated. Each memory map entry has 5 fields (Figure 4.4(b)): The first field is the data line address. If this line is in the same bank as its respective sacrificial line, the second field will set to the address of the respective semi-sacrificial line. Otherwise, it will get the address of the sacrificial line. The third, fourth, and fifth fields should be set to the data line's group number, group address, and "0", respectively. **2)** *Sacrificial line:* Although no change in the memory map is required, a fault map entry should be allocated for each sacrificial line. Each fault map entry has a field per data chunk – 6 fields in Figure 4.4. For faulty data chunks of a sacrificial line or the ones which are not assigned to any faulty data chunks, corresponding fields in the fault map should be set to "-". For other data chunks, corresponding fault map fields should be set to the group addresses of the data/semi-sacrificial lines to which those data chunks are assigned. **3)** *Semi-sacrificial line:* This is similar to the first case, except the last field of the memory map entry should be set to "1". **4)** *Disabled line:* Nothing needs to be done in this case.

Low Power Mode Operation: The first time a processor switches to low power mode, the built-in self test (BIST) module scans the cache for the potential faulty cells. After determining the faulty chunks of each cache line in low power mode, the processor switches back to high power mode and constructs the mentioned graph and solves the MCC problem using the modified DSATUR solver. As mentioned before, the solver time for a 2MB L2 cache is less than 5*ms* on an Intel Core™2 Duo machine. This solution contains the information that is required to be stored in the memory and the fault maps. This configuration information can be stored on the hard-drive and is written to the memory map and fault map at the next system boot-up. In addition, the memory map, fault map and the tag arrays need to be protected using, for example, the well studied 10T cell [27] which has about 66% area overhead for these relatively small arrays. These 10T cells are able to meet the target voltage in this work for the aforementioned memory structures without failing. However, as we will discuss in Section 4.4, 10T cells are not a cost-effective option for protecting the entire L1 and L2 caches.

Cache Addressing Mechanism after Capacity Reduction: In our design, the memory map can be used to remap the *original address* of the sacrificial lines to other functional lines. As a result, a small fraction of the sets will have the functionality of two sets. For those dual-set word-lines, associativity is reduced by half. These dual-set lines are distributed evenly across the cache to prevent biased miss rate for an address sub-range. The tag comparison and replacement logic needs straight-forward modifications to make this work, details of which are omitted in the interest of space.

High Power Mode Operation: In the high power mode, our scheme is turned off in order to minimize the unwanted overheads: **1)** All the cache lines are functional and there is

no sacrifice of the cache capacity. **2)** Assuming clock gating, there is a negligible overhead for the dynamic power due to the switching in the bypass MUXes which consists of the MUXing layer and an additional MUX for bypassing the memory map. **3)** Leakage power overhead remains the same. However, power gating techniques can be used for general leakage mitigation [49].

4.3 Evaluation

This section evaluates the potential of AP in reducing the power of the processor while keeping the overheads as low as possible. Comparisons with related work are presented in the next section.

4.3.1 Methodology

For performance evaluation, we use SimAlpha, a validated micro-architectural simulator based on the SimpleScalar out-of-order simulator [14]. The processor is configured as shown in Table 4.1 and is modeled after the DEC Alpha 21364 at an ambient temperature of $40°C$ [81, 57]. Dynamic processor power consumption is calculated using Wattch [26] based on the activity factors of individual core structures, and leakage power is computed with HotLeakage [109]. CACTI [72] is leveraged to evaluate the delay, power, and area of the base *cache* structures. To take into account the overheads of the memory map and fault map arrays, we use the SRAM generator module provided by the $90nm$ Artisan Memory Compiler. Lastly, the Synopsys standard industrial tool-chain (with a TSMC $90nm$ technology library) is used to evaluate the overheads of the remaining miscellaneous logic (i.e,

bypass MUXes, comparators, subset tag comparison disabling, etc.). Moreover, to find the matching frequency for a given V_{dd}, we use the alpha-power model described in [66].

For a given set of cache parameters (e.g., V_{dd}, chunk size, MCS, etc.), a Monte Carlo simulation with 1000 iterations is performed using the modified DSATUR solver described in Section 4.2 to identify the portion of the cache that should be disabled. As discussed earlier, solutions generated by the MCC solver target a 99% yield. In other words, only 1% of manufactured and configured on-chip caches are allowed to exhibit failures when operating in low-power mode. On the other hand, the target yield directly impacts the size of the fault map. Its size is set based on the maximum number of cliques formed across all cache instances in the Monte Carlo process when ignoring as many worst-case situations as the target yield allows.

Table 4.1: The target system configuration

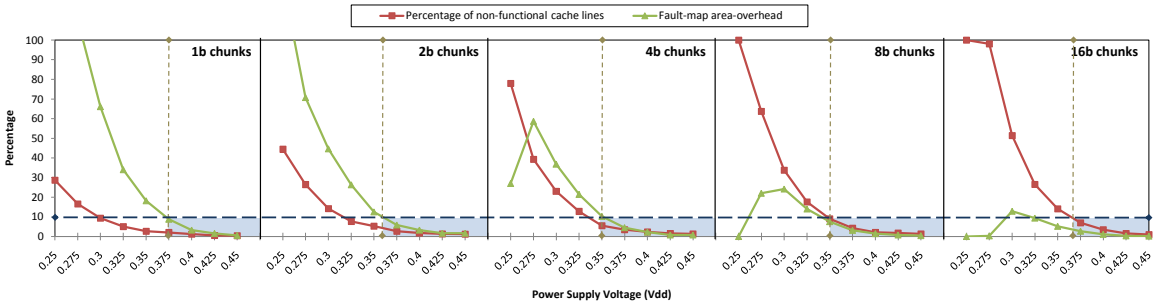
Parameters	**Value**
Technology	90 nm
Clock frequency	1.9 GHz [57]
V_{dd} nominal	1.2 V
L1 Cache	2 banks 64KB data, 2 banks 64KB instruction, split, 2-way set associative, 4 cycles hit latency, 1 port, LRU, 64B block size, write-back
L2 Cache	2 banks 2MB Unified, 8-way set associative, 10 cycles hit latency, 1 port, LRU, 128B block size, write-back
Registers	80 integer, 72 floating point
ROB (re-ordering buffer)	128 entries
LSQ (load/store queue)	64 entries
Instruction fetch buffer	32 instructions
Integer/FP issue queue	32/32 entries
FU (functional unit)	4 int ALU, 4 int mult/div, 2 memory system ports
FPU (floating point unit)	4 FP ALU, 1 FP mult/div
Main memory	225 cycles (high power), 34 cycles (low power)
Branch predictor	combined (bimodal and 2-level)
BHT (branch history table)	4096 entries
RAS (return address stack)	32 entries
BTB (branch target buffer)	2048 entries, 2-way associative

4.3.2 Design Space Exploration

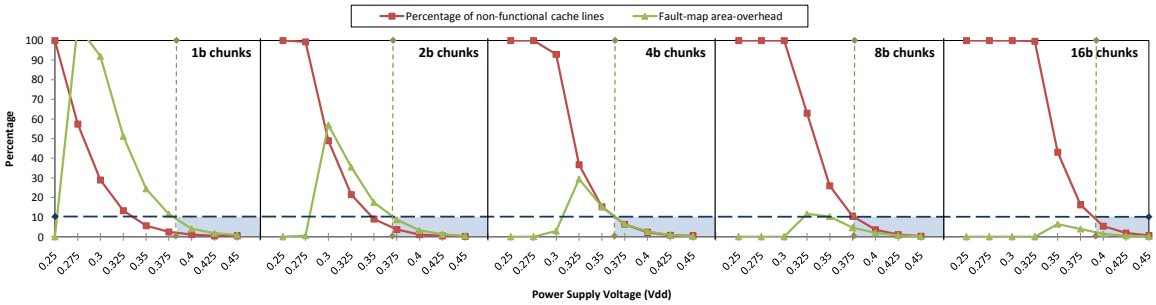
Figure 4.7 shows the process of determining the minimum achievable V_{dd} for our target system. In this figure, chunk size varies from 1 *bit* to 16 *bits* for both L1 and L2 caches. In high-power mode, both fault and memory map arrays remain idle and leak power. It is crucial to minimize the size of these structures. The size of the memory map is essentially fixed by the number of lines in the cache. The fault map size, however, can vary significantly depending on configuration parameters, motivating a closer look at the size of the fault map as an important design factor. Consequently, we limit the area overhead of the fault map to 10% (of the cache area). Furthermore, since cache size has a strong correlation with system performance, we limit our scheme to setting at most 10% of the cache lines to *non-functional*.

As evident in Figure 4.7, decreasing V_{dd} increases the non-functional portion of the cache and also the area of the fault map array. However, beyond a certain point, the area overhead of the fault map starts decreasing. This phenomena is due to the large fraction of the cache lines that get *disabled* as lowering V_{dd} leads to increasing error rates and a precipitous increase in faulty chunks. As we mentioned earlier, for these disabled lines, no entry in the fault map array is required.

Here, the vertical dotted lines highlight the minimum achievable V_{dd} based on the aforementioned 10% limits on non-functional lines and fault map size. It is notable that for small chunk sizes, area overhead of the fault map is the limiting factor, while for larger chunks, the number of non-functional lines becomes dominant. As can be seen in this figure, the minimum achievable V_{dd} for L1 is lower than L2. This was expected due to L2's longer



(a) Percentage of non-functional lines and area overhead of the fault-map for **L1** while varying V_{dd} and data chunk size

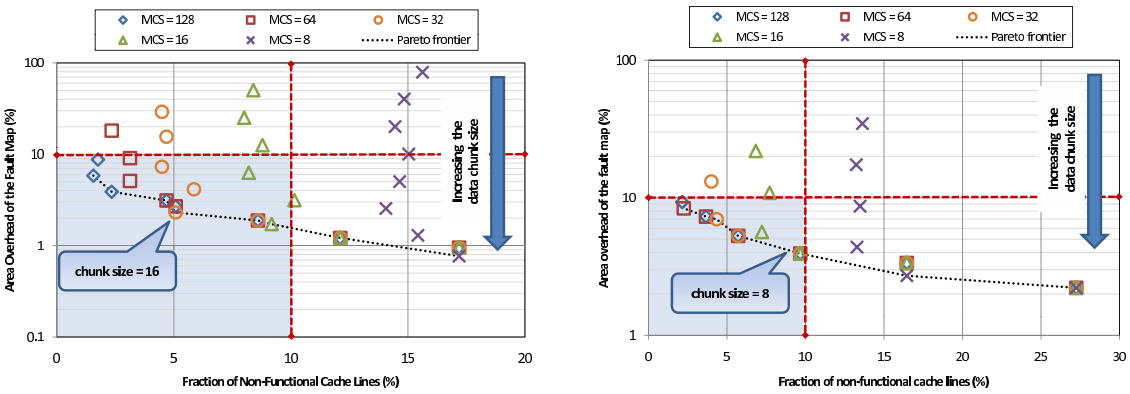


(b) Percentage of non-functional lines and area overhead of the fault-map for **L2** while varying V_{dd} and data chunk size

Figure 4.7: Process of determining the minimum achievable V_{dd} for L1 and L2 caches while limiting the fraction of the non-functional cache lines and also the area overhead of the fault map structure to $\leq 10\%$. Moreover, in these 10 sub-plots, vertical dotted lines show the minimum achievable V_{dd} while data chunk size varies from 1 *bit* to 16 *bits*.

lines and larger size which makes protection of L2 harder than L1 [103]. Therefore, L2 protection cost dictates the minimum operating voltage of our system. In addition, based on the trend in Figure 4.7, it should be clear that a chunk size outside of the presented range will only result in a higher minimum V_{dd}. As a result, we select $375mV$ as the minimum V_{dd} (i.e, low-power mode operating voltage) since all other lower voltages violate our 10% limits.

For $V_{dd} = 375mV$, a design space exploration of L1-(D/I) and L2 caches is demonstrated in Figure 4.8. There are two important parameters to note: **1)** MCS, the maximum allowable clique size (Section 4.2.3) and **2)** chunk size, varying from 1*b* to 128*b* for L1 and



(a) L1 design space exploration

(b) L2 design space exploration

Figure 4.8: Design points for different Maximum Clique Size (MCS) and chunk size pairs are shown that can achieve a 99% yield. For each MCS value, corresponding chunk sizes from $\{2^n \mid n \in \{0, 1, ..., 7\}\}$ for L1 and from $\{2^n \mid n \in \{0, 1, ..., 5\}\}$ for L2 are chosen. The shaded boxes represents the region of interest where both the fault-map overhead and the fraction of non-functional lines is limited to $\leq 10\%$. The black dotted line is the Pareto frontier.

from 1*b* to 32*b* for L2. From Figure 4.8, it should be clear that the chunk sizes outside of the presented range always violate at least one of our aforementioned 10% limits. For every pair of (MCS, chunk size), a Monte Carlo simulation, targeting 99% yield, is performed with 1000 iterations. This simulation identifies the necessary area overhead of the fault map and the fraction of non-functional lines in the cache. Here, we still limit the fraction of the non-functional lines to 10% while trying to minimize the area overhead of the fault map.

In Figure 4.8(a), within the shaded region, only points on the black dotted line (Pareto frontier) are considered since they dominate the other design points. Note that making the chunk size larger decreases the area overhead of the fault map since the number of entries in the fault map is reduced. However, this reduction is also accompanied by an increase in the fraction of non-functional cache lines, the result of fewer edges in the graph described in Section 4.2.3. For L1, the design point with $MCS = 32$ and $chunk\ size = 16$ bits is

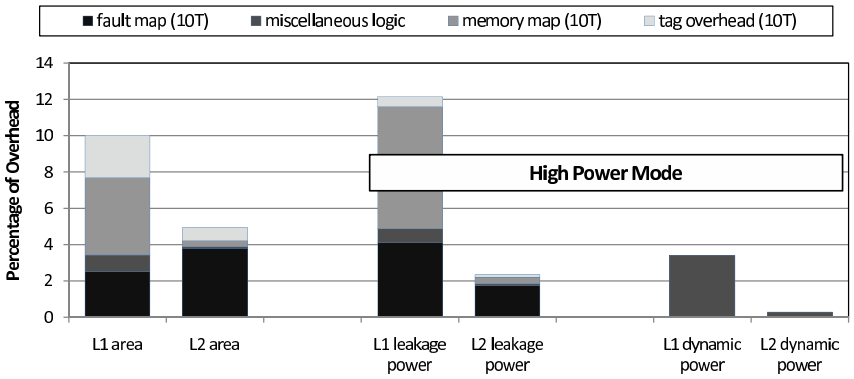


Figure 4.9: Area, leakage, and dynamic power overheads of our scheme for both L1 and L2 caches. Here, 10T cell is used for protecting fault map, memory map, and tag arrays.

selected, highlighting an interesting trade-off between the area overhead of the fault map and the fraction of non-functional lines. By repeating the same process for L2, as illustrated in Figure 4.8(b), the design point with $MCS = 16$ and $chunk\ size = 8$ bits is selected.

4.3.3 Results

Figure 4.9 summarizes the overheads of our scheme for both L1 and L2 caches. As mentioned before, we also account for the overheads of using 10T SRAM cells [27] for protecting the tag, fault map, and memory map arrays in low-power mode. In addition, the fault map, memory map, and the second bank have their own separate decoders which are accounted for in our evaluation. Leakage overhead in high power mode corresponds to the fault map, memory map, miscellaneous logic, and extra leakage of 10T cells for tag array. Note, the memory map is a far greater contributor to area and leakage power overhead in the L1 than in the L2. The reason behind this is that the L1 has only $\frac{1}{4}$ the lines of the L2, while its overall size is $\frac{1}{32}$. For L2, the fault map is the major component of overhead. Due to its significantly larger size, the L2 cache dominates the processor *leakage* and *area* overheads. Nevertheless, as can be seen in this figure, our scheme has only modest overheads for the

L2 cache. Dynamic power overhead in high-power mode can be mainly attributed to bypass MUXes since we assume clock gating for the fault map and memory map arrays. In AP, when in low-power mode, the memory map and MUXing layer are on the critical path of the cache access. Based on our timing analysis, the access delay overhead of L1 and L2 are $0.41ns$ and $0.58ns$, respectively. Based on our system clock frequency (Table 4.1), this translates to 1 additional cycle latency for L1 and 2 additional cycles for L2 in low-power mode.

As mentioned before, the fraction of non-functional lines is based on a 99% yield in a 1000-run for a Monte Carlo simulation. As a result, the fraction of non-functional lines would be smaller than what is presented in Figure 4.8 for many chips. This is because across all the fabricated dies, process induced parametric variation causes different cache fault patterns to appear. However, we consider the worst-case capacity loss during our evaluation. In order to evaluate the *worst-case* performance penalty of our scheme in low-power mode, we ran the cross-compiled Alpha binaries of SPEC-CPU-2K benchmark suite on SimAlpha after fast-forwarding to an early SimPoint [85]. We assume one extra cycle latency for L1 and 2 extra cycles for L2. Cache size is also reduced based on the fraction of non-functional lines in Figure 4.8. On average, a 4.6% performance penalty is seen in low-power mode from which 0.6% is contributed by the cache capacity loss due to the presence of non-functional lines (Figure 4.10). As can be seen, our strict limit on the fraction of non-functional lines results in minimal impact on performance because of cache capacity loss. However, one should note that low-power mode performance is usually not a major concern. In high power mode, there is no capacity loss since no failures need to be tolerated. Furthermore, based on our CACTI delay results and the frequency of the

system (Table 4.1), there is enough slack on the access time of our L1 and L2 caches to fit the small bypass MUXes (additional 0.07ns delay) without adding any extra cycles to the access time of these caches. In other words, there is no performance loss in high-power mode. However, one might have a cache design without any slack available. In that scenario, we add an additional cycle in high-power mode for L1 and L2, which translates into a 3.6% performance loss for SPEC-2K.

Summary of benefits and overheads: Figure 4.11 shows the savings and overheads for the Alpha 21364 microprocessor using AP for protecting the on-chip caches. As can be seen in Figure 4.11(b), the overheads of the proposed method are almost negligible. These overheads are evaluated in 90nm using the methodology described in Section 4.3.1. On the other hand, Figure 4.11(a) depicts the percentage reduction in leakage power, dynamic power, and minimum achievable supply voltage by using AP for protecting the on-chip caches. These results are reported in the 45nm, 65nm, 90nm, and 130nm technology nodes. The relation between the supply voltage and the expected SRAM bit-cell failure

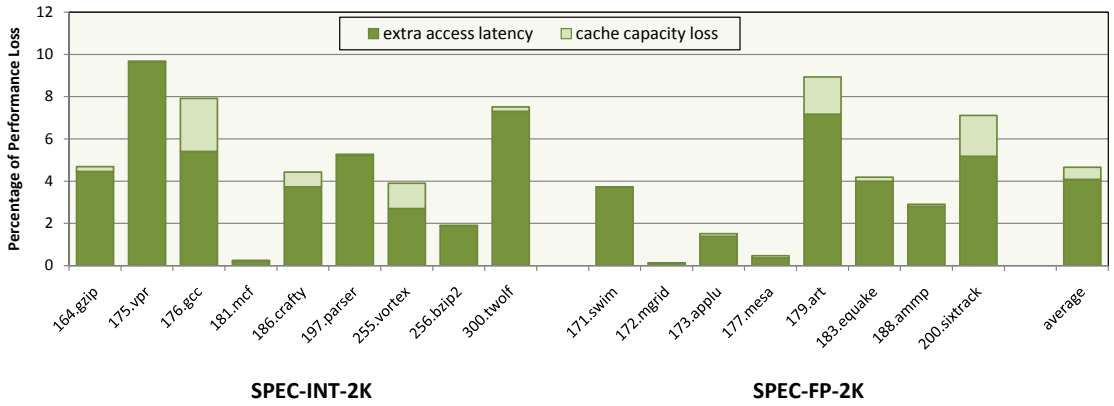
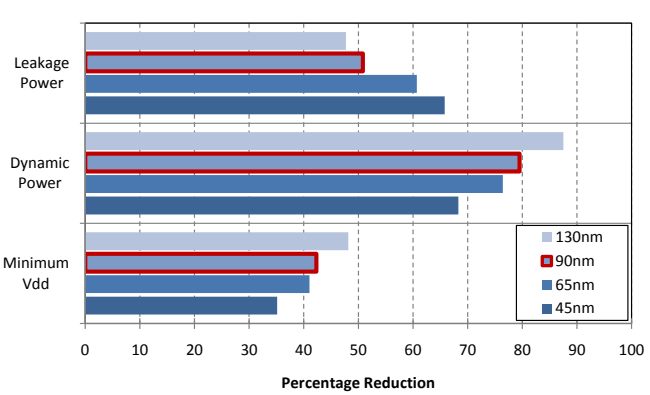
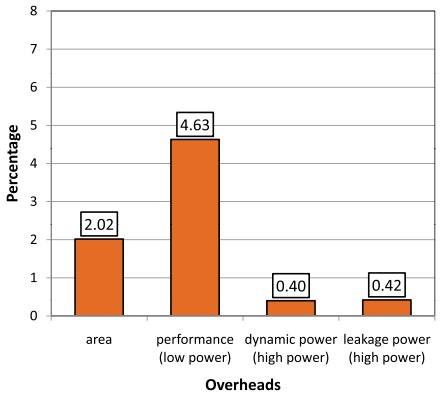


Figure 4.10: Performance loss break-down for our scheme in low power mode using SPEC-2K benchmarks. As can be seen, since the fraction of non-functional lines is limited to be less than 10%, the access latency overhead is the dominant factor in performance penalty.



(a) Percentage of reduction in leakage power, dynamic power, and minimum achievable V_{dd} using 4 different technology nodes (i.e., $45nm$, $65nm$, $90nm$, $130nm$).



(b) Overheads of using AP for protecting on-chip caches of the mentioned microprocessor in $90nm$

Figure 4.11: Low-power mode benefits and also overheads of an Alpha 21364 microprocessor system (Table 4.1) augmented with Archipelago. Here, we account for the dynamic power overhead of accessing the second bank in low power mode for handling failures.

rate for these four technology nodes are extracted from [103, 58, 78, 37, 77, 28, 21]. Considering the $90nm$ technology node, AP enables DVS to save 79% dynamic power and 51% static power in the near-threshold region. With the aggressive technology scaling, the systematic and random variations are expected to increase [77]. This results in higher sensitivity/vulnerability of SRAM cells to power supply variations. Hence, the percentage of reduction in dynamic power/minimum achievable V_{dd} gradually reduces when heading toward deeper sub-micron technologies. Nevertheless, a 68% dynamic power reduction can be achieved in $45nm$. In contrast, for leakage power, the percentage reduction gradually increases as the technology scales down. This is mainly due to the drastic differences in correlations between the leakage power and V_{dd} across technology generations. In deeper technology nodes, a reduction in V_{dd} manifests as a much larger saving in leakage power [44]. Therefore, for deeper technologies, even though we achieve less V_{dd} reduction, the net leakage power saving is larger. Due to the excessive increase in the sub-threshold

Table 4.2: Comparison of different protection schemes

Protection scheme	**MinV_{dd} (mV)**	**Cache area overhead (%)**	**Freq. (MHz)**	**Norm. IPC**	**Power norm. to Archipelago**
6T cell	651	0.0	920	1.0	4.35
Row redun.	550	5.1	710	1.0	2.62
SECDED	530	6.3	670	1.0	2.35
ECC-2	490	7.4	580	0.96	1.87
ZerehCache [11]	430	10.7	450	0.96	1.31
Wilkerson [103]	420	3.4	430	0.89	1.35
10T cell [27]	380	66	340	1.0	1.17
Archipelago	375	5.2	320	0.95	1.0

transistor leakage, it is expected that leakage power dominates the total power consumption of a chip in the future technologies [44].

4.4 Quantitative Comparison to Alternative Methods

In order to illustrate the benefits of our design, we quantitatively compare AP with the baseline 6T SRAM cell, three well-known conventional cache protection methods (row redundancy, 1-bit error correction code (SECDED), and 2-bit ECC), and three state of the art works (ZerehCache [11], Wilkerson et. al. [103], and 10T SRAM cell [27]). Table 4.2 summarizes this comparison – in $90nm$ – based on the minimum achievable V_{dd}, area overhead for the caches, processor clock frequency, normalized IPC, and normalized power. In order to have a fair comparison, the number of redundant rows and coding granularities are set so that the area overheads of the row redundancy, SECDED, ECC-2, and AP are equal/comparable. In this table, different techniques are sorted based on their minimum achievable V_{dd} – targeting 99% manufacturing yield for on-chip caches.

Overheads for AP are calculated by considering all extra SRAM structures, decoders, MUXing layer, comparators, bypass MUXes, and other miscellaneous small logics. How-

ever, some of the comparisons shown in Table 4.2 are conservative at best because we overlook: **1)** Area and delay overhead of the programmable decoder for row redundancy. **2)** Area and power overhead of the encoder and decoder for ECC and ECC-2. **3)** Power overhead of the extra logic which is added to the caches in [103]. **4)** A $380mV$ minimum achievable V_{dd} for the 10T cell was derived in $65nm$ [27] and it is clear that in $90nm$ this value would be higher.

Overall, even by overlooking all the mentioned overheads for other schemes, AP can still achieve the lowest V_{dd} and highest power saving among the other methods. The three closest competitors to our work are the 10T cell, Wilkerson [103], and ZerehCache [11]. However, the 10T cell incurs 66% area overhead which acts as a burden in the high power mode. In contrast, our scheme only has 5.2% area overhead and does not considerably influence the normal operation of the system. Comparing to [103], our scheme can achieve a significantly lower V_{dd} and higher power saving. In addition, Wilkerson's work suffers an 11% performance drop-off – for SPEC-2K – in low power mode and 6% in high power mode. Comparing to ZerehCache, our scheme achieves a considerably lower V_{dd}, power consumption, and area overhead. However, ZerehCache is a more suitable solution for single-bank caches and it does not need an address remapping technique to deal with capacity reduction. Overall, the inherent efficiency, high degree of freedom in redundancy replacement, and intelligent assignment of the spare elements are the main advantages of AP that allow it to tolerate a higher failure rate compared to the other techniques.

4.5 Summary

With aggressive CMOS scaling, dealing with power dissipation has become a challenging design issue. Consequently, a large amount of effort has been devoted to the development of DVS methods to tackle this problem. When decreasing the operational voltage of a modern microprocessor, large on-chip cache structures are the first components to fail. Tolerating these SRAM failures allows DVS to target lower V_{dd} values while preserving the core frequency scaling trend. In this work, we proposed a flexible fault-tolerant cache design, Archipelago, which benefits from a high degree of freedom in redundancy substitution and an intelligent configuration algorithm for redundancy allocation and group assignment. AP allows fault-free operation in the near-threshold region by partitioning the cache to multiple autonomous islands with various number of word-lines to minimize the cache capacity loss. Our scheme enables DVS to reach $375mV$ in $90nm$. This translates to 79% dynamic and 51% leakage power savings for our target system which is modeled after the Alpha 21364. This significant amount of saving comes with 2% area and 4.6% performance overhead for the microprocessor when operating in low-power mode. Finally, we compared our scheme with several conventional and state of the art methods to illustrate its efficiency and effectiveness.

CHAPTER V

Enhancing System Throughput by Animating Dead Cores

5.1 Introduction

Manufacturing defects is one of the main challenges for the semiconductor industry, which have a direct impact on yield. Based on the latest ITRS report [48], for current and near future CMOS technology, one manufacturing defect per five $100mm^2$ dies can be expected. Fortunately, a large fraction of die area is devoted to memory structures, in particular caches, which can be protected using existing techniques such as row/column redundancy, 2D-ECC [54], ZerehCache [11], Bit-Fix [103], and sub-block disabling [1]. With appropriate protection mechanisms in place for caches, the processing cores become the major source of defect vulnerability on the die. Consequently, we try to tackle hard-faults in the non-cache parts of the processing core. Due to the inherent irregularity of the general core area, it is well-known that handling defects in the non-cache parts is challenging [75]. A common solution is core disabling [6]. However, the industry is currently dominated by Chip Multi-Processor (*CMP*) systems with only a modest number of high-performance cores (e.g., Intel Core 2), systems which cannot afford to lose a core due to

manufacturing defects. The other extreme of the solution spectrum lies fine-grained micro-architectural redundancy [87, 25, 92]. Here, broken micro-architectural structures, such as ALUs, are isolated or replaced to maintain core functionality. Unfortunately, since the majority of the core logic is non-redundant, the fault coverage from these approaches is very limited – less than 10% for an Intel processor [75].

In this chapter, we propose Necromancer (*NM*) [9, 7] to tackle manufacturing defects in current and near future technology nodes. NM enhances overall system throughput and mitigates the performance loss caused by defects in the non-cache parts of the core. To accomplish this, we first relax the correct execution constraint on a faulty core – the *un-dead core* – since it cannot be trusted to faithfully execute programs. Next, we leverage high level execution information (*hints*) from the undead core to accelerate the execution of an *animator core*. The animator core is an additional core, introduced by NM, that is an older generation of the baseline cores in the CMP with less resources and the same instruction set architecture (*ISA*). The main rationale behind our approach is the fact that, for most defect instances, the execution flow of the program on the undead core *coarsely resembles* the fault-free program execution on the animator core – when starting from the same architectural state (i.e., program counter (*PC*), architectural registers, and memory). Moreover, in the animator core, these hints are only treated as performance enhancers and do not influence execution correctness. In NM, we rely on intrinsically robust hints and effective hint disabling to ensure the animator core is not mislead by unprofitable hints. Dynamic inter-core state resynchronization is also employed to update the undead core with valid architectural state whenever it strays too far from the correct execution path. To increase our design efficiency, we share each small animator core among multiple cores.

Our scheme is unique in the sense that it keeps the undead core on a semi-correct execution path, ultimately enabling the animator core to achieve a performance close to the performance of a live (fully-functional) core. In addition, NM does not noticeably increase the design complexity of the baseline cores and can be easily applied to current and near future CMP systems to enhance overall system throughput.

5.2 Utility of an Undead Core

We motivate the NM design by demonstrating the high-level rationale behind it. To this end, we provide evidence that supports the following two statements: **(1)** Although an aggressive out-of-order (*OoO*) core with a hard-fault in the non-cache area cannot be trusted to perform its normal operation, it can still provide useful execution hints in most cases. **(2)** By exploiting hints from the undead core, the animator core can typically achieve a significantly higher performance.

5.2.1 Effect of Hard-Faults on Program Execution

Prior work has studied the effect of a single-event upset, or a transient fault, on program execution for high-performance microprocessors. Using fault-injection, it has been shown that transient faults are often masked, easier to categorize, and have a temporal effect on program behavior [100]. On the other hand, the effect of hard-faults on program execution is hard to study since each hard-fault can result in a complicated intertwined behavior. For example, a hard-fault can cause multiple data corruptions that finally mask each others effect. Moreover, hard-faults are persistent and their effect does not go away. As a result,

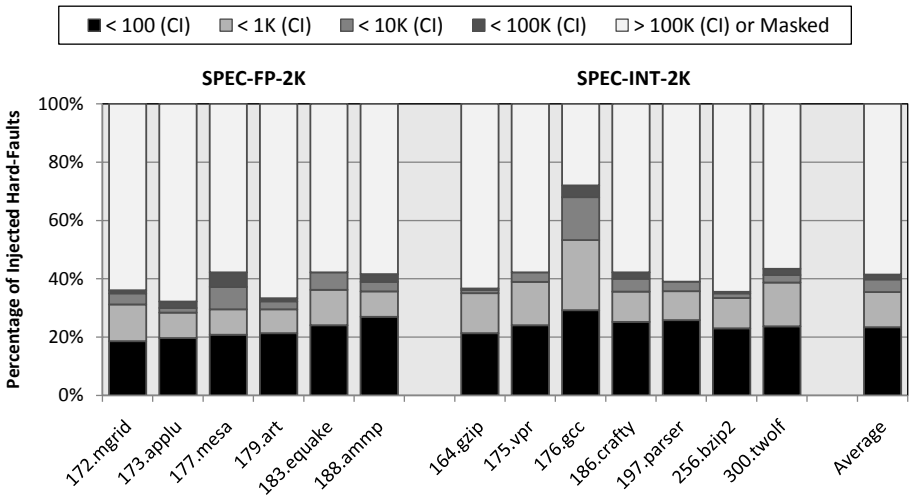


Figure 5.1: Distribution of injected hard-faults that manifest as architectural state mismatches across different latencies – in terms of the number of committed instructions (*CI*).

hard-faults can dramatically corrupt program execution. In order to illustrate the negative impact of hard-faults on program execution, we study the average number of instructions that can be committed before observing an architectural state mismatch. This result, for 5000 area-weighted hard-fault injection experiments across SPEC-CPU-2K benchmarks, is depicted in Figure 5.1.

Details of the Monte Carlo engine, statistical area-weighted fault injection infrastructure, target system, and benchmark suite can be found in Section 5.5.1. For these experiments, we have a golden execution which compares its architectural state with the faulty execution every cycle and as soon as a mismatch is detected, it stops the simulation and reports the number of committed instructions up to that point. For instance, looking at 188.amp, 26% of the injected hard-faults cause an architectural state mismatch to happen in less than 100 committed instructions. Since 176.gcc more uniformly stresses different core resources, it shows a higher vulnerability to hard-faults. As this figure shows, more than 40% of the injected hard-faults can cause an immediate – $<$ 10K – architectural state

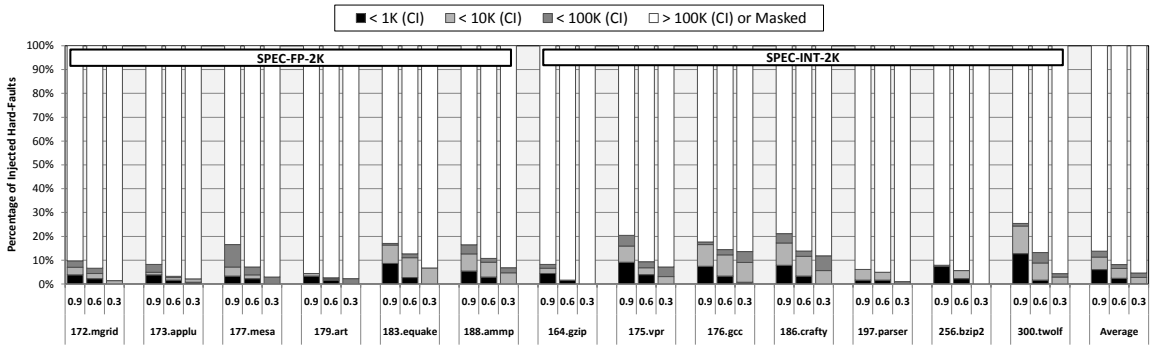


Figure 5.2: Number of instructions that are committed (*CI*) before an injected hard-fault results in a violation of a pre-specified similarity index threshold. For this purpose, 5K hard-faults were injected while considering three different similarity index thresholds (90%, 60%, and 30%).

mismatch. Thus, a faulty core cannot be trusted to provide correct functionality even for short periods of program execution.

5.2.2 Relaxing Correctness Constraints

As just discussed, program execution on a dead core cannot be trusted. Here, we try to determine the quality of program execution on a dead core when relaxing the absolute correctness constraints. In other words, we are interested in knowing for what expected level of correctness, a dead core can practically execute large chunks of a program. Based on 5K injected hard-faults, Figure 5.2 depicts how many instructions can be committed in a dead core before it gets considerably off the correct execution path. In order to have a practical system, the dead core should be able to execute the program over reasonable time periods before its execution becomes ineffectual. Here, we define a similarity index (*SI*) that measures the similarity between the PC of committed instructions in the dead core and a golden execution of the same program. This SI is calculated every 1K instructions

and whenever it becomes less than a pre-specified threshold, we stop the simulation and record the number of committed instructions. For instance, a similarity index of 30% for PC values means, that during each 1K instruction window, 30% of PCs hit exactly the same instruction cache line in both the golden execution and program execution on the dead core. Figure 5.2 shows the number of committed instructions for three different SI thresholds. For instance, considering SI threshold of 90%, on average only 12% of the hard-faults renders the program execution on a dead core ineffectual before at least 10K instructions get committed. Hence, even for an SI threshold of 90%, in more than 85% of cases, the dead core can successfully commit at least 100K instructions before its execution differs by more than 10%.

5.2.3 Opportunities for Acceleration

Since the execution behavior of a dead core coarsely matches the intact program execution for long time periods, we can take advantage of the program execution on the dead core to accelerate the execution of the same program on another core. This can be done by extracting useful information from the execution of the program on the dead core and sending this information (hints) to the other core (the animator core), running the same program. We allow the *undead core* to run without requiring absolutely correct functionality. The undead core is only responsible to provide helpful hints for the animator core. This symbiotic relation between the two cores enables the animator core to achieve a significantly higher performance. When the hints lose their effectiveness, we resynchronize the architectural state of the two cores. Since an architectural state resynchronization, between two cores in a CMP system, takes about 100 cycles [75] and resynchronization in more than

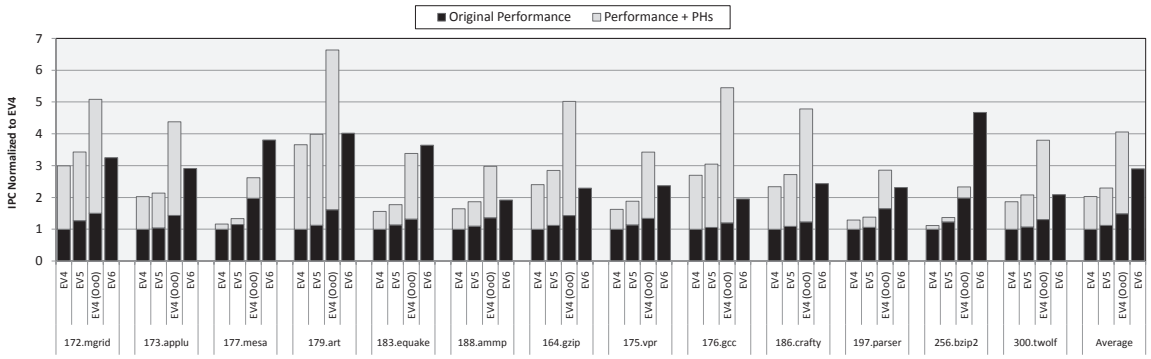


Figure 5.3: IPC of different DEC Alpha microprocessors, normalized to EV4's IPC. In most cases, by providing perfect hints for the simpler cores (EV4, EV5, and EV4 (OoO)), these cores can achieve a performance comparable to that achieved by a 6-issue OoO EV6.

85% of cases happens after at least 100K committed instructions, the overhead associated with resynchronization is small.

For the purpose of evaluation and since we want to have a single ISA system, based on the availability of the data on the power, area, and other characteristics of microprocessors, we use an EV6 (DEC Alpha 21264 [52]) for the baseline cores. On the other hand, for the animator core, we select a simpler core like the EV4 (DEC Alpha 21064) or EV5 (DEC Alpha 21164) to save on the overheads of adding this extra core to the CMP system. In order to evaluate the efficacy of the hints, in Figure 5.3, we show the performance boost for the aforementioned DEC Alpha cores using perfect hints (*PHs*) – perfect branch prediction and no L1 cache miss. Here, we have also considered the EV4 (OoO), an OoO version of the 2-issue EV4, as a potential option for our animator core. As can be seen, by employing perfect hints, the EV4 (OoO) can outperform the 6-issue OoO EV6 in most cases; thus, demonstrating the possibility of achieving a performance close to the performance of a live core through the NM system. Nevertheless, achieving this goal is quite challenging due to the presence of defects, different sources of imperfection in hints, and inter-core

communication issues.

5.3 From Traditional Coupling to Animation

In a CMP system, prior work has shown two cores can be coupled together to achieve higher single-thread performance. Since the overall performance of a coupled core system is bounded by the slower core, these two cores were traditionally identical to sustain an acceptable level of single-thread performance. However, in order to accelerate program execution, one of these coupled cores must progress through the program stream faster than the other. In order to do so, three methods have been proposed:

- In Paceline [40], the core that runs ahead (*leader*) and the core that receives execution hints (*checker*) from the leader core operate at different frequencies. Paceline cuts the frequency safety margin of the leader core and continuously compares the architectural state (excluding memories) of the two cores. When a mismatch happens, the frequency of the leader is adjusted, L1 state match is enforced, and finally the checkpoint interval is rolled back for re-execution.
- Slipstream processors [76] and Master/Slave speculative parallelization [111] need two different versions of the same program. In these schemes, the leader core runs a shorter version of the program based on the removal of ineffectual instructions while the checker core runs the unmodified program.
- Finally, Flea-Flicker two pass pipelining [15] and Dual-Core Execution [110] allow the leader core to return an invalid value on long-latency operations and proceed.

Although these schemes have widely varying implementation details, they share some common traits. In these schemes, the leader core tries to get ahead and sends hints that can accelerate checker core execution. These two cores are connected through one/several first-in first-out (*FIFO*) hardware queues to transfer hints and retired instructions along with their PCs. The checker core takes advantage of program execution on the leader core in 3 ways. First, the checker core receives pre-processed instruction and data streams. Second, during the program execution in the leader core, most branch mispredictions get resolved. Third, the program execution in the leader core automatically initiates L2 cache prefetches for the checker core.

A straight-forward extension of these ideas to animate a dead core seems plausible. However, NM encounters major difficulties when trying to fit the dead core into this execution model. Here, we briefly describe the two main challenges, leaving discussions of the proposed microarchitectural solutions for subsequent sections.

Fine-Grained Variations: One of the main sources of problems is the presence of defects in the dead core. Due to the presence of defects, the *undead core* might execute/commit more or less number of instructions, causing variations in the similarity of program executions between the two cores. For instance, in many cases, the undead core can take the wrong direction on an IF statement and get back to the right execution path afterwards, thereby preventing a perfect data or instruction stream for the animator core. This necessitates employing generic hints that are more resilient to these local abnormalities. Moreover, the number of times that each PC is visited cannot be used to synchronize the two cores. A mechanism is required to help the animator core identify the proper time for pulling the hints off the communication queue. Given the variation in the usefulness

of the hints, in order to enhance the efficiency of the animator core, fine-grained hint disabling can be leveraged. For instance, if the last K branch prediction hints for a particular PC were not useful, branch prediction for this particular PC can be handled by the animator core's branch predictor.

Global Divergences: When the undead core gets completely off the correct execution path, hints become useless, and it needs to be brought back to a valid execution point. For this purpose, the architectural state of the animator core can be copied over to the undead core. Although exact state matching, by checkpointing the register file, has been used in prior work [40], it is not applicable for animating a dead core since architectural state mismatches occur so frequently. Therefore, we need coarse-grained online monitoring of the effectiveness of the hints over a large time period to decide whether the undead core should be resynchronized with the animator core. Moreover, resynchronizations should be cheap and relatively infrequent to avoid a noticeable impact on the overall performance of the animator core. One possible approach for maintaining correct memory state, suggested by Paceline, is to re-fetch the cache-lines that are accessed during the last checkpointed interval into the L1 cache of the leader core [40]. However, since this might happen often for a dead core, we need a low-cost resynchronization approach that does not require substantial book keeping.

5.4 NM Architecture

The main objective of NM is to mitigate system throughput loss due to manufacturing defects. For this purpose, it leverages a robust and flexible heterogeneous core coupling

execution technique which will be discussed in the rest of this section. Given a group of cores, we introduce an animator core, an older generation with the same ISA, that is shared among these cores for defect tolerance purposes. In this section, we describe the architectural details for a coupled pair of dead and animator cores. The high-level NM design for a CMP system with more cores will be discussed in the next section. In Section 5.2, we showed that the faulty core – the undead core – cannot be trusted to run even a short part of the program. However, as we relaxed the exact architectural state match and looked at the global execution pattern, the undead core can execute a moderate portion of the program before a resynchronization is required. By executing the program on the undead core, NM provides hints to accelerate the animator core without requiring multiple versions of the same program. In other words, the undead core is used as an external run-ahead engine for the animator core that has been added to the CMP system. We believe NM is a valuable solution for improving the system throughput of the current and near future mainstream CMP systems without notably influencing design complexity.

5.4.1 High-Level NM System Description

Figure 5.4 illustrates the high-level NM heterogeneous coupled core design. As discussed in Section 5.2, for the purpose of evaluation, we use 6-issue OoO EV6 for the baseline cores and a 2-issue OoO EV4 as our animator core. In our design, most communications are unidirectional from the undead core to the animator core with the exception of the resynchronization and hint disabling signals. Thus, a single queue is used for sending the hints and cache fingerprints to the animator core. The hint gathering unit attaches a 3-bit tag to each queue entry to indicate its type. When this queue gets full and the undead

core wants to insert a new entry, it stalls. To preserve correct memory state, we do not allow the dirty lines of the undead core's data cache to be written back to the shared L2 cache. As a result, a dirty data cache-line of the undead core is simply dropped whenever it requires replacement. Exception handling is also disabled at the undead core since the animator core maintains the precise state.

As discussed in Section 5.2, the animator core with perfect hints has the potential of surpassing the average performance of a live core. Nonetheless, the performance of the undead core can be a bottleneck for the NM system since: **a.** In many cases (Figure 5.3), performance of a baseline core is worse than the performance of the animator core with perfect hints. **b.** After each resynchronization, the undead core needs to warm-up the branch predictor and local caches. Therefore, we allow the undead core to proceed on the data cache L2 misses, without waiting for the several hundred cycles needed to receive data back from main memory. We simply return zero since L2 misses are not common and also value prediction would not be beneficial. This has a large impact on the performance of

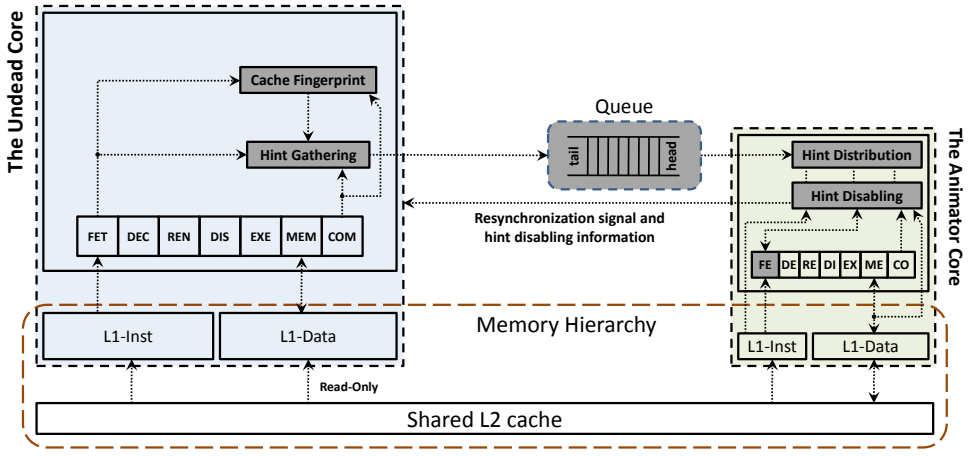


Figure 5.4: The high-level architecture of NM is shown in this figure and modules that are modified or added to the underlying cores are highlighted (not drawn to scale).

the undead core, potentially shortening the resynchronization period. Given the ability to eliminate stalls on L2 misses and also semi-perfect hints from the undead core, NM can *potentially* achieve even a higher performance than that of a live core. Nevertheless, providing even semi-perfect hints is challenging due to defects in the undead core, queue size, limited performance of the undead core, queue delay, and natural fluctuations in program behavior.

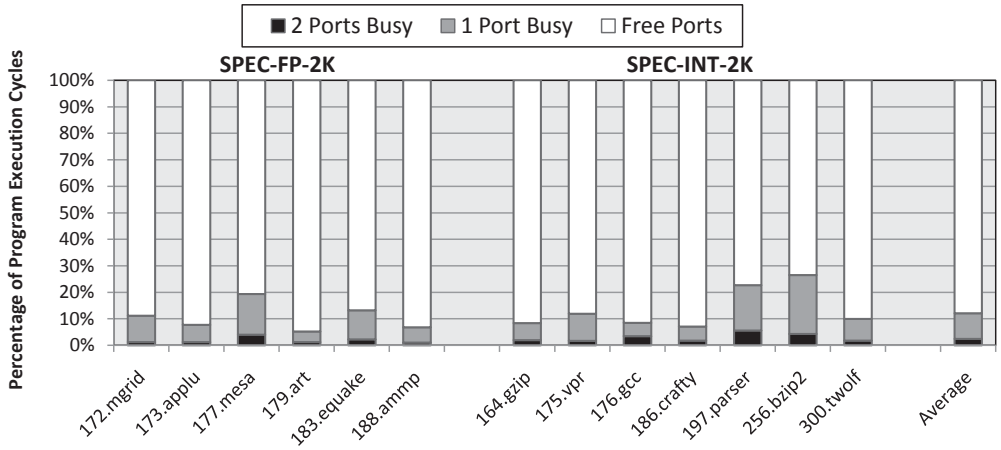
NM uses a heterogeneous core coupling program execution with a pruned core that has a significantly smaller area compared to a baseline core. In NM, we do not rely on overclocking the undead core or having multiple versions of the same program. Furthermore, it is a hardware-based approach that is transparent to the workload and operating system (*OS*). It also does not require register file checkpointing for performing exact state matching between two cores. Instead, we employ a fuzzy hint disabling approach based on the continuous monitoring of the hints effectiveness, and initiating resynchronizations when appropriate. Hint disabling also helps to enhance performance and save on communication power for program phases in which the undead core cannot get ahead of the animator core. Apart from that, the undead core might occasionally get off the correct execution path (e.g., taking the wrong direction on an IF statement) and return to the correct path afterwards – Y-branches [99]. In order to make the hints more robust against microarchitectural differences between two cores and also variations in the number/order of executed instructions, we leverage the number of committed instructions for hint synchronization and attach this number to every queue entry as an *age tag*. Moreover, we introduce the *release window* concept to make the hints more robust in the presence of aforementioned variations. For a particular hint type, the release window helps the animator core to determine the right time

to utilize a hint. For instance, assuming the data cache (*D-cache*) release window is 100, and 1000 instructions have already been committed in the animator core, D-cache hints with age tags ≤ 1100 can be pulled off the queue and applied.

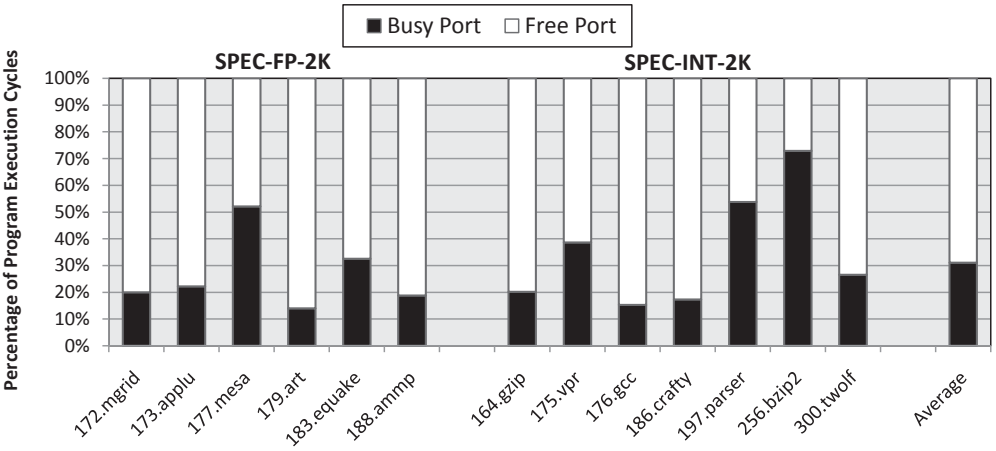
5.4.2 Hint Gathering and Distribution

Program execution on the undead core automatically warms-up the shared L2 cache without requiring communication between two cores. However, other hints – i.e., L1 data cache, L1 instruction cache, and branch prediction hints – need to be sent through the queue to the animator core. The hint gathering unit in the undead core is responsible for gathering hints and cache fingerprints, attaching the age and 3-bit type tags, and finally inserting them into the queue. On the other side, the hint distribution unit receives these packets and compares their age tag with the local number of committed instructions plus the corresponding release window sizes.

Every cycle, the hint gathering unit looks over the committed instructions for data and instruction cache (*I-cache*) hints. In fact, the PC of committed instructions and addresses of committed loads and stores are considered as I-cache and D-cache hints, respectively. On the animator core side, the hint distribution unit treats the incoming I-cache and D-cache hints as prefetching information to warm-up its local caches. For the animator core, Figure 5.5 depicts the utilization of two D-cache ports and a single I-cache port. Given the pipelined cache access for all high-performance processors, as can be seen for D-cache, both ports are busy for less than 5% of cycles. Therefore, we leverage the original cache ports for applying our D-cache hints. However, since hints can only *potentially* help the program execution, priority of the access should always be given to the normal operation of



(a) Port activity for the animator core's L1-data cache



(b) Port activity for the animator core's L1-instruction cache

Figure 5.5: Port activity breakdown for local caches of the animator core. Here, we show the percentage of cycles that each cache port is either busy or free. For our animator core, the data cache has 2 ports while the instruction cache has a single port.

the animator core. On the other hand, the I-cache port is busy for more than 50% of cycles for 3 benchmarks and is free only if the instruction fetch queue (*IFQ*) is full. Moreover, since the I-cache operation is critical for having a sustainable performance, we add an extra port to this cache in the animator core.

In order to provide branch prediction hints, the hint gathering unit looks at the branch predictor (*BP*) updates and every time the BP of the undead core gets updated, a hint will

be sent through the queue. In the animator core side, the default BP – for EV4 – is a simple bimodal predictor. We firstly add an extra bimodal predictor (*NM BP*) to keep track of incoming branch prediction hints. Furthermore, we employ a hierarchical tournament predictor to decide for a given PC, whether the original or NM BP should take over. During our design space exploration, the size of these structures will be determined – Section 5.5.2. As mentioned earlier, we introduced release window size to get the hints just before they are needed. However, due to the variations in the number of executed instructions on the undead core, even the release window cannot guarantee the perfect timing of the hints. In such a scenario, for a subset of instructions, the tournament predictor can give an advantage to the original BP of the animator core to avoid any performance penalty. Having this in mind, Figure 5.6 shows a simple example in which the NM BP can only achieve 33% branch prediction accuracy. This is mainly due to the existence of a tight inner loop – number of instructions in the loop body is less than BP release window size – with a low trip count. Switching to the original BP can enhance the overall branch prediction accuracy for this code region.

Another aspect of the NM dual core execution is the potential of hints on the speculative execution paths. If a speculative path turns to be a correct path, instructions on this path will eventually be committed and the corresponding hints will be sent to the animator core. On the other hand, for a wrong path, although sending hints can potentially accelerate the execution of speculative paths on the animator core, this acceleration can only decrease the efficiency of our hints for the correct paths. For instance, if the animator core executes a wrong path faster, it will bring more useless data to its local D-cache which causes prefetched data for non-speculative paths to be dropped out of D-cache. Therefore,

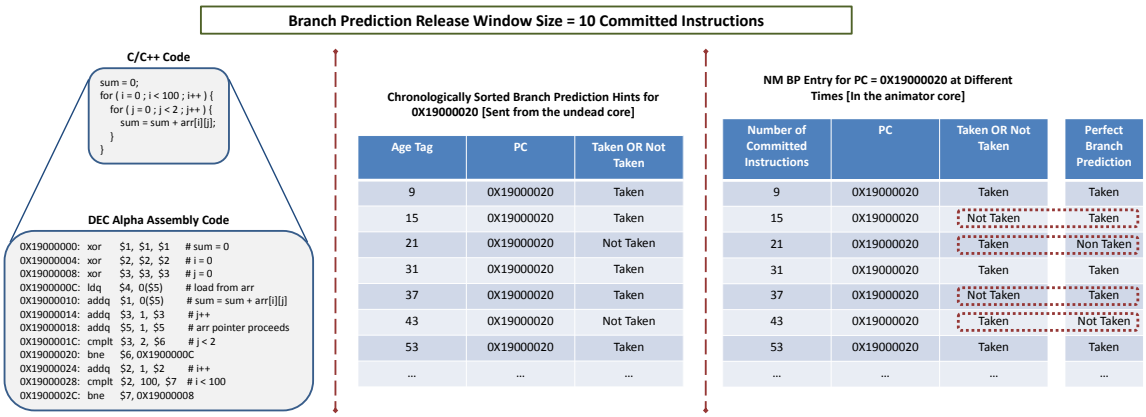


Figure 5.6: A code example in which the NM BP performs poorly and switching to the original BP of the animator core is required. The code simply calculates the summation of a 2D-array elements which are stored in a row-based format. It should be noted that the branch prediction release window size is normally set so that the branch prediction accuracy for the entire execution gets maximized. As can be seen, hints are received by the animator core at improper times, resulting in low branch prediction accuracy.

it is clear that sending hints for speculative paths can merely hurt the performance of the NM system.

5.4.3 Reducing Communication Overheads

In order to reduce the queue size, communication traffic needs to be limited to more beneficial hints. Consequently, in the hint gathering unit, we use two content addressable memories (*CAM*s) with several entries to discard I-cache and D-cache hints that were recently sent. Eliminating redundant hints also minimizes the resource contention on the animator core side. For this purpose, these two CAMs keep track of the last N – number of CAM entries – committed load/store addresses in the undead core. In addition to sending less number of hints, queue size can be reduced by sending less bits per hint. Saving on the number of bits can be done in several ways: sending only the block related bits of address for I-cache and D-cache hints, ignoring hints on the speculative paths, and for branch pre-

diction hints, only sending lower bits of the PC that are used for updating branch history table of the NM BP.

Given a design with multiple communication queues, the undead core stalls when at least one queue is full and it wants to insert a new entry to that queue. The other queues that are not full during these stalls remain underutilized; thus, using a single aggregated queue guarantees a higher utilization, which reduces the area overhead, number of stalls, and overheads of interconnection wires. On the other hand, since a single queue is used, multiple entries might need to be sent to or received from the queue at the same cycle. This can be solved by grouping together several hints with the same age tag and sending them as a single packet over the queue. This requires a small buffer in the hint distribution unit to handle the case that hints have non-identical release windows sizes.

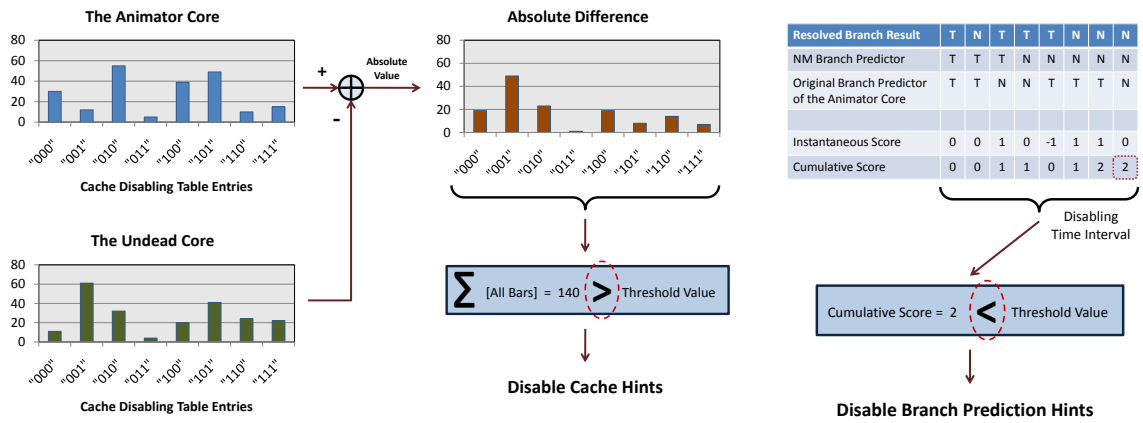
5.4.4 Hint Disabling Mechanisms

Hints can be disabled when they are no longer beneficial for the animator core. This might happen because of several reasons. First, the program execution on the undead core gets off the correct execution path due to the destructive impact of defects. Second, in certain phases of the program, performance of the animator core might be close to its ideal case, attenuating the value of hints. Lastly, at certain parts of the program, due to the intertwined behavior of the NM system, the animator core might not be able to get ahead of the undead core. In all these scenarios, hint disabling helps in four ways:

- It avoids occupying resources of the animator core with ineffective hints that does not buy any performance benefit.
- The queue fills up less often which means less number of stalls for the undead core.

- Disabling hint gathering and distribution saves power and energy in both sides.
- It serves as an indicator of when the undead core has strayed far from the correct path of execution (i.e., when hints are frequently disabled) and resynchronization is required.

The hint disabling unit is responsible for realizing when each type of hint should get disabled. In order to disable cache hints, the cache fingerprint unit generates high-level cache access information based on the committed instructions in the last disabling time interval – e.g., last 1K committed instructions. These fingerprints are sent through the queue and compared with the animator core's cache access pattern. Based on a pre-specified threshold value for the similarity between access patterns, the animator core decides whether the cache hint disabling should happen. In addition, when a hint gets disabled, that hint remains disabled during a time period called the back-off period. More precisely, the cache fingerprint unit retains two tables for keeping track of non-speculative I-cache and D-cache accesses in the last disabling time interval. Figure 5.7(a) illustrates an example of cache disabling. Considering D-cache hints, the corresponding table has only several entries – 8 entries in our example – and each entry will be incremented for a committed load/store, whenever the LSBs of the address match the rank order of that entry. Therefore, the cache disabling table maintains a high-level distribution of addresses that are accessed during the last interval. At the end of each interval, the table contents will be sent over the queue to the animator core and entries will be cleared for the next interval. Given a similar cache access distribution at the animator core's side, for evaluating similarity between two distributions, $(V_1, V_2, ..., V_{16})$ for the undead core and $(S_1, S_2, ..., S_{16})$ for the animator core, we calculate $K = \sum_{i=1}^{16} |S_i - V_i|$. Then, if K (140 in our example) is less than a pre-specified threshold,



(a) Disabling cache hints

(b) Disabling branch prediction hints

Figure 5.7: Two high-level examples of cache and branch prediction hint disabling mechanisms. Here, values on the X-axes of the plots correspond to eight entries of the cache disabling table.

a signal will be sent to the undead core to stop gathering that particular hint for the back-off period.

Disabling branch prediction hints can solely be done by the animator core. Apart from prioritizing the original BP of the animator core for a subset of PCs, the NM BP can be also employed for global disabling of the branch prediction hints. For this purpose, we continuously monitor the performance of the NM BP and if this performance – compared to the original BP – is worse than a pre-specified threshold for the last disabling time interval, we disable branch prediction hints. As Figure 5.7(b) depicts, for branch prediction hint disabling, we use a score-based scheme with a single counter. For every branch that the original and NM BPs either both correctly predict or both mispredict no action should be taken. Nonetheless, for the branches that the NM BP correctly predicts and the original BP does not, the score counter is incremented by one. Similarly, for the ones that NM BP mispredicts but the original BP correctly predicts, the score counter is decremented. Finally, at the end of each disabling time interval, if the score counter (2 in our example)

is less than a certain threshold, the branch prediction hints will be disabled for the back-off period. For performing infrequent disabling-related computations, we add a 4-bit ALU to the hint disabling unit.

5.4.5 Resynchronization

Since the undead core might get off the correct execution path, a mechanism is required to take it back to a valid architectural state. In order to do so, we use resynchronization between the two cores during which the animator core's PC and architectural register values get copied to the undead core. According to [75], for a modern processor, the process of copying PC and register values between cores takes on the order of 100 cycles. Moreover, all instructions in the undead core's pipelines are squashed, the rename table is reset, and the D-cache content is also invalidated for "resynchronizing" the memory state.

Resynchronization should happen when the undead core gets off the correct execution path and it can no longer provide useful hints for the animator core. The simplest policy is to resynchronize every N committed instructions where N is a constant number like 100K. However, as we will show in Section 5.5.2, a more dynamic resynchronization policy can achieve a higher overall speed-up for the NM system. We take advantage of the hint disabling information to identify when resynchronization should happen. An aggressive policy is to resynchronize every time a hint gets disabled. However, such a policy results in too many resynchronizations in a short time which clearly reduces the efficiency of our scheme. Another potential policy is to resynchronize only if at some point in time all or at least two of the hints get disabled. Later in Section 5.5.2, we will compare some of these potential resynchronization policies.

5.4.6 NM Design for CMP Systems

So far, we described the NM heterogeneous coupled core execution approach and its architectural details. Here, NM for CMP systems will be discussed. Figure 5.8 illustrates the NM design for a 16-core CMP system with 4 clusters modeled after the Sun Rock processor. Each cluster contains 4 cores which share a single animator core, shown in the call-out. In order to maintain scalability of the NM design, we employ the aforementioned 4-core cluster design as the building block. Although a single animator core might be shared among more cores, it introduces long interconnection wires that should travel from one corner of the die to another. Therefore, given the low area overhead of NM for a 4-core CMP (5.3% as will be discussed in Section 5.5.2), the proposed building block preserves design scalability. On the other hand, since many dies are fault-free, in order to avoid disabling the animator cores, these cores can be leveraged for accelerating the operation of live cores. One possibility is to use the animator cores to exploit Speculative Method-Level Parallelism by spawning an extra thread and moving it to the animator core to execute the method call. The original thread executes the code that follows the method's return by leveraging a return value predictor. This is based on the observation that inter-method dependency violations are infrequent. However, evaluation of the latter is beyond the scope of this work.

For a heterogeneous CMP system, the problem is slightly more difficult due to the inherent diversity of the cores. Therefore, sharing an animator core between multiple cores might not be possible since those cores have different computational capabilities. A potential solution is to partition the CMP system to groups of cores in which each group contains

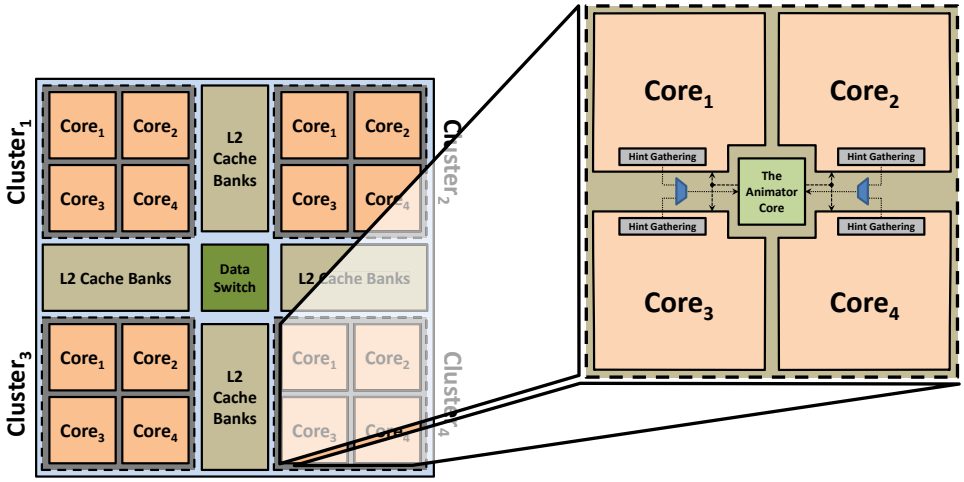


Figure 5.8: The high-level NM design for a large CMP system with 16 cores, modeled after the Sun Rock processor, which has 4 cores per cluster. The details of NM core coupling can be found in Figure 5.4.

cores with similar characteristics and performance. Therefore, each group can share an animator core with different specifications. An alternative is to partition the cores to groups such that in each group, we have several large cores and a small core – all from the original set of heterogeneous cores. In each group, the smaller core should have the capability of operating as a conventional core or as an animator core when there is a defect in one of the larger cores in its own group. These dual purpose cores are a suitable fit for many heterogeneous CMP systems that come with a bunch of simpler cores such as the IBM Cell processor.

In our design, since the animator core is shared among multiple cores, it is reasonable to shift the overheads to the animator core side to avoid replicating of the same module in the baseline cores. For instance, most of the similarity matching structures for hint disabling are located on the animator core side. Furthermore, since the undead core runs significantly ahead of the animator core in the program stream, the communication queue should also be closer to the animator core to reduce the timing overhead of accessing the queue and

checking the age tags. Finally, disabling hints, when they are no longer beneficial, allows the undead core to avoid gathering and sending the hints which saves power/energy on both sides.

5.5 Evaluation

In this section, we describe experiments performed to quantify the potential of NM in enhancing the system throughput.

5.5.1 Experimental Methodology

In order to model NM's heterogeneous coupled core execution, we heavily modified SimAlpha, a validated cycle accurate microarchitectural simulator based on SimpleScalar [14]. We run two different versions of the simulator, implementing the undead and animator cores, and use inter process communication (*IPC*) to model the information flow

Table 5.1: The target NM system configuration

Parameter	**The animator core**	**A baseline core**
Fetch/issue/commit width	2 per cycle	6 per cycle
Reorder buffer entries	32	128
Load/store queue entries	8/8	32/32
Issue queue entries	16	64
Instruction fetch queue	8 entries	32 entries
Branch predictor	tournament (bimodal + NM BP)	tournament (bimodal + 2-level)
Branch target buffer size	256 entries, direct-map	1024 entries, 2-way associative
Branch history table entries	1024	4096
Return address stack entries	-	32
L1 data cache	8KB DM, 3 cycles, 2 ports	64KB 4-way, 5 cycles, 4 ports
L1 instr. cache	4KB DM, 2 cycles, 2 ports	64KB 4-way, 5 cycles, 1 port
L2 cache	2MB Unified, 8-way, 15 cycles hit latency, 1 port	
Main memory	250 cycles access latency	

between two cores (e.g., L2 warm-up, hints, and cache fingerprints). As mentioned earlier, a 6-issue OoO EV6 and a 2-issue OoO EV4 are chosen as our baseline and animator cores, respectively. The configuration of these two coupled cores and the memory system is summarized in Table 5.1. We simulate the SPEC-CPU-2K benchmark suite cross-compiled for DEC Alpha and fast-forwarded to an early SimPoint [85].

To study the effect of manufacturing defects on the NM system, we developed an area-weighted, Monte Carlo fault injection engine. During each iteration of Monte Carlo simulation, a microarchitectural structure is selected and a random single stuck-at fault is injected into the timing simulator. Table 5.2 summarizes the fault locations used in our experi-

Table 5.2: Fault injection locations and their corresponding pipeline stages along with stage-level area break-down for EV6.

Pipeline Stage	Area Break-down	Fault Location
Fetch	14.3%	Program counter
		Branch target buffer
		Instruction fetch queue (instruction bits)
		Instruction fetch queue (PC bits)
Decode	15.6%	Input latch of decoder (instr. opcode bits)
		Input latch of decoder (instr. source register bits)
		Input latch of decoder (instr. destination register bits)
Rename	5.1%	Rename alias table
Dispatch	24.1%	Integer register file
		Floating point register file
		Reorder buffer
Backend	40.8%	Integer ALU
		Integer multiplier
		Integer divider
		Floating point ALU
		Floating point multiplier
		Floating point divider
		Load/store queue

ments. Since every transistor has the same probability of being defective, hard-fault injections should be distributed across microarchitectural structures in proportion to their area. Therefore, for each fault injection experiment, we inject 5000 hard-faults while artificially prioritizing structures that have larger area. These stuck-at faults are injected one by one in the course of each individual experiment. As a result, at any point in time, there is a single stuck-at fault in the undead core. Given an operational frequency of 600MHz [59] for EV6 in 0.35μm, scaling to a 90*nm* technology node would result in a frequency of 2.3*GHz* at 1.2V. This frequency is a pessimistic value for the animator core and NM can clearly achieve even better overall performance if the animator core were allowed to operate at a higher frequency. Nevertheless, since the amount of work per pipeline stage remains relatively consistent across Alpha microprocessor generations [59], for a given supply voltage level and a technology node, the peak operational frequency of these different cores are essentially the same.

Dynamic power consumption for both cores is evaluated using Wattch [26] and leakage power is evaluated with HotLeakage [109]. Area for our EV6-like core – excluding the I/O pads, interconnection wires, the bus-interface unit, L2 cache, and control logic – is derived from [59]. In order to derive the area for the animator core, we start from the publicly available area break-down for the EV6 and resize every structure based on the size and number of ports. Furthermore, CACTI [72] is used to evaluate the delay, area, and power of the on-chip caches. Overheads for the SRAM memory structures that we have added to the design, such as the NM branch prediction table, are evaluated with the SRAM generator module provided by the 90*nm* Artisan Memory Compiler. Moreover, the Synopsys standard industrial tool-chain, with a TSMC 90*nm* technology library, is used to

evaluate the overheads of the remaining miscellaneous logic (e.g., MUXes, shift registers, and comparators). Finally, the area for interconnection wires between the coupled cores is estimated using the same methodology as in [60], with intermediate wiring pitch taken from the ITRS road map [48].

5.5.2 Experimental Results

In this section, we evaluate different aspects of the NM design such as design space, achievable speed-up in the presence of defects, performance impact of different hard-fault locations, area and power overheads, and finally throughput enhancement.

Design Space Exploration: Here, we fix the architectural parameters that are involved in the NM design. Since there is a variety of parameters (both hardware and policy), due to space considerations, we only present a subset of the exploration for parameters with the most interesting behaviors. During the exploration, we initially assign a nominal value to each of the parameters and as we select a proper value for each parameter, we use the updated value for the reminder of the experiments. Figures 5.9 to 5.14 depict this design space exploration for a pruned set of NM parameters.

In Figure 5.9, the release window size is varied between 0 to 256 committed instructions while monitoring the data cache miss rate of the animator core. As can be seen, there is an optimal window size (i.e., 16 committed instructions) that maximizes prefetching efficiency, given the variations in the number of committed instructions on the undead core. The D-cache miss rate, even before optimizing other parameters, is reduced from 10.7% to 5.3%. Figure 5.10 illustrates the effect of reducing the branch history table (*BHT*) size of the NM BP on the branch prediction accuracy of the animator core. To save area, we limit

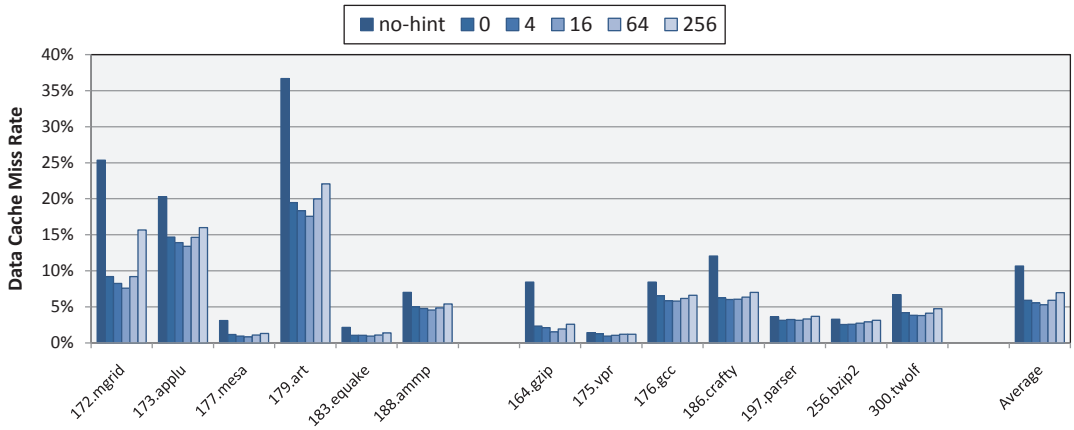


Figure 5.9: Effect of the NM D-cache release window size on the data cache miss rate of the animator core.

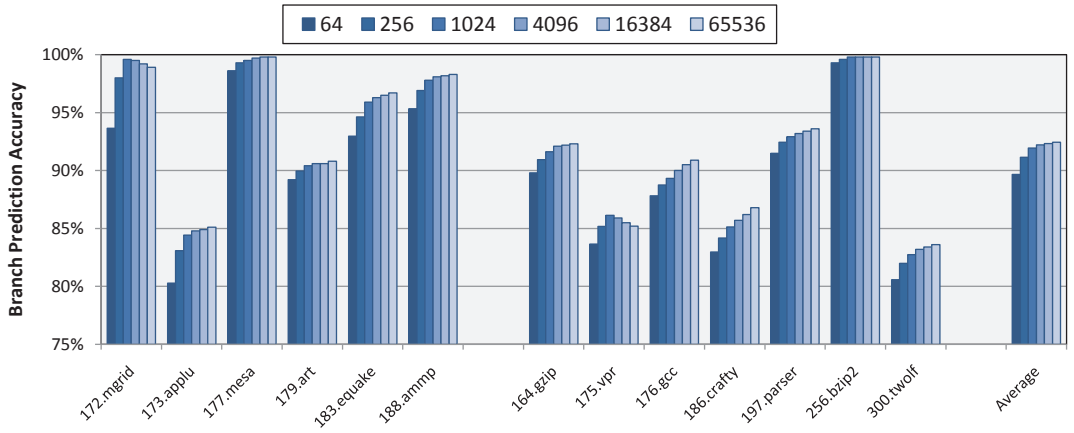


Figure 5.10: Effect of the branch history table size of the NM BP on the overall branch prediction accuracy of the animator core.

the BHT size to 1024 entries, causing less than 0.5% reduction in the achievable branch prediction accuracy.

The size of the D-cache hint CAM is a double-edged sword and its impact on the D-cache miss rate and communication traffic is shown in Figure 5.11. Increasing the CAM size, reduces the communication traffic and queue size. However, this aggravates the efficiency of D-cache hints. The reason is that sending more up-to-date hints increases the likelihood that data is present in the local D-cache of the animator core when it is needed.

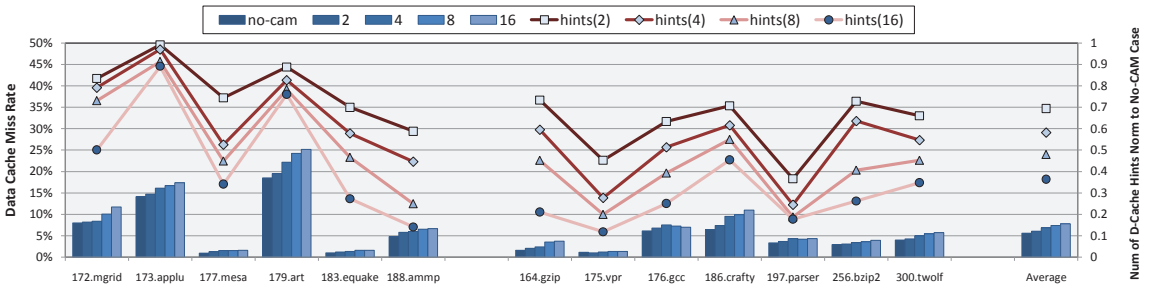


Figure 5.11: Effect of CAM size that are used for reducing the number of D-cache hints – generated in the undead core – on the data cache miss rate of the animator core. Here, the lines show the number of data cache hints should be sent to the animator core per cycle, normalized to the the case without any CAM.

Nevertheless, using a CAM with 2 entries can reduce the number of transmitted D-cache hints by more than 30% while affecting the D-cache miss rate by less than 0.5%. Next, Figure 5.12 illustrates the effect of varying the threshold for disabling branch prediction hints. For each injected hard-fault and benchmark, we record the number of instructions committed before the branch prediction hint is disabled. Results of this process are depicted for 3 different threshold values (i.e., 50%, 70%, and 90% similarities). For high similarity requirements, such as 90%, the branch prediction hints are mostly disabled even before 5K instruction are committed in the animator core. Consequently, we select 70% similarity so that the hint disabling does not occur too frequently while still receiving occasional feedback about the effectiveness of the hints during program execution.

Finally, Figures 5.13 and 5.14 show the impact of different resynchronization policies and communication queue sizes on the achievable speed-up by NM, respectively. In these two plots, speed-ups are normalized to the performance of a baseline animator core. We consider 4 candidates for the resynchronization policy, consisting of one static and 3 dynamic polices. For the static policy, resynchronization occurs periodically after committing

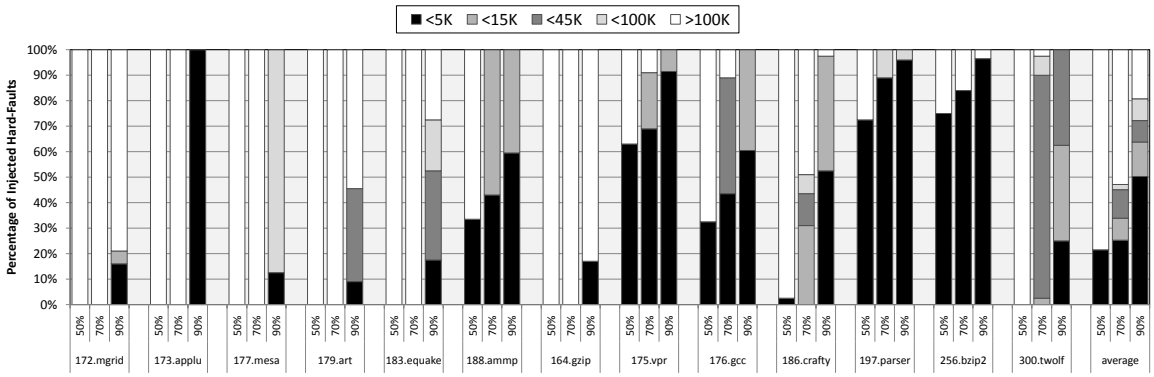


Figure 5.12: Number of instructions committed in the animator core before the branch prediction hint is disabled for different pre-specified branch prediction hint disabling thresholds (i.e., 50%, 70%, and 90% similarities).

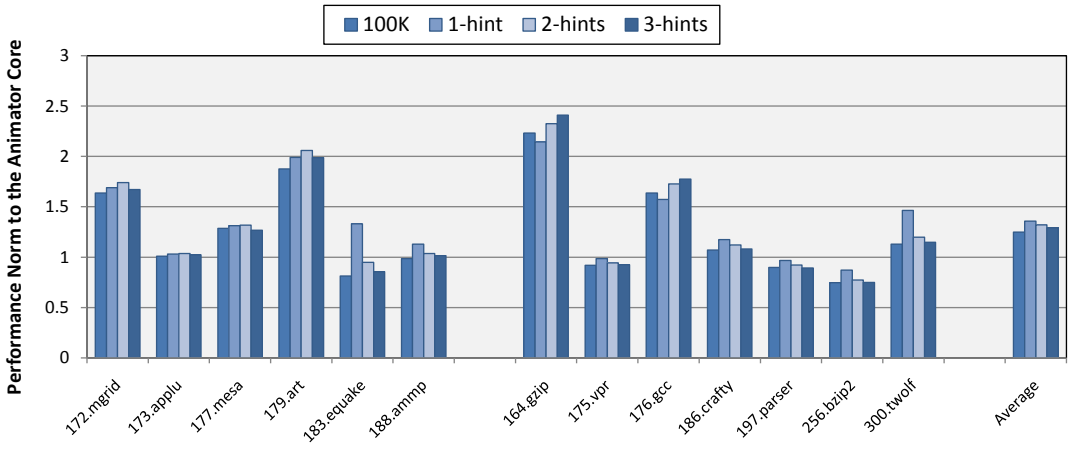


Figure 5.13: Effect of different resynchronization policies on the overall speed-up of the NM coupled cores normalized to the performance of the baseline animator core.

100K instructions while for the dynamic policies, the number of disabled hints determines whether resynchronization is required. Since we aggressively exploit the hints by rarely disabling them, the resynchronization policy that is invoked on the first disabled hint achieves a better speed-up. Finally, the sensitivity to the communication queue size is presented in Figure 5.14. Although it seems that a larger queue is always better, an extremely large queue enables the undead core to get too far ahead of the animator core, polluting the L2 cache with unprofitable prefetches.

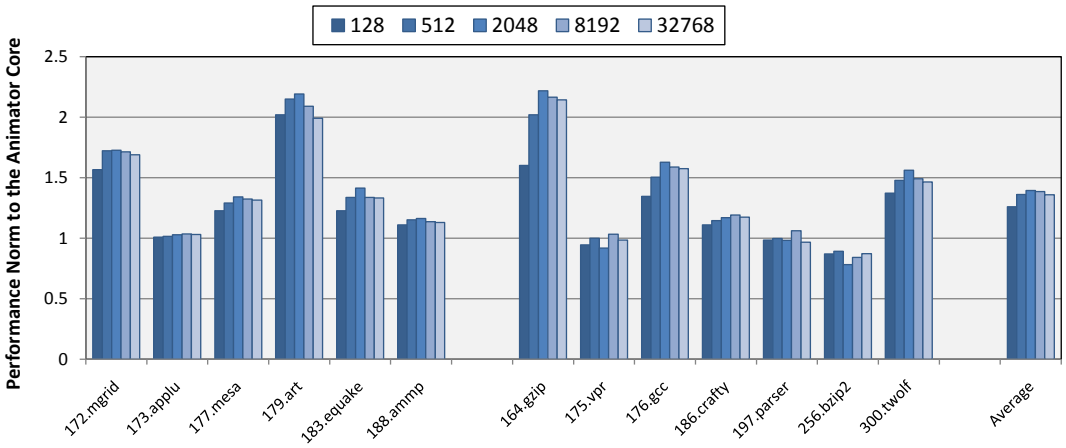


Figure 5.14: Effect of communication queue size on the overall speed-up of the NM coupled cores normalized to the performance of the baseline animator core.

The values for the remaining parameters were identified in a similar fashion: I-cache release window size (4 committed instructions), branch prediction release window size (4 committed instructions), I-cache hint CAM size (2 entries), branch prediction hint disabling threshold (70% similarity), D-cache hint disabling threshold (70% similarity), I-cache hint disabling threshold (80% similarity), D-cache hint disabling table size (32 entries), and I-cache hint disabling table size (32 entries). Given these parameter values, on average, NM can achieve 39.5% speed-up over the baseline animator core. In our simulation, we set the queue delay to 15 cycles – same as L2 cache; however, since the NM coupled core design is highly pipelined, it has a minimal sensitivity to the queue delay. For instance, even setting this delay to 45 cycles, only affects the final speed-up by less than 1%.

Performance Impact of Different Hard-Fault Locations: In order to highlight the impact of a fault location on the achievable speed-up by the NM system, Figure 5.15 depicts the performance breakdown results for the fault locations described in Table 5.2. Results in each row of this plot is normalized to the average speed-up that can be achieved by the NM coupled core for that particular benchmark. This was done to eliminate the

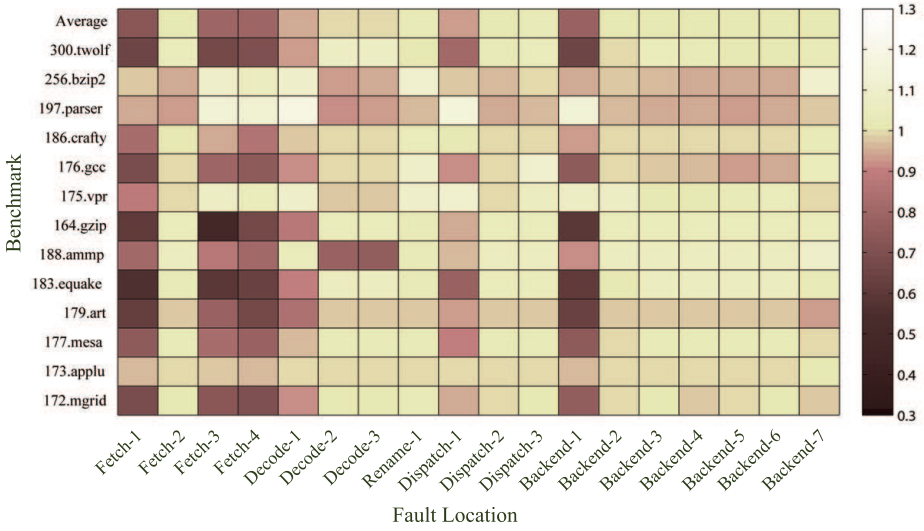


Figure 5.15: Variations in the speed-up of the animator core for different hard-fault locations across SPEC-CPU-2K benchmarks. To only highlight the impact of hard-fault locations, in each row, results are normalized to the average speed-up that can be achieved by the NM coupled cores for that particular benchmark.

advantage/disadvantage that comes from the inherent benchmark suitability for core coupling. As can be seen, hard-faults in some locations are more harmful than others. These locations consist of the PC, integer ALU, and instruction fetch queue. Another interesting observation is that, for a benchmark like 197.parser, reaction to defects can significantly differ from other benchmarks. We conclude two main points from this plot. First, on average, there are only a few fault locations that can drastically impact the NM speed-up gain. Second, for a given fault location, different benchmarks show various degrees of susceptibility; thus, heterogeneity across the benchmarks running on a CMP system helps NM to achieve a higher speed-up by having a more suitable workload assigned to the coupled cores.

Summary of Benefits and Overheads: Figure 5.16 demonstrates the amount of speed-up that can be achieved by the NM coupled cores for CMP systems with different numbers

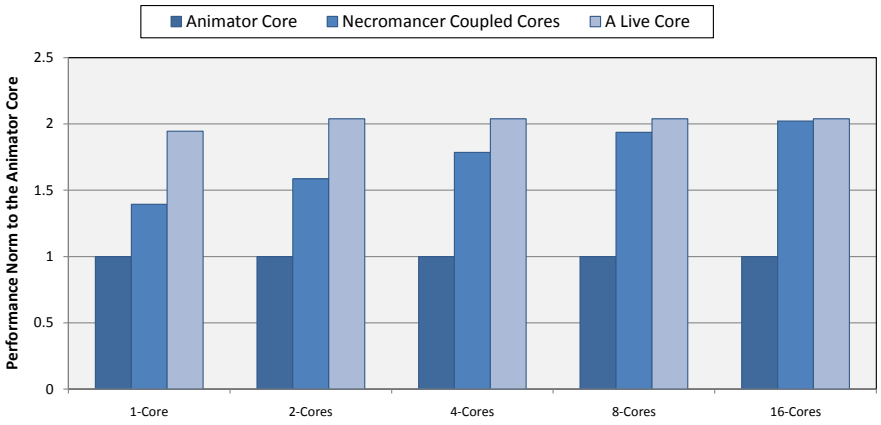


Figure 5.16: Performance of the baseline animator core, NM coupled cores, and a live core normalized to the average performance of a baseline animator core. Due to the higher heterogeneity across the benchmarks for a CMP system with more cores, NM can achieve a higher overall speed-up.

of cores. As can be seen, NM achieves a higher overall speed-up as the number of cores increases. For a 16-core system, on average, the coupled cores can achieve the performance of a live core, essentially providing the appearance of a fully-functional 6-issue baseline core with a 2-issue animator core. This is because NM achieves different speed-ups based on the defect type, location, and the workload running on the system. Here, we assume full utilization, which means there is always one job per core. Hence, for larger CMPs, with more heterogeneity across the benchmarks running on the system, there is more opportunity for NM to exploit. The speed-up evaluation was done by conducting a Monte Carlo simulation with 1000 iterations. In each iteration, we select one benchmark for each core, while allowing replication in the selected benchmarks.

Figure 5.17 shows the breakdown of area and power overheads for our scheme. Here, we assume a single core system has 2MB L2 while assuming 1MB shared L2 per core for CMP systems. As can be seen, the area overhead gradually shrinks as the number of cores grows since the cost of the animator core is amortized among more cores. Nevertheless,

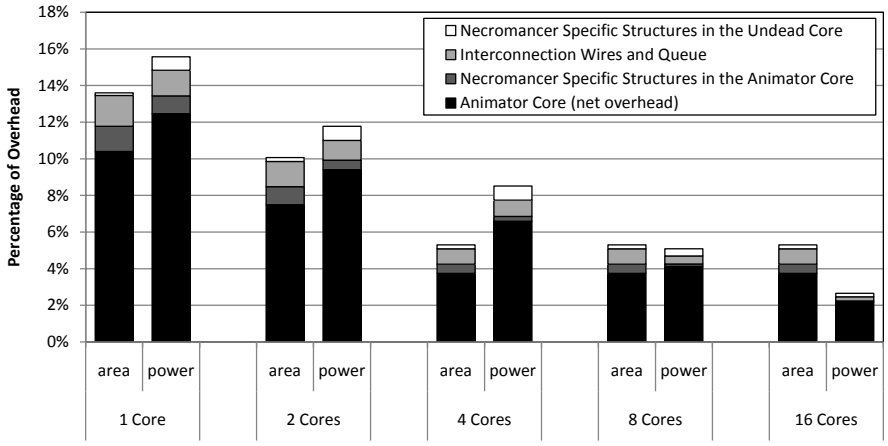


Figure 5.17: Break-down of NM area and power overheads for CMP systems with different numbers of cores. As can be seen, the overheads that are imposed by the the baseline animator core is typically the major component, which gets amortized as the number of cores grows.

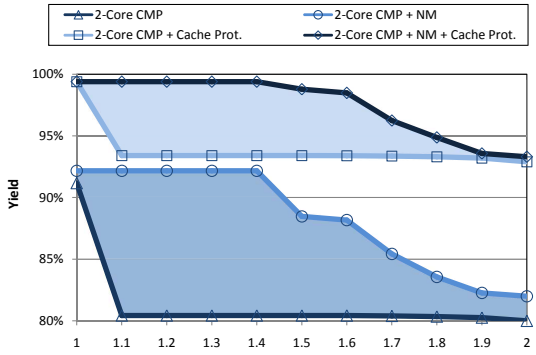
since we simply replicate the 4-core building block to construct CMPs with more than 4 cores, the area overhead remains the same. In terms of power overhead, two points should be noted. First, based on our target defect rate, for CMPs with more than 4 cores, other animator cores remain disabled and do not contribute to the power consumption. Next, as the speed-up results show, for CMPs with less than 8 cores, the undead core remains ahead of the animator core and it needs to stall when the queue gets full. During stall times, the undead core does not consume dynamic power which is accounted for in the net overhead of the animator core – Figure 5.17.

5.6 Throughput Enhancement

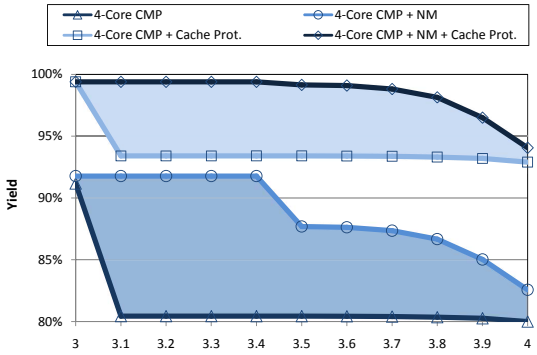
Given a population of manufactured chips, the main objective of NM is to improve the average system throughput of the population. For this purpose, we model 1000 manufactured chips with randomly distributed defects based on our target defect rate. If the anima-

tor core, communication queue, or any of the NM specific modules like the hint gathering unit are faulty, we simply disable the animator core. Figure 5.18 depicts the throughput enhancement results (shaded regions) based on *throughput binning* for 2, 4, and 8-core CMP systems. Note that NM significantly enhances the overall system throughput for a population of manufactured chips. In each sub-plot, we consider two baselines (a CMP system without any cache protection and a CMP system with proper protection for on-chip caches). The horizontal axes show the system throughput, normalized to the throughput of a single baseline core. For a CMP system with N baseline cores, we illustrate the throughput binning results for throughput values between $N-1$ and N. Since we assume, on average, one defect per each 5 chips, yield is always above 80%. However, there is a small chance that multiple defects hit the same chip which precludes a yield of 100% at a throughput of $N-1$, even when protecting the on-chip caches. As can be seen, cache protection is a necessity and fortunately, can be provided easily (e.g., row/column redundancies). As discussed earlier, based on the expected defect rate for current and near future CMOS technologies, on average one defect per five manufactured 100mm2 dies should be expected. In the case of a defect in one of the original cores, we apply our scheme. On the other hand, if any of the animator cores, communication queues, or NM specific modules like the hint gathering unit are faulty, we simply disable the animator core and the rest of the system can continue their normal operation. Although extremely rare, there are cases that both the animator and an original core can have defects. All these possible scenarios were considered in our evaluation.

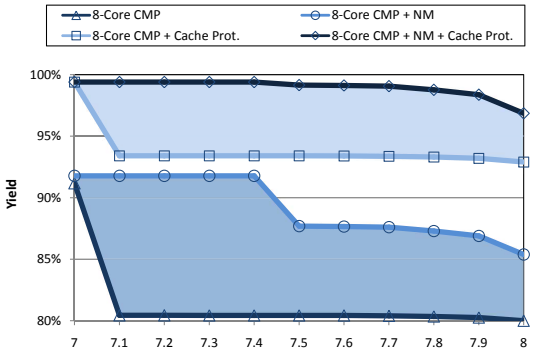
Finally, as discussed earlier, based on the expected defect rate for current and near future CMOS technologies, on average one defect per five manufactured $100mm^2$ dies should be



(a) Achievable yield for a 2-core CMP, given an expected level of system throughput.



(b) Achievable yield for a 4-core CMP, given an expected level of system throughput.



(c) Achievable yield for an 8-core CMP, given an expected level of system throughput.

Figure 5.18: Throughput enhancement for a population of manufactured chips with different number of cores. Here, we consider two baselines, CMP system without and with proper protection for on-chip caches, and show the yield improvement for these two cases (shaded regions) when applying NM. Each line presents the achievable yields for different expected throughput values.

expected. In the case of a defect in one of the original cores, we apply our scheme. On the other hand, if any of the animator cores, communication queues, or NM specific modules like the hint gathering unit are faulty, we simply disable the animator core and the rest of the system can continue their normal operation.

5.7 Summary

Since manufacturing defects directly impact yield in nanoscale CMOS technologies, to maintain an acceptable level of manufacturing yield, these defects need to be addressed properly. Non-cache parts of a core are less structured and homogeneous; thus, tolerating defects in the general core area has remained a challenging problem. In this work, we presented Necromancer, an architectural scheme to enhance the system throughput by exploiting dead cores. Although a dead core cannot be trusted to perform program execution, for most defect incidences, its execution flow – when starting from a valid architectural state – coarsely matches the intact program behavior for a long time period. Hence, Necromancer does not rely on correct program execution on a dead core; instead, it only expects this undead core to generate effective execution hints to accelerate the animator core. In order to increase Necromancer efficacy, we use microarchitectural techniques to provide intrinsically robust hints, effective hint disabling, and dynamic inter-core state resynchronization. For a 4-core CMP system, on average, our approach enables the coupled core to achieve 87.6% of the performance of a live core. This defect tolerance and throughput enhancement comes at modest area and power overheads of 5.3% and 8.5%, respectively. We believe NM is a valuable and low-cost solution for tolerating manufacturing defects and improving the throughput of the current and near future mainstream CMP systems.

CHAPTER VI

Conclusions

The rapid growth of the silicon process over the last decade has substantially improved semiconductor integration levels. However, as device density grows, each transistor gets smaller and more fragile leading to an overall higher susceptibility of chips to hard-faults. Manufacturing defects, process variation, and wearout induced failures are the main sources of hard-faults in deep submicron technology nodes. These hard-faults result in permanent silicon defects and impact manufacturing yield, performance, and lifetime throughput of semiconductor devices. In addition, power consumption and heat dissipation have become key challenges in the design of microprocessors. Growing power consumption affects device lifetime, the cost of thermal packaging, cooling, electricity, and data center air conditioning. Dynamic voltage scaling is commonly used to reduce the power consumption. However, the supply voltage of a microprocessor cannot be reduced below a certain threshold without addressing SRAM failures. Therefore, to allow a robust operation in the presence of faults, these reliability concerns need to be addressed in the current and future CMOS designs.

Compared to simpler semiconductor devices, protecting high-performance micropro-

cessors against hard-faults is a challenging and relatively new problem. These microprocessors contain hundreds of millions of transistors and failure of any transistor can impact the correct operation. Considering a multicore system, as the number of transistors per core increases, it becomes more likely to have a faulty transistor in a given core. In such a scenario, simple reliability solutions like disabling the faulty core is apparently not cost effective. Furthermore, the design complexity of these microprocessors increases everyday. Complexity in the connectivity between stages along with super-pipelining prevent designers from employing reliability techniques which break cores into pipeline stages and allow stage borrowing between cores. Moreover, the operational clock frequency of these microprocessors is relatively high. Solutions based on fine-grained spares are not practical in this domain, due to the tight delay budget and the inherent complexity in the connectivity. In addition, these processors conventionally operate at the highest possible frequency at a given supply voltage. Therefore, simple reliability techniques that suggest using high voltage and frequency guard-bands or slowing down the processor, eliminate most of the achievable performance of these microprocessors.

To tackle hard-faults in modern high-performance microprocessors, we proposed a comprehensive, low-cost solution for protecting the entire core area including on-chip caches and also the non-cache parts of the core. First, we presented a flexible cache architecture, ZerehCache, to protect regular SRAM structures against high degree of process variation, wearout induced failures, and manufacturing defects. ZerehCache virtually reorganizes the cache data array using a permutation network to provide higher degrees of freedom for spare allocation. In order to study the impact of fault patterns on the redundancy requirements in a cache, we proposed a methodology to model the collision patterns

in caches as a graph problem. Given this model, a graph coloring scheme is employed to minimize the amount of additional redundancy required for protecting the cache. Next, to efficiently tolerate the large number of SRAM failures that arise, in the large multi-banked caches, when operating in the near-threshold region, a highly reconfigurable cache design, Archipelago, was presented. Since low-power operation is optional, instead of relying on redundancy, Archipelago resizes the cache to provide spare elements. Furthermore, to maximize the effective cache capacity in low-power mode, a near optimal minimum clique covering configuration algorithm was introduced.

Finally, to protect the general core area against hard-faults, a robust and heterogeneous core coupling execution scheme, Necromancer, was presented. Although a faulty core cannot be trusted to correctly execute programs, we observed that for most defects, when starting from a valid architectural state, execution traces on a defective core actually coarsely resemble those of fault-free executions. In light of this insight, Necromancer exploits a functionally dead core to improve system throughput by supplying hints regarding high-level program behavior. We partition the cores in a conventional CMP system into multiple groups in which each group shares a lightweight core that can be substantially accelerated using these execution hints from a potentially dead core. However, due to the presence of defects, a perfect data or instruction stream cannot be provided by the dead core. This necessitates employing generic hints that are more resilient to local abnormalities. Given the variation in the usefulness of the execution information, in order to enhance the efficiency of the lightweight core, we introduced several fine-grained hint disabling mechanisms. Besides, when the faulty core gets completely off the correct execution path, hints become useless, and it needs to be brought back to a valid execution point. For this purpose, we

leverage coarse-grained online monitoring of the effectiveness of the hints over a large time period to decide whether the faulty core should be resynchronized with the lightweight core.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. Gonzalez. Low vccmin fault-tolerant cache with highly predictable performance. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.

[2] D. Achlioptas and C. Moore. The chromatic number of random regular graphs. In *8th International Workshop on Randomization and Computation*, pages 219–228, 2004.

[3] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. In *Proc. of the 36th ACM Symposium on Theory of Computing*, pages 587–593, New York, NY, USA, 2004. ACM.

[4] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy. Process variation in embedded memories: failure analysis and variation aware architecture. *Journal of Solid State Circuits*, 49(9):1804–1814, 2005.

[5] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(1):27–38, Jan. 2005.

[6] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *Proc. of the 34th Annual International Symposium on Computer Architecture*, pages 470–481, 2007.

[7] A. Ansari, S. Feng, S. Gupta, and S. Mahlke. Putting faulty cores to work. *IEEE Micro*, 31(2), 2011.

[8] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *Proc. of the 2009 International Symposium on Low Power Electronics and Design*, pages 307–310, 2009.

[9] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Necromancer: enhancing system throughput by animating dead cores. In *Proc. of the 37th Annual International Symposium on Computer Architecture*, pages 473–484, 2010.

[10] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Archipelago: A polymorphic cache design for enabling robust near-threshold operation. In *Proc. of the 17th International Symposium on High-Performance Computer Architecture*, 2011.

[11] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Zerehcache: Armoring cache architectures in high defect density technologies. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, pages 100–110, 2009.

[12] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Maximizing spare utilization by virtually reorganizing faulty cache lines. *IEEE Transactions on Computers*, 60(1), 2011.

[13] T. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *Proc. of the 32nd Annual International Symposium on Microarchitecture*, pages 196–207, 1999.

[14] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Transactions on Computers*, 35(2):59–67, Feb. 2002.

[15] R. D. Barnes, E. N. Nystrom, J. W. Sias, S. J. Patel, N. Navarro, and W. W. Hwu. Beating in-order stalls with ”flea-flicker” two-pass pipelining. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, page 387, 2003.

[16] W. Bartlett and L. Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):87–96, 2004.

[17] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3):459–466, 1990.

[18] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen. Nonstop advanced architecture. In *International Conference on Dependable Systems and Networks*, pages 12–21, June 2005.

[19] K. Bernstein. Nano-meter scale cmos devices (tutorial presentation), 2004.

[20] S. Bhunia, S. Mukhopadhyay, and K. Roy. Process variations and process-tolerant design. In *Proc. of the 2007 International Conference on VLSI Design*, pages 699–704, Washington, DC, USA, 2007. IEEE Computer Society.

[21] D. Bol, R. Ambroise, D. Flandre, and J. D. Legat. Analysis and minimization of

practical energy in 45nm subthreshold logic circuits. In *Proc. of the 2008 International Conference on Computer Design*, pages 294–300, Oct. 2008.

[22] B. Bollobas. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[23] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[24] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin. Tolerating hard faults in microprocessor array structures. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, page 51, 2004.

[25] F. A. Bower, D. J. Sorin, and S. Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In *Proc. of the 38th Annual International Symposium on Microarchitecture*, pages 197–208, 2005.

[26] D. Brooks, V. Tiwari, and M. Martonosi. A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[27] B. Calhoun and A. Chandrakasan. A 256kb sub-threshold sram in 65nm cmos. *2008 IEEE International Solid-State Circuits Conference*, pages 2592–2601, Feb. 2006.

[28] B. H. Calhoun and A. P. Chandrakasan. A 256-kb 65-nm sub-threshold sram design for ultra-low-voltage operation. *Journal of Solid State Circuits*, 42(3):680–688, Mar. 2007.

[29] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, and W. Haensch. Stable sram cell design for the 32 nm node and beyond. *Symposium on VLSI Technology*, pages 128–129, June 2005.

[30] G. Chen, D. Blaauw, T. Mudge, D. Sylvester, and N. Kim. Yield-driven near-threshold sram design. In *Proc. of the 2007 International Conference on Computer Aided Design*, pages 660–666, Nov. 2007.

[31] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 0, 2009.

[32] A. Christou. *Electromigration and Electronic Device Degradation*. John Wiley and Sons, Inc., 1994.

[33] K. Constantinides, S. Plaza, J. A. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. Bulletproof: A defect-tolerant CMP switch architecture. In *Proc. of the 12th International Symposium on High-Performance Computer Architecture*, pages 3–14, Feb. 2006.

[34] W. Culbertson, R. Amerson, R. Carter, P. Kuekes, and G. Snider. Defect tolerance on the teramac custom computer. In *Proc. of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, pages 116–123, 1997.

[35] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing

speculation. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 7–18, 2003.

[36] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. *Proc. of the 29th Annual International Symposium on Computer Architecture*, pages 148–157, 2002.

[37] H. Fujiwara, S. Okumura, Y. Iguchi, H. Noguchi, H. Kawaguchi, and M. Yoshimoto. A 7t/14t dependable sram and its array structure to avoid half selection. In *Proc. of the 2009 International Conference on VLSI Design*, pages 295–300, Jan. 2009.

[38] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[39] A. H. Gebremedhin and F. M. I. Parallel graph coloring algorithms using openmp. In *First European Workshop on OpenMP*, pages 10–18, 1999.

[40] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale cmps through core overclocking. In *Proc. of the 16th International Conference on Parallel Architectures and Compilation Techniques*, pages 213–224, 2007.

[41] S. Gupta, S. Feng, A. Ansari, J. A. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 141–151, 2008.

[42] S. Gupta, S. Feng, A. Ansari, J. A. Blome, and S. Mahlke. Stagenetslice: A reconfigurable microarchitecture building block for resilient cmp systems. In *Proc.*

of the 2008 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pages 1–10, 2008.

[43] S. Gupta, S. Feng, A. Ansari, and S. Mahlke. Stagenet: A reconfigurable fabric for constructing dependable cmps. *IEEE Transactions on Computers*, 60(1), 2011.

[44] M. Hempstead, G. Y. Wei, and D. Brooks. Architecture and circuit techniques for low-throughput, energy-constrained systems across technology generations. In *Proc. of the 2006 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 368–378, 2006.

[45] T. Higashiki. Status and future lithography for sub hp32nm device. In *2009 Lithography Workshop*, 2009.

[46] M. Horiguchi. Redundancy techniques for high-density drams. In *2nd Annual IEEE International Conference on Innovative Systems Silicon*, pages 22–29, 1997.

[47] L. D. Hung, M. Goshima, and S. Sakai. Seva: A soft-error- and variation-aware cache architecture. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, pages 47–54, Washington, DC, USA, 2006. IEEE Computer Society.

[48] ITRS. International technology roadmap for semiconductors 2008, 2008. http://www.itrs.net/.

[49] D. Kannan, A. Shrivastava, V. Mohan, S. Bhardwaj, and S. Vrudhula. Temperature and process variations aware power gating of functional units. In *Proc. of the 2008 International Conference on VLSI Design*, pages 515–520, 2008.

[50] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[51] S. Kaxiras, H. Zhigang, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. *Proc. of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.

[52] R. E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[53] C. Kim, S. Sethumadhavan, M. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler. Composable lightweight processors. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 381–393, Dec. 2007.

[54] 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 Annual International Symposium on Microarchitecture*, 2007.

[55] W. Klotz. Graph coloring algorithms, 2002. Mathematik-Bericht 5, Clausthal University of Technology, Clausthal, Germany.

[56] I. Koren and Z. Koren. Incorporating yield enhancement into the floorplanning process. *IEEE Transactions on Computers*, 49:532–541, 2000.

[57] J. Kowaleski, T. Truex, D. Dever, D. Ament, W. Anderson, L. Bair, , et al. Implementation of an alpha microprocessor in soi. *2003 IEEE International Solid-State Circuits Conference*, 1:248–491, 2003.

[58] J. P. Kulkarni, K. Kim, and K. Roy. A 160 mv, fully differential, robust schmitt

trigger based sub-threshold sram. In *Proc. of the 2007 International Symposium on Low Power Electronics and Design*, pages 171–176, New York, NY, USA, 2007. ACM.

[59] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proc. of the 36th Annual International Symposium on Microarchitecture*, pages 81–92, Dec. 2003.

[60] R. Kumar, N. Jouppi, and D. Tullsen. Conjoined-core chip multiprocessing. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 195–206, 2004.

[61] S. Kundu, T. M. Mak, and R. Galivanche. Trends in manufacturing test methods and their implications. In *Proc. of the 2004 International Test Conference*, pages 679–687, Washington, DC, USA, 2004. IEEE Computer Society.

[62] J. H. Lee, Y. J. Lee, and Y. B. Kim. SRAM Word-oriented Redundancy Methodology using Built In Self-Repair. In *IEEE International ASIC Conference '04*, pages 219–222, 2004.

[63] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. S. Hari, and S. V. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults. In *Proc. of the 15th International Symposium on High-Performance Computer Architecture*, pages 105–116, 2009.

[64] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks. Replacing 6t srams with 3t1d drams in the l1 data cache to combat process variability. *IEEE Micro*, 28(1):60–68, 2008.

[65] T. Luczak. Chromatic number of random graphs. *Combinatorica*, 11(1):45–54, 1991.

[66] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. of the 2002 International Conference on Computer Aided Design*, pages 721–725, 2002.

[67] A. Meixner, M. Bauer, and D. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. *IEEE Micro*, 28(1):52–59, 2008.

[68] K. Meng and R. Joseph. Process variation aware cache leakage management. *Proc. of the 2006 International Symposium on Low Power Electronics and Design*, pages 262–267, Oct. 2006.

[69] F. Moradi, D. Wisland, S. Aunet, H. Mahmoodi, and T. Cao. 65nm sub-threshold 11t-sram for ultra low voltage applications. *Intl. Symposium on System-on-a-Chip*, pages 113–118, Sept. 2008.

[70] Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto. An area-conscious low-voltage-oriented 8t-sram design under dvs environment. *IEEE Symposium on VLSI Circuits*, pages 256–257, June 2007.

[71] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscale cmos. *IEEE*

Transactions on Computer-Aided Design of Integrated Circuits and Systems, pages 1859–1880, 2005.

[72] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *IEEE Micro*, pages 3–14, 2007.

[73] D. Nassimi and S. Sahni. A self routing benes network. In *Proc. of the 7th Annual International Symposium on Computer Architecture*, pages 190–195, New York, NY, USA, 1980. ACM.

[74] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. *Proc. of the 39th Annual International Symposium on Microarchitecture*, 0:15–25, 2006.

[75] M. D. Powell, A. Biswas, S. Gupta, and S. S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *Proc. of the 36th Annual International Symposium on Computer Architecture*, June 2009.

[76] Z. Purser, K. Sundaramoorthy, and E. Rotenberg. A study of slipstream processors. In *Proc. of the 33rd Annual International Symposium on Microarchitecture*, pages 269–280, 2000.

[77] A. Raychowdhury, S. Mukhopadhyay, and K. Roy. A feasibility study of subthreshold sram across technology generations. In *Proc. of the 2005 International Conference on VLSI Design*, pages 417–422, Oct. 2005.

[78] D. Roberts, N. S. Kim, and T. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 570–578, Aug. 2007.

[79] B. F. Romanescu and D. J. Sorin. Core cannibalization architecture: Improving lifetime chip performance for multicore processor in the presence of hard faults. In *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[80] N. Sadler and D. Sorin. Choosing an error protection scheme for a microprocessor's l1 data cache. In *Proc. of the 2006 International Conference on Computer Design*. IEEE, 2006.

[81] K. Sankaranarayanana, S. Velusamy, M. Stan, and K. Skadron. A case for thermal-aware floorplanning at the microarchitectural level. *The Journal of Instruction-Level Parallelism*, 2005.

[82] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, pages 3–13, Feb. 2008.

[83] K. Sasaki. A 9-ns 1-mbit cmos ram. *Journal of Solid State Circuits*, 24:1219–1225, 1989.

[84] E. Schuchman and T. N. Vijaykumar. Rescue: A microarchitecture for testabil-

ity and defect tolerance. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 160–171, 2005.

[85] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, New York, NY, USA, 2002. ACM.

[86] Z. Shi and R. Lee. Implementation complexity of bit permutation instructions. In *Signals, Systems and Computers*, pages 879–886, Nov. 2003.

[87] P. Shivakumar, S. Keckler, C. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proc. of the 2003 International Conference on Computer Design*, page 481, Oct. 2003.

[88] D. Siewiorek and R. Swarz. *Reliable Computer Systems: Design and Evaluation, 3rd Edition*. AK Peters, Ltd., 1998.

[89] L. Spainhower and T. Gregg. IBM S/390 Parallel Enterprise Server G5 Fault Tolerance: A Historical Perspective. *IBM Journal of Research and Development*, 43(6):863–873, 1999.

[90] E. Sperling. Turn down the heat...please, 2006. http://www.edn.com/article/CA6350202.html.

[91] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 177–186, June 2004.

[92] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 520–531, June 2005.

[93] K. Takahashi, H. Doi, N. Tamura, K. Mimuro, T. Hashizume, Y. Moriyama, and Y. Okuda. A 0.9 v operation 2-transistor flash memory for embedded logic lsis. *Symposium on VLSI Technology*, pages 21–22, 1999.

[94] K. Takeda, Y. Hagihara, Y. Aimoto, M. Nomura, Y. Nakazawa, T. Ishii, and H. Kobatake. A read-static-noise-margin-free sram cell for low-vdd and high-speed applications. *2006 IEEE International Solid-State Circuits Conference*, 41(1):113–121, Jan. 2006.

[95] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proc. of the 35th Annual International Symposium on Computer Architecture*, pages 363–374, June 2008.

[96] K. M. Thompson. Intel and the myths of test. *IEEE Journal of Design & Test of Computers*, 13(1):79–81, 1996.

[97] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 129–140, Dec. 2008.

[98] N. Verma and A. Chandrakasan. A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy. *IEEE Journal of Solid-State Circuits*, 43(1):141–149, Jan. 2008.

[99] N. J. Wang, M. Fertig, and S. J. Patel. Y-branches: When you come to a fork in the road, take it. In *Proc. of the 12th International Conference on Parallel Architectures and Compilation Techniques*, pages 56–65, 2003.

[100] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *International Conference on Dependable Systems and Networks*, page 61, June 2004.

[101] C. Weaver and T. M. Austin. A fault tolerant approach to microprocessor design. In *Proc. of the 2001 International Conference on Dependable Systems and Networks*, pages 411–420, Washington, DC, USA, 2001. IEEE Computer Society.

[102] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983.

[103] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. *Proc. of the 35th Annual International Symposium on Computer Architecture*, 0:203–214, 2008.

[104] E. Wu, J. M. McKenna, W. Lai, E. Nowak, and A. Vayshenker. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides. *Solid-State Electronics*, 46:1787–1798, 2002.

[105] X. Yang, M. Vachharajani, and R. B. Lee. Fast subword permutation instructions based on butterfly networks. In *SPIE, Media Processor 2000*, pages 80–86, 2000.

[106] L. Youngs and S. Paramanandam. Mapping and repairing embedded-memory defects. *IEEE Journal of Design and Test*, 14(1):18–24, 1997.

[107] S. Zafar et al. A model for negative bias temperature instability (nbti) in oxide and high k pfets. In *Symposium on VLSI Technology*, pages 45–50, 2004.

[108] J. Zeigler. Terrestrial cosmic ray intensities. *IBM Journal of Research and Development*, 42(1):117–139, 1998.

[109] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, Univ. of Virginia Dept. of Computer Science, Jan. 2003.

[110] H. Zhou. Dual-Core Execution: Building a Highly Scalable Single-Thread Instruction Window. In *Proc. of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 231–242, Sept. 2005.

[111] C. Zilles and G. Sohi. Master/slave speculative parallelization. In *Proc. of the 35th Annual International Symposium on Microarchitecture*, pages 85–96, Nov. 2002.