

UNIVERSITY OF CALIFORNIA,  
IRVINE

Resilient On-Chip Memory Design in the Nano Era

DISSERTATION

submitted in partial satisfaction of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Abbas BanaiyanMofrad

Dissertation Committee:  
Professor Nikil Dutt, Chair  
Professor Alex Nicolau  
Professor Alex Veidenbaum

2015



# DEDICATION

*To my wife Marjan*  
whom provided me the necessary strength to pursue my dreams.

# TABLE OF CONTENTS

|  | Page       |
|--|------------|
| <b>LIST OF FIGURES</b>                                     | <b>vi</b>  |
| <b>LIST OF TABLES</b>                                      | <b>ix</b>  |
| <b>ACKNOWLEDGMENTS</b>                                     | <b>x</b>   |
| <b>CURRICULUM VITAE</b>                                    | <b>xii</b> |
| <b>ABSTRACT OF THE DISSERTATION</b>                        | <b>xv</b>  |
| <b>1 Introduction</b>                                      | <b>1</b>   |
| 1.1 Nano Era Design Trends and Challenges . . . . .        | 1          |
| 1.1.1 Technology Trend . . . . .                           | 1          |
| 1.1.2 Chip Design Trend . . . . .                          | 2          |
| 1.1.3 Application Trend . . . . .                          | 2          |
| 1.2 Memories and Errors . . . . .                          | 3          |
| 1.3 State-of-the-art Research Efforts . . . . .            | 5          |
| 1.4 Motivation . . . . .                                   | 7          |
| 1.5 Thesis Contributions . . . . .                         | 10         |
| 1.6 Thesis Organization . . . . .                          | 14         |
| <b>2 Flexible and Low-Cost Fault-tolerant Cache Design</b> | <b>15</b>  |
| 2.1 Introduction . . . . .                                 | 15         |
| 2.2 Related Work . . . . .                                 | 19         |
| 2.2.1 Circuit-level Techniques . . . . .                   | 19         |
| 2.2.2 Error Coding Techniques . . . . .                    | 20         |
| 2.2.3 Architecture-Level Techniques . . . . .              | 21         |
| 2.3 FFT-Cache . . . . .                                    | 23         |
| 2.3.1 Proposed Architecture . . . . .                      | 23         |
| 2.3.2 Evaluation . . . . .                                 | 32         |
| 2.4 EEC: Embedded Erasure Coding . . . . .                 | 41         |
| 2.4.1 Preliminaries . . . . .                              | 41         |
| 2.4.2 Proposed EEC . . . . .                               | 43         |
| 2.4.3 Evaluation . . . . .                                 | 48         |
| 2.5 Summary . . . . .                                      | 55         |

|          |   |            |
|----------|---|------------|
| <b>3</b> | <b>Scalable Fault-tolerant Cache Design in CMPs</b>         | <b>56</b>  |
| 3.1      | Introduction . . . . .                                      | 56         |
| 3.2      | Related Work and Motivation . . . . .                       | 59         |
| 3.3      | RESCUE . . . . .  | 61         |
| 3.3.1    | Baseline Architecture . . . . .                             | 61         |
| 3.3.2    | RESCUE Overview . . . . .                                   | 64         |
| 3.3.3    | RESCUE Remapping Policies . . . . .                         | 73         |
| 3.3.4    | Evaluation . . . . .  | 78         |
| 3.3.5    | Quantitative Comparison to Alternative Techniques . . . . . | 82         |
| 3.3.6    | Experimental Results . . . . .                              | 83         |
| 3.4      | Summary . . . . .   | 93         |
| <b>4</b> | <b>Interconnect-aware Resilient Cache Design</b>            | <b>94</b>  |
| 4.1      | Introduction . . . . .                                      | 94         |
| 4.2      | Related Work . . . . .                                      | 98         |
| 4.2.1    | Fault-tolerant Cache Design . . . . .                       | 99         |
| 4.2.2    | Fault-tolerant NoC design . . . . .                         | 99         |
| 4.2.3    | Memory Reliability in Multi/Many-cores . . . . .            | 100        |
| 4.3      | NoC-based Fault-tolerance of LLC . . . . .                  | 101        |
| 4.3.1    | Basic Idea and Baseline Architecture . . . . .              | 101        |
| 4.3.2    | Proposed Architecture . . . . .                             | 106        |
| 4.3.3    | Evaluation . . . . .  | 113        |
| 4.3.4    | Results . . . . .   | 116        |
| 4.4      | CoDEC . . . . .   | 123        |
| 4.4.1    | Background and Motivation . . . . .                         | 123        |
| 4.4.2    | CoDEC Overview . . . . .                                    | 127        |
| 4.4.3    | Evaluation . . . . .  | 131        |
| 4.4.4    | Experimental Results . . . . .                              | 136        |
| 4.5      | Summary . . . . .   | 141        |
| <b>5</b> | <b>Relaxing Error Resiliency in On-chip Memories</b>        | <b>143</b> |
| 5.1      | Introduction . . . . .                                      | 143        |
| 5.1.1    | Cost of Guard-banding . . . . .                             | 144        |
| 5.1.2    | Approximate Computing . . . . .                             | 144        |
| 5.1.3    | Challenges for Effective Use of PFMs: . . . . .             | 145        |
| 5.1.4    | Contributions . . . . .                                     | 146        |
| 5.2      | Related Work . . . . .                                      | 147        |
| 5.2.1    | Exploiting Memory Variability . . . . .                     | 147        |
| 5.2.2    | Approximate Computing . . . . .                             | 148        |
| 5.3      | Relaxed Cache . . . . .                                     | 150        |
| 5.3.1    | Hardware Support . . . . .                                  | 150        |
| 5.3.2    | Software Support . . . . .                                  | 155        |
| 5.4      | Experimental Evaluation . . . . .                           | 158        |
| 5.4.1    | Experimental Setup . . . . .                                | 158        |
| 5.4.2    | Benchmarks . . . . .  | 159        |

|          |   |            |
|----------|---|------------|
| 5.4.3    | Results . . . . .   | 160        |
| 5.5      | Summary . . . . .   | 163        |
| <b>6</b> | <b>System-wide Memory Resiliency Design Space Exploration</b> | <b>164</b> |
| 6.1      | Introduction . . . . .  | 164        |
| 6.2      | Background . . . . .  | 167        |
| 6.2.1    | Reliable Memory Design . . . . .                              | 167        |
| 6.2.2    | Design Space Exploration . . . . .                            | 168        |
| 6.2.3    | Exemplar Many-core Architecture . . . . .                     | 169        |
| 6.2.4    | Motivation . . . . .  | 169        |
| 6.3      | System-Level Fault-tolerance Modeling . . . . .               | 172        |
| 6.3.1    | Reliability Clustering . . . . .                              | 172        |
| 6.4      | Evaluation . . . . .  | 173        |
| 6.4.1    | Experimental Setup . . . . .                                  | 174        |
| 6.4.2    | Design Space Exploration Studies . . . . .                    | 175        |
| 6.4.3    | Sample Exploration Results . . . . .                          | 176        |
| 6.5      | Summary . . . . .   | 183        |
| <b>7</b> | <b>Conclusion and Future Work</b>                             | <b>184</b> |
| 7.1      | Conclusion . . . . .  | 184        |
| 7.2      | Future Work . . . . .   | 185        |
|          | <b>Bibliography</b>   | <b>189</b> |

# LIST OF FIGURES

|  | Page |
|--|------|
| 1.1 Memory Abstractions, Errors, and Opportunities. . . . .  | 4    |
| 1.2 Summary of the thesis main contributions. . . . .  | 10   |
| 2.1 FFT-Cache Access Flowchart. . . . .  | 25   |
| 2.2 Details of a row of FDM. . . . .   | 26   |
| 2.3 An example of FDM configuration and remapping for a given distribution of faults in a 4-way set associative cache. . . . .                           | 30   |
| 2.4 Architecture details of (a) a conventional 4-way set associative cache, (b) the proposed FFT-Cache with FDM and 2 sub-blocks per block. . . . .      | 31   |
| 2.5 The effect of changing one design parameter of L1 FFT-Cache while fixing other parameters for different Vdd values. . . . .                          | 35   |
| 2.6 The effect of changing one design parameter of L2 FFT-Cache while fixing other parameters for different Vdd values. . . . .                          | 36   |
| 2.7 Processor performance degradation when applying FFT-Cache for (a) L1 (b) L2 and (c) L1 and L2 at the same time. . . . .                              | 37   |
| 2.8 Power and Area overheads of FFT-Cache. . . . .   | 38   |
| 2.9 Comparison of effective cache size for different fault-tolerant techniques. . . . .  | 39   |
| 2.10 MBU size distribution and particular patterns with high occurrence probabilities in a 45 nm SRAM Array . . . . .                                    | 42   |
| 2.11 Basic erasure coding technique. . . . .   | 43   |
| 2.12 A sample parity-based erasure code with one redundant block. . . . .  | 45   |
| 2.13 A 4-way L1 cache architecture protected with EEC that shows the updated write operation in three steps. . . . .                                     | 46   |
| 2.14 Performance overhead of EEC for L1 and L2 caches on Alpha processor . . . . .   | 51   |
| 3.1 Empirical data for the cache size per core across various CMP architecture. . . . .  | 62   |
| 3.2 Baseline 8 processor CMP structure. . . . .  | 63   |
| 3.3 Architecture details of (a) a conventional 4-banked 2-way set associative cache, (b) the proposed RESCUE with DFM and 2 sub-blocks per line. . . . . | 66   |
| 3.4 RESCUE cache access flowchart. . . . .   | 67   |
| 3.5 DFM initialization algorithm. . . . .  | 69   |
| 3.6 DFM configuration algorithm. . . . .   | 70   |
| 3.7 Effective cache size for different techniques with the (a) two banks and (b) four banks LLC. . . . .   | 74   |

|      |  |     |
|------|--|-----|
| 3.8  | Example showing the (a)DNUCA optimized scheme with remapping limited to one bank with 1 hop (b)SNUCA optimized scheme with remapping limited to one bank with 1 hop (c)Adjacent mapping scheme (d)Global mapping scheme. | 76  |
| 3.9  | Area, leakage, and dynamic power overheads of RESCUE for an 8-core CMP with 16 LLC banks in 45nm. . . . .  | 82  |
| 3.10 | Effective cache size after applying RESCUE policies. . . . .   | 85  |
| 3.11 | Sensitivity studies of RESCUE policies across various workloads. . . . .   | 86  |
| 3.12 | Sensitivity studies of network configuration with MEM-WL; Average Packet Latency (a), Normalized IPC (b), Total Power (c), and EDP (d). . . . .  | 88  |
| 3.13 | Sensitivity studies of NUCA policies with MEM-WL; Average packet latency (1st row), normalized IPC (2nd row), total power consumption(3rd row), EDP(4th row). . . . .  | 89  |
| 3.14 | Sensitivity studies of Cache size over-provisioning with MIX-WL; Average packet latency (a), normalized IPC (b), total power consumption (c), and EDP (d). . . . .   | 90  |
| 3.15 | Power reduction across process technology nodes over all organizations and policies with MEM-WL, MIX-WL, and CPU-WL when applying RESCUE (left) and RESCUE+COP (right). . . . .  | 91  |
| 4.1  | Baseline NoC architecture. . . . .   | 102 |
| 4.2  | SoCIN component details. . . . .   | 105 |
| 4.3  | Three phases of a fault-tolerant LLC access. . . . .   | 107 |
| 4.4  | A general view of the NoC with different fault-tolerant mapping policies. . .  | 110 |
| 4.5  | Router architecture: (a) Conventional router; (b) fault-aware router. . . . .  | 111 |
| 4.6  | Choosing the intermediary node in the third node policy. . . . .   | 112 |
| 4.7  | Cache bank architecture: (a) A conventional 4-way set-associative cache bank; (b) modified cache bank with three subblocks in each way. . . . .  | 112 |
| 4.8  | Network latency of different policies normalized to the baseline. . . . .  | 117 |
| 4.9  | IPC of different policies normalized to the baseline. . . . .  | 118 |
| 4.10 | Network latency of different policies normalized to the baseline. . . . .  | 119 |
| 4.11 | Energy of different policies normalized to the baseline. . . . .   | 120 |
| 4.12 | Energy-delay product of different polices normalized to the baseline. . . . .  | 121 |
| 4.13 | Percentage of LLC remote vs local accesses in a 16 MB shared LLC. . . . .  | 124 |
| 4.14 | The ratio of ECC logic latency to total global LLC access Latency. . . . .   | 124 |
| 4.15 | Motivational example of LLC and NoC error coding integration . . . . .   | 125 |
| 4.16 | A multi/many-core architecture with a mesh NoC interconnect. . . . .   | 127 |
| 4.17 | The architecture of an ECC-protected LLC bank in the Conventional and CoDEC approaches. . . . .  | 129 |
| 4.18 | Data flow of LLC block through the network in Conventional vs. CoDEC approach. . . . .   | 130 |
| 4.19 | CoDEC LLC data protection over the network form source to destination. . .   | 132 |
| 4.20 | Impact of segment size on conventional versus CoDEC design. . . . .  | 134 |
| 4.21 | Performance (IPC) improvement of a system with CoDEC LLC, normalized to the baseline. . . . .  | 137 |
| 4.22 | Cache experimental results normalized to DECTED results. . . . .   | 137 |



|      |   |     |
|------|---|-----|
| 4.23 | Power consumption of NoC router’s major blocks for various ECC sizes . . . .  | 139 |
| 4.24 | Area of NoC router’s major blocks for various ECC sizes . . . . .   | 140 |
| 5.1  | High-level diagram showing HW/SW components of Relaxed Cache and their interactions. . . . .  | 151 |
| 5.2  | A Sample 4-way Cache with VDD=580mV and AFB=4. . . . .  | 152 |
| 5.3  | Encoding and decoding defect map info in Relaxed Cache . . . . .  | 153 |
| 5.4  | Abstracting Relaxed Cache knobs up to metrics familiar to a software programmer (i.e., performance, fidelity, and energy consumption). Note that (VDD = High, AFB = Mild) and (VDD = High, AFB = Aggressive) combinations are sub-optimal, hence not applicable. . . . .  | 157 |
| 5.5  | A sample code showing programmer’s data criticality declarations and cache configurations to be used by Relaxed Cache. . . . .  | 158 |
| 5.6  | Leakage energy savings for a 4-Way L1 cache with 3 relaxed ways. (energy savings are normalized to a baseline cache that uses 700mV.) . . . . .   | 160 |
| 5.7  | Fidelity results for (a) Scale and (b) Image-Smoothing benchmarks. . . . .  | 161 |
| 5.8  | Fidelity results for Edge-Detection benchmark (VDD=480mV). . . . .  | 162 |
| 5.9  | FPS-PSNR trade-offs w/ and w/o Relaxed Cache scheme (AFB=4) . . . . .   | 163 |
| 6.1  | A general system-level design space exploration problem [101]. . . . .  | 168 |
| 6.2  | Exemplar Many-core Architecture with an 8x8 mesh NoC. . . . .   | 169 |
| 6.3  | A sample case of a $2 \times 2$ NoC and four tasks mapped on its cores; a)a sample task mapping; b)uniform distribution of faults vs non-uniform distribution of faults (shaded areas represent redundancy resources and cross signs represent faulty blocks); c)cross-layer design methodology; d)DSE and Pareto-optimal solution. . . . . | 170 |
| 6.4  | Possible redundancy configuration patterns in (a) cluster-level and (b) system-level models with four redundancy nodes per cluster. . . . .   | 177 |
| 6.5  | Normalized performance improvement of different cluster-level configurations across different a) benchmarks, b)fault rates, c)amount of redundancy. . . . .   | 179 |
| 6.6  | Normalized performance improvement of different system-level configurations across different a) benchmarks, b)fault rates, c)amount of redundancy. . . . .  | 181 |
| 6.7  | Energy results across different amount of redundancy for (a) Intra-cluster, (b) Inter-cluster (system-level) configurations. . . . .  | 182 |

# LIST OF TABLES

|  | Page |
|--|------|
| 2.1 Processor Configuration . . . . .  | 33   |
| 2.2 Comparison of different cache protection schemes . . . . .               | 40   |
| 2.3 Simulated System Parameters . . . . .                                    | 50   |
| 2.4 Reliability and coding area comparison of different techniques . . . . . | 53   |
| 3.1 Architectural Specification . . . . .                                    | 63   |
| 3.2 Workload Mix, Spec2006 benchmarks are denoted with _06 . . . . .         | 80   |
| 3.3 Comparison of different cache protection schemes . . . . .               | 83   |
| 4.1 Simulation Configuration . . . . .                                       | 113  |
| 4.2 Benchmarks Description . . . . .   | 115  |
| 4.3 RDB Remapping Results . . . . .  | 118  |
| 4.4 Power and Area Overhead Results . . . . .                                | 122  |
| 4.5 Simulation Configuration . . . . .                                       | 133  |
| 4.6 Overhead and improvements comparison. . . . .                            | 140  |
| 5.1 A sample criticality table for Relaxed Cache . . . . .                   | 154  |
| 5.2 gem5 Common Parameter Settings . . . . .                                 | 159  |
| 6.1 Simulation Configuration . . . . .                                       | 175  |

# ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor, Professor Nikil Dutt, for his support and mentorship in these past years. He gave me the freedom to choose my research topic, always interacted with me to make my ideas concrete, constantly kept me motivated during the formative years of my graduate student tenure, and has consistently inspired me to do quality research. I am extremely grateful towards him for all his advices, guidance, and support throughout this journey. During these years, I learned a lot of lessons from him which helped me to be more successful in my research, career, and more importantly in my life. I also owe thanks to my candidacy and dissertation committee, Professor Ahmed Eltawil, Professor Tony Givargis, Professor Alex Nicolau, and Professor Alex Veidenbaum for their time and feedback. Their valuable comments and feedback immensely improved the quality of my dissertation.

I would also like to express my gratitude to all my past mentors and collaborators including Prof. Houman Homayoun, Dr. Luis Bathen, Dr. Gustavo Giro, Majid Shoushtari, Mark Gottscho, Ashkan Eghbal, Pooria Yaghini, Dr. Fabian Oboril and Mojtaba Ebrahimi for their great feedback, guidance, and collaborations, which resulted in an excellent research with many successful publications. They have all at some point in my life made an enormous impact, encouraged me to follow my dream, and to not give up. Especially, Prof. Houman Homayoun, who helped me to start a good research and gave me the words I needed to hear to keep moving forward.

I would like to mention special thanks to my friends and colleagues at UCI. In particular, I thank Arup Chakraborty, Dr. Hesam Kooti, Dr. Amirhossein Gholamipour, Dr. Luis (Danny) Bathen, Dr. Kazuyuki Tanimura, Codrut Stancu, Dr. Gustavo Giro, Hossein Tajik, Dr. Jun Yong Shin, Santanu Sarma, Dr. Trent Lo, Majid Shoushtari, Jurngyu Park, Bryan Donyanavard, Roger Chen-Ying Hsieh, Tiago Mck, Ashkan Eghbal, and Pooria Yaghini. They all made a great impact in my life.

I would like to thank Professor Alex Nicolau, Professor Dean Tullsen, Professor Puneet Gupta, Professor Nader Bagherzadeh, and Professor Mehdi Tahoori for their great advices, feedback, and willingness to collaborate with me.

My research at UCI has been supported in part by the National Science Foundation under NSF Variability Expedition Grant Number CCF-1029783.

Most importantly, my family deserve major gratitude. I would like to thank my parents for all their unconditional love and support, for having faith in me and unconditionally supporting me in every step I take. I would also like to thank my wife's parents for all their support. I would also like to thank my extended family members including my siblings and siblings of my wife for being a great cheering team and constantly fueling my aspirations. I feel blessed to have a family who has given me all the opportunities to succeed in life.

I am very grateful for the support, encouragement and patience of the love of my life, Marjan,

throughout the long years I worked toward my Ph.D. She made a lot of sacrifices and provided me the necessary strength to pursue my dreams, I will be forever in her dept. I also owe my son Nickan, a debt of gratitude for being such a nice and lovely boy who gave me the energy to enjoy the life and work hard.

Finally, thank God, whom I respect and thank for the opportunities I was given and placing so many wonderful people in my path.

# CURRICULUM VITAE

Abbas BanaiyanMofrad

## EDUCATION

|  |  |
|--|--|
| <b>Doctor of Philosophy in Computer Science</b><br>University of California, Irvine    | <b>2015</b><br><i>Irvine, California</i> |
| <b>Master of Science in Computer Engineering</b><br>University of Tehran               | <b>2006</b><br><i>Tehran, Iran</i>       |
| <b>Bachelor of Science in Computer Engineering</b><br>Isfahan University of Technology | <b>2003</b><br><i>Isfahan, Iran</i>      |

## RESEARCH EXPERIENCE

|  |   |
|--|---|
| <b>Graduate Research Assistant</b><br>University of California, Irvine | <b>2010–2015</b><br><i>Irvine, California</i> |
| <b>Visiting Researcher</b><br>Karlsruhe Institute of Technology (KIT)  | <b>2014</b><br><i>Karlsruhe, Germany</i>      |
| <b>Graduate Research Assistant</b><br>University of Tehran             | <b>2003–2006</b><br><i>Tehran, Iran</i>       |

## SELECTED HONORS AND AWARDS

|   |                   |
|---|-------------------|
| <b>Yahoo! Best Dissertation Student Award</b><br>University of California, Irvine                                 | <b>2014</b>       |
| <b>Karlsruhe House of Young Scientists (KHYS) scholarship</b><br>Karlsruhe Institute of Technology (KIT), Germany | <b>2014</b>       |
| <b>SIGDA-DAC PhD Forum Travel Award</b><br>University of California, Irvine                                       | <b>2014</b>       |
| <b>Design and Automation Conference (DAC) Young Student Award</b><br>University of California, Irvine             | <b>2011, 2012</b> |
| <b>ICS Dean's Fellowship for 4 years of support</b><br>University of California, Irvine                           | <b>2010–2014</b>  |
| <b>Deans honored graduate</b><br>University of Tehran, Iran   | <b>2003</b>       |

## TEACHING EXPERIENCE

|   |   |
|---|---|
| <b>Teaching Assistant</b><br>University of California, Irvine | <b>2012–2013</b><br><i>Irvine, California</i> |
| <b>Teaching Assistant</b><br>University of Tehran             | <b>2005–2006</b><br><i>Tehran, Iran</i>       |

## REFEREED JOURNAL PUBLICATIONS

|  |             |
|--|-------------|
| <b>RESCUE: Fault-tolerant Design of NUCA Caches at Near-Threshold Voltages</b><br>ACM Transaction on Architecture and Code Optimization (TACO)(under review)   | <b>2015</b> |
| <b>DPCS: Dynamic Power/Capacity Scaling for SRAM Caches in the Nanoscale Era</b><br>ACM Transaction on Architecture and Code Optimization (TACO)(second stage) | <b>2015</b> |
| <b>Exploiting Partially-Forgetful Memories for Approximate Computing</b><br>IEEE Embedded Systems Letters (IEEE-ESL)   | <b>2015</b> |
| <b>Using a Flexible Fault-Tolerant Cache to Improve Reliability for Ultra Low Voltage Operation</b><br>ACM Transaction on Embedded Computing Systems (TECS)    | <b>2015</b> |
| <b>NoC-Based Fault-Tolerant Cache Design in Chip Multiprocessors</b><br>ACM Transaction on Embedded Computing Systems (TECS)                                   | <b>2014</b> |

## REFEREED CONFERENCE PUBLICATIONS

|   |                 |
|---|-----------------|
| <b>Protecting Caches Against Multiple Bit Upsets Using Embedded Erasure Coding</b><br>IEEE European Test Symposium (ETS)  | <b>May 2015</b> |
| <b>Partially-Forgetful Memories: Relaxing Memory Guard-bands for Approximate Computing</b><br>Workshop on Approximate Computing Across the System Stack (WACAS) | <b>Feb 2015</b> |
| <b>Power / Capacity Scaling: Energy Savings with Simple Fault-Tolerant Caches</b><br>ACM/IEEE Design Automation Conference (DAC)(Invited)                       | <b>Jun 2014</b> |
| <b>Multi-Layer Memory Resiliency</b><br>ACM/IEEE Design Automation Conference (DAC)   | <b>Jun 2014</b> |

- REMEDiate: A Scalable Fault-tolerant Architecture for Low-Power NUCA Cache in Tiled CMPs** **Aug 2013**  
International Green Computing Conference (IGCC)
- Modeling and Analysis of Fault-tolerant Distributed Memories for Networks-on-Chip** **Mar 2013**  
IEEE Design, Automation, and Test in Europe (DATE)
- A Novel NoCBased Design for Fault-Tolerance of Last-Level Caches in CMPs** **Oct 2012**  
IEEE/ACM International Conference on Hardware-Software Codesign and System Synthesis, (CODES+ISSS)
- FFT-Cache: A Flexible Fault-Tolerant Cache Architecture for Ultra Low Voltage Operation** **Oct 2011**  
International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES)

## **SOFTWARE**

- FFT-Cache framework** <https://github.com/abanaiya/fft-cache>  
*C simulation framework for fault-tolerant cache design*

# ABSTRACT OF THE DISSERTATION

Resilient On-Chip Memory Design in the Nano Era

By

Abbas BanaiyanMofrad

Doctor of Philosophy in Computer Science

University of California, Irvine, 2015

Professor Nikil Dutt, Chair

Aggressive technology scaling in the nano-scale regime makes chips more susceptible to failures. This causes multiple reliability challenges in the design of modern chips, including manufacturing defects, wear-out, and parametric variations. By increasing the number, amount, and hierarchy of on-chip memory blocks in emerging computing systems, the reliability of the memory sub-system becomes an increasingly challenging design issue. The limitations of existing resilient memory design schemes motivate us to think about new approaches considering scalability, interconnect-awareness, and cost-effectiveness as major design factors. In this thesis, we propose different approaches to address resilient on-chip memory design in computing systems ranging from traditional single-core processors to emerging many-core platforms. We classify our proposed approaches in five main categories: 1) Flexible and low-cost approaches to protect cache memories in single-core processors against permanent faults and transient errors, 2) Scalable fault-tolerant approaches to protect last-level caches with non-uniform cache access in chip multiprocessors, 3) Interconnect-aware cache protection schemes in network-on-chip architectures, 4) Relaxing memory resiliency for approximate computing applications, and 5) System-level design space exploration, analysis, and optimization for redundancy-aware on-chip memory resiliency in many-core platforms. We first propose a flexible fault-tolerant cache (FFT-Cache) architecture for SRAM-based on-chip cache memories in single-core processors working at near-threshold voltages. Then, we ex-



tend the technique proposed in FFT-Cache, to protect shared last-level cache (LLC) with Non-Uniform Cache Access (NUCA) in chip multiprocessor (CMP) architectures, proposing REMEDIATE that leverages a flexible fault remapping technique while considering the implications of different remapping heuristics in the presence of cache banking, non-uniform latency, and interconnected network. Then, we extend REMEDIATE by introducing RESCUE with the main goal of proposing a design trend (aggressive voltage scaling + cache over-provisioning) that uses different fault remapping heuristics with salable implementation for shared multi-bank LLC in CMPs to reduce power while exploring a large design space with multiple dimensions and performing multiple sensitivity analysis. Considering multi-bit upsets, we propose a low-cost technique to leverage embedded erasure coding (EEC) to tackle soft errors as well as hard errors in data caches of a high-performance as well as an embedded processor. Considering non-trivial effect of interconnection fabric in memory resiliency of network-on-chip (NoC) platforms, we then propose a novel fault-tolerant scheme that leverages the interconnection network to protect the LLC cache banks against permanent faults. During a LLC access to a faulty area, the network detects and corrects the faults, returning the fault-free data to the requesting core. In another approach, we propose CoDEC, a Co-design approach to error coding of cache and interconnect in many-core architectures to reduce the cost of error protection compared to conventional methods. Proposing a system-wide error coding scheme, CoDEC guarantees end-to-end protection of LLC data blocks throughout the on-chip network against errors. Observing available tradeoffs among reliability, output fidelity, performance, and energy in emerging error-resilient applications in approximate computing era motivates us to consider application-awareness in resilient memory design. The key idea is exploiting the intrinsic tolerance of such applications to some level of errors for relaxing memory guard-banding to reduce design overheads. As an exemplar we propose Relaxed-Cache, in which we relax the definition of faulty block depending on the number and location of faulty bits in a SRAM-based cache to save energy. In this part of thesis, we aim at cross-layer characterization and optimization of on-chip

memory resiliency over the system stack. Our first contribution toward this approach is focusing more on scalability of memory resiliency as a system-level design methodology for scalable fault-tolerance of distributed on-chip memories in NoCs. We introduce a novel reliability clustering model for effective shared redundancy management toward cost-efficient fault-tolerance of on-chip memory blocks. Each cluster represents a group of cores that have access to shared redundancy resources for protection of their memory blocks.

# Chapter 1

## Introduction

### 1.1 Nano Era Design Trends and Challenges

We categorize the general trends and challenges of design in the Nano era in three major groups of *Technology*, *Chip Design*, and *Application* that we briefly describe in below.

#### 1.1.1 Technology Trend

Aggressive technology scaling in Nano era coupling with environmental issues make CMOS circuits more susceptible to various sources of failures from soft errors to hard errors [118]. Hence, there are major trends and challenges in circuit reliability with many evolving techniques for dealing with them [48]. The main sources of hard faults in deep sub-micron technology nodes include manufacturing defects, process variation, and wearout induced failures [48].

### 1.1.2 Chip Design Trend

As semiconductor manufacturers continue to push for Chip-Multiprocessor technology in recent platforms such as IBM cell processor [70], Intel's 48-core IA-32 processor [72], Intel Larrabee many-core [140], Teraflops Research Chip [151], Single-chip Cloud Computer (SCC) [64], Intel Xeon Phi [78], NVIDIA Tesla [97], and Tiler TILE64 processor [31], many-core architectures are become one of the major design options of next-generation large-scale systems. Since reliability and efficiency are challenges of large-scale system design [149], new challenges emerge in terms of effective and reliable design of different components in recent platforms [37]. Furthermore, since memory system constitutes a majority of the silicon in a large-scale system, the memory subsystem has become vulnerable to a host of manufacturing, environmental, and operational failure/degradation mechanisms that affect the overall resiliency of the system. By increasing the number, amount, and hierarchy of on-chip memory blocks in emerging large interconnected many-core architectures, reliability of memory sub-system becomes an increasingly challenging issue in the design of such platforms [69].

### 1.1.3 Application Trend

On the other hand, most of the applications in emerging domains such as big data, cyber physical systems, social networks and so on are memory-intensive. Also, there exist some class of applications such as embedded, multimedia, and Recognition, Mining and Synthesis (RMS), are inherently resilient to some level of errors [92]. Traditional fault-tolerant approaches which try to protect the memories against all existing faults and errors, are not efficient for such applications. Hence, techniques for effective memory protection needed to be addressed in the design of emerging computing systems. (From this point forward, an emerging computing system is a large-scale platform such as many-core which is running new class of applications such as memory-intensive benchmarks.)

## 1.2 Memories and Errors

Figure 1.1 shows the typical hardware/software abstraction layers for computing systems. Each row of Figure 1.1 describes the system abstraction layer, the memory abstraction at that level, and typical manifestations of memory errors that can compromise system resiliency. The last column of Figure 1.1 describes mitigation schemes in the face of memory error manifestations at that level of abstraction. Memory errors manifest themselves in different ways across abstraction stack. For instance, an unstable memory cell at the circuit/device level can cause a bit failure at the memory logic level, which in turn might propagate up the abstraction stack as a faulty memory access at the architecture level, a wrong function call or system halt at OS-level, and finally an output error or an exception at application layer. Figure 1.1 represents a symbolic abstraction of memory errors over the entire hardware/software system stack. Traditionally, memory resilience has been addressed via disparate techniques at each level of design abstraction, while newer efforts attempt to couple strategies across layers with the goal of improving system efficiency for energy, heat dissipation, lifetime, cost, etc. Furthermore, efforts in relaxed and approximate computing attempt to create designs that can trade off application quality for these system efficiency goals. To understand memory faults, we can classify them by their temporal behaviors (persistence) as well as their causes. With respect to persistence, a memory fault can be permanent or transient. Permanent faults persist indefinitely in the system after occurrence, while transient faults manifest for a relatively short period of time after occurrence. Furthermore, causes of memory faults can be hard or soft. Hard faults are static and caused by device failure or wear-out failure. In contrast, soft faults are dynamic and are typically caused by the operating environment.

In general, memories suffer from different sources of unreliability that can be classified into three main groups:


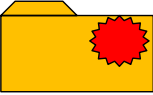

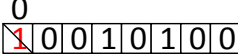
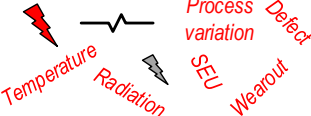
| System Abstraction        | Memory Abstraction   | Error Manifestation  | Mitigation                                      |
|---------------------------|--|--|---|
| <b>Application</b>        | <i>Program, Data Structures, Files, Libraries</i>          |  <i>Incorrect Output, Infinite Loop, Crash</i>                    | <i>TMR, Recomputing, Information Redundancy</i> |
| <b>System (OS)</b>        | <i>Main Memory, File System Address Space, Heap, Stack</i> |  <i>Wrong Pointer, Erroneous System Call, Trap</i>                | <i>Error Check, Assertion, Exception</i>        |
| <b>Architecture (ISA)</b> | <i>Buffers, Register File, L1\$, L2\$, SPM</i>             |  <i>Faulty Word, Cache Block, Way</i>                             | <i>ECC, Spare Elements, Disabling</i>           |
| <b>Logic</b>              | <i>Memory Cells, Bit Arrays</i>                            |  <i>Bit flip, Stuck @ 0/1</i>                                     | <i>Adding Parity, Spare Cells</i>               |
| <b>Circuit/ Device</b>    | <i>Voltage, Current, Transistor, Cell</i>                  |  <i>Low Noise Margin, Unstable Cell, V<sub>th</sub> Variation</i> | <i>Larger transistors, More transistors</i>     |

Figure 1.1: Memory Abstractions, Errors, and Opportunities.

1. *Manufacturing.* Worsening manufacturing imperfections in nanoscale technologies result in increasing variability of device and circuit-level parameters. This process variation particularly affects transistor threshold voltages through random dopant fluctuation (RDF), increasing the likelihood of memory cells failing permanently due to insufficient noise margins at a given supply voltage.
2. *Environmental.* Alpha particle radiation coming from the operating environment can cause single event upsets (SEU). Combined with weakened noise margins from manufacturing effects, memory cells are also becoming more susceptible to SEU, impacting their soft error resilience [2]. Noise stemming from variations in the supply voltage and thermal effects can also cause memory faults exhibiting dynamic and random behavior.
3. *Aging and Wearout.* Depending on the type of technology used, memory cells can age, reducing their performance, data retention capability, and power consumption. Aging can

lead to memory wearout, resulting in permanent faults.

Different memory technologies suffer from various sources of unreliability. Volatile memories such as SRAM and DRAM mostly suffer from manufacturing defects and environmental issues that lead to hard and soft errors, respectively. Endurance is not an issue in SRAM and DRAM. In contrast, different nonvolatile memories (NVMs) have their own sources of unreliability. For flash and phase change memory (PCM), wearout is the primary source of unreliability due to limited write endurance. PCMs also suffer from hard and soft errors [162]. Other emerging NVMs such as MRAM and its newer cousin STT-RAM also suffer from hard and soft errors. However, for these devices, wearout is not as great of a reliability threat, because they have large write endurances similar to that of SRAM.

The design of reliable computer systems has a rich history spanning several decades: variants of spatial, temporal, and information redundancy have been exploited to improve reliability. Memory systems also deploy these forms of redundancy to achieve resilience across various layers of system abstraction. Additionally, memory designers have leveraged a variety of other memory-specific techniques.

In next section, we provide a sampling of common techniques used for reliable memory design at the architectural level. A significant body of research exists on the design of a reliable memory hierarchy comprising multiple levels of caches and main memory.

### **1.3 State-of-the-art Research Efforts**

A significant amount of literature targets the memory reliability concerns due to different aspects of vulnerability including transient errors, manufacturing defects, process variation, wearout, and stability in near/sub-threshold operation [54]. We briefly review the prior related work in three major categories including: Reliable and fault-tolerant cache design,

memory reliability concerns in multi/many-core platforms, and techniques for approximate computing on memories or application-aware memory resiliency.

**Reliable and Fault-tolerant Cache/Memory design** A significant body of research exists on the design of a reliable memory hierarchy comprising multiple levels of caches and main memory [54]. We can categorize resilient SRAM cache design efforts into three main groups of Cell/Circuit-Level Techniques, Error Coding Techniques, and Architecture-Level Techniques. Many of these have the common property of fault-tolerant voltage-scalable (FTVS) design, because low voltage operation while critical for achieving power and energy savings is the primary driver behind unreliable memories. In general, regardless of whether the fault-tolerant design is done at the cell, circuit, coding, or architecture level, there is a trade-off in terms of memory capacity and area. This may be due to larger memory cells, spare or redundant cells, error correction logic, or a reduced amount of reliable memory available for use by the application.

Fault-tolerant memory designs have often used simple techniques such as adding redundant rows/columns to the memory array [139] or applying memory down-sizing techniques by disabling a faulty row or cache line (block) [122]. Many recent architecture-level schemes deploy more complicated redundancy or capacity downsizing techniques to improve the reliability of cache memories. Information redundancy via error coding is also commonly used to improve the reliability of memory components. Wide ranges of error detection and correction codes (EDC and ECC, respectively) have been used [96]. Typically, EDCs are simple parity codes, while the most common ECCs use Hamming [67] or Hsiao [73] codes. ECC is proven as an effective mechanism for handling soft errors. For NVMs that have limited write endurance, various wear-leveling approaches have been proposed to mitigate aging and extend memory lifetime. In the domain of embedded systems, design of reliable software-controlled scratchpad memories (SPMs) has also received great attention recently, including efforts that address the reliability of SPMs for chip-multiprocessors (E-RoC [25] and SPMVisor [27]), or



for hybrid memories (FTSPM [110]).

**Memory reliability in multi/many-cores** There is a large body of previous work on fault-tolerant design of on-chip memories and caches, mostly focused on either single bank or multi-bank caches with uniform access latency in single-core processors [54]. However, they face severe limitations when applied to emerging multi-core/many-core architectures where the number of access points (cores) and memory banks are increasing, large on-chip memories are shared and distributed via interconnected network among all cores, memory access latencies are not unified, interconnect backbone is also erroneous and affects memory resiliency, and cost of uniform fault-tolerance is very high [19]. Hence, most of the current fault-tolerant schemes are not modular, scalable, and cost-efficient enough to be applied to these architectures. Only a few efforts somehow addressed on-chip memory reliability in CMP or NoC platforms [7, 154, 25, 5, 16].

**Memory Reliability Relaxation** A significant amount of work has proposed different hardware and software techniques that trade-off reliability or application fidelity to gain benefits in energy, performance, lifetime, or yield for applications that can tolerate some level of error in exchange for loss of quality (e.g., RMS applications). Many studies on approximate computing era have shown that a variety of applications have a high tolerance to errors [51, 45, 92, 109, 95]. While most of the previous efforts on approximate computing have focused on the computational part, there exist only a few works that have addressed relaxing reliability in memories which mainly focused on off-chip memories [100, 135].

## 1.4 Motivation

In this section, we first define some general as well as introduced terminology we leverage throughout the thesis.

- Resiliency: The power or ability to recover from or adjust easily to a bad/unlikely situation.
- Reliability: The ability to be relied on or depended on, as for accuracy, honesty, or achievement.
- System Reliability: The ability of a system or component to perform its required functions under stated conditions for a specified period of time.
- System Resiliency: The ability of a system to continue to function correctly, sometimes in a degraded fashion, in the presence of faults, errors, or other variation.

Resilience is a measure of a system's ability to bounce back from a failure to continue to offer some level of performance (possibly not the original level of performance).

Design for Resiliency is a way of achieving Reliability by trading off other system metrics like performance. The reliability issues and limitations of common schemes mentioned above motivate us to think about new approaches that address cost-efficient resilient design of on-chip memory in emerging multi/many-core platforms. We consider all limiting factors of traditional and current approaches as well as emerging technology/architecture/application requirements and challenges in design of our new approaches. Some of the key *resiliency design features* that are either missed or addressed inefficiently in previous approaches, considered in design of our novel approaches and include: *flexibility, scalability, interconnect-awareness, multi-layer design, cost-efficiency, non-uniformity, redundancy-awareness, and application-awareness*.

By reviewing the previous approaches as well as current efforts and design trends, we gain multiple important observations. We summarize the key observations as follows:

- Inefficiency of previous fault-tolerant cache/memory design approaches for emerging applications and platforms

- Cost-efficient resilient memory design becomes a major challenge in design of emerging platforms
- For many application domains, it is acceptable to trade some design metrics (performance, yield, fidelity, etc.) to increase the efficiency.
- New approaches required to provide memory resiliency in emerging computing systems

The key missed resiliency features as well as our observations mentioned above, motivates us to approach to memory resiliency in a different way. Our key idea in this thesis is that we propose novel approaches with the goal of *Design for Memory Resiliency or Resilient Memory Design* other than traditional design for Reliability or fault-tolerance memory design. That is, we try to address most of the key required resiliency features in our new approaches to memory resiliency to make it more efficient and reduce the costs by considering different design parameters such as possible design trade-offs, multi-layer design possibilities, platform features (resources and constraints), and application features (requirements and constraints).

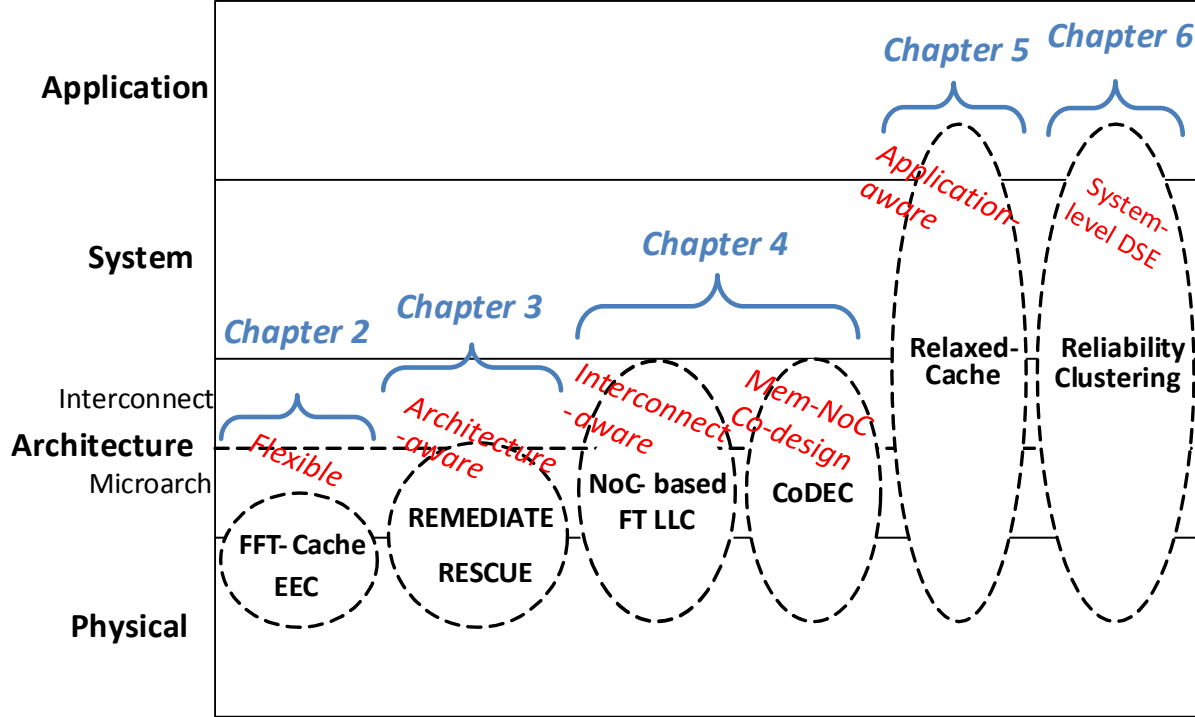


Figure 1.2: Summary of the thesis main contributions.

## 1.5 Thesis Contributions

In this thesis, we make multiple contributions and propose novel approaches in different layers of system hierarchy to address resilient memory design issues in computing systems ranging from traditional single-core processors to emerging many-core platforms as indicated in Figure 1.2. We classify our proposed approaches in five main categories: 1) Flexible low-cost Cache Design (Chapter II), 2) Scalable Resilient Cache Design (Chapter III), 3) Interconnect-aware resilient cache design (Chapter IV), 4) Application-aware Memory Resiliency (Chapter V), and 5) System-level Memory Resiliency Design Space Exploration and Optimization (Chapter VI).

In this thesis, we make the following contributions:

- Proposing two flexible and low-cost approaches to protect cache memories in single-core

processors against permanent and transient faults, respectively.

- **FFT-Cache**, a flexible cache architecture, 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.
- **EEC**, an embedded erasure coding approach to protect cache memories against multi-bit upsets. The EEC scheme employs fast bit-interleaved parity coding as error detection in combination with optimal parity-based erasure coding as error correction over set-associative cache memories. Reserving a part of cache (e.g., one way) as erasure storage, EEC trades cache capacity for fast and low-cost error correction.
- Proposing two scalable fault-tolerant approaches to protect last-level cache (LLC) with non-uniform cache access (NUCA) in chip multiprocessors (CMPs).
  - **REMEDiate** to protect shared NUCA caches in tiled-based multicore platforms. REMEDIATE achieves fault-tolerance through redundancy from multiple banks to maximize the amount of fault remapping, and minimize the amount of capacity lost in the cache when the failure rate is high.
  - **RESCUE** extends REMEDIATE to enhance fault-tolerant voltage-scalable design of shared NUCA caches in multi/many-core core platforms. While the main goal in REMEDIATE is to develop a remapping-based fault-tolerant scheme for protection of shared LLC in Tile-based CMPs while operating in low voltages, the main goal in RESCUE is to propose a design trend (aggressive voltage scaling + cache over-provisioning) that uses different fault remapping heuristics with salable implementation for shared multi-bank LLC in non-tiled-based CMPs to reduce power while exploring a large design space with multiple dimensions and

performing multiple sensitivity analysis.

- Proposing two interconnect-aware LLC protection schemes in many-core platforms.
  - **FT-LLC**, NoC-based fault-tolerance of LLC in mesh-based CMPs. This approach leverages the NoC fabric to implement a remapping-based fault-tolerant scheme protecting the LLC cache banks against permanent faults in a scalable and efficient way with minimal overhead. In this approach, during an LLC access to a faulty block, the components inside the network (routers) are responsible for detecting and correcting the faults and then returning the fault-free data to the corresponding core.
  - **CoDEC**, a Co-design of error coding in cache and interconnect in many-cores. CoDEC deploys a united error coding scheme that guarantees the end-to-end protection of LLC data blocks throughout the on-chip network against both hard and soft errors. CoDEC integrates strong multi-bit ECC in cache banks with s2s error coding of interconnect while removing their overlapped redundancy.
- Introducing Partially-Forgetful Memory (PFM) as an application-aware Memory Resiliency relaxation approach.
  - **Relaxed-Cache** as an exemplar to address memory variability and approximate computing challenges for employing partially-forgetful SRAM caches. Relaxed Cache lets the application programmer adjust the cache guard-banding knobs and therefore adapt its reliability and capacity to the application's demands. Accordingly, the application programmer needs to identify critical memory objects as well as major criticality and computational phases of the application.
- Proposing approaches to system-level Design Space exploration, analysis, and optimization for on-chip memory resiliency.

- **System-level Memory Resiliency**, We propose a system-level design methodology for scalable fault-tolerance of distributed on-chip memory blocks in many-core platforms.
- **Reliability Clustering**, to model distribution of shared redundancy resources. A novel approach to manage shared redundancy resources for distributed fault-tolerance of on-chip memory components in many-cores. Each cluster represents a group of cores that have access to shared redundancy resources for protection of their own memory blocks which include LLC banks in our approach.

## 1.6 Thesis Organization

The rest of this thesis is organized as follows. First, Chapter II presents FFT-Cache and EEC as two flexible and low-cost approaches that trade performance for energy efficiency and reliability, respectively. FFT-Cache is 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. EEC leverages embedded erasure coding to tackle multi-bit upsets in caches. In Chapter III, we introduce RESCUE, a highly flexible fault-tolerant and energy-efficient cache design that by reconfiguring its internal organization can efficiently tolerate the large number of NUCA LLC failures that arise when operating in the near-threshold region in CMP architectures. In Chapter IV, we consider the effect of interconnect in design of fault-tolerant LLC in NoC-based architectures in two different approaches. Next, in order to perform an application-aware resilient memory design, in Chapter V, we introduce Partially-Forgetful Memory approach to propose a relaxed reliability cache design, Relaxed-cache, that relaxes the memory guard-banding to save energy in SRAM-based cache memories. Next, in order to perform a system-level design space exploration, in Chapter VI, we propose a novel system-level design methodology by introducing the reliability clustering approach to explore the effective management of shared redundancy resources for on-chip memory protection in many-core platforms. Finally, Chapter VII presents the summary of thesis and our directions for future work.



# Chapter 2

## Flexible and Low-Cost Fault-tolerant Cache Design

### 2.1 Introduction

On-chip caches are known to consume a large portion (about 30%-70%) of total processor power [158, 164] and their size will continue to grow due to device scaling coupled with performance requirements. Therefore, it becomes critical to manage the power and reliability of the caches in order to reduce total power consumption while maintaining the reliability of the entire processor system. Traditionally, voltage scaling has been used to reduce the dynamic and the leakage power consumption of the cache. However, aggressive voltage scaling causes process-variation-induced failures in the SRAM cells. An SRAM cell can fail due to an access time failure, a destructive read failure or a write failure [2, 133]. Figure 1 represents the failure rate of an SRAM cell based on the operational voltage in a 90nm technology [113]. To save power while achieving an acceptable manufacturing yield of 99.9% for 64KB L1 and 2MB L2 caches, a minimal operational voltage must be selected. From

Figure 1 we can see that the probability of failure for each cache array must be kept at less than 1 out of 1000 to achieve this yield. Based on this assumption, we estimate the minimum operational voltage for a 64KB L1 is 620 mV and for a 2MB L2 660 mV, and we are not able to further reduce the operational voltage without incurring cell failures.

Since applications may not be tolerant to even a single bit error, typically caches must be operated at a high Vdd to ensure a very low probability of failure, leading to high energy consumption. However, by exploiting mechanisms that allow a cache to become inherently resilient to a large number of cell failures, we can operate the cache at a lower Vdd and thus gain significant energy savings. There is a large body of previous work on fault-tolerant cache design at different layers of abstraction that we review in next section. However, most of them are not efficient for high fault rates arising from operation in the near-threshold regime, and these techniques incur significant overheads. To address these issues, in this chapter we first propose an approach that aims to: 1) design a low power and flexible fault-tolerant cache architecture that can detect and replicate SRAM faults arising from operation in the near-threshold voltage region; 2) minimize non-functional or disabled cache area to lessen impact on processor performance; and 3) tolerate cache faults as much as possible when the fault rate is high or voltage is very low.

On the other hand, exponentially growing number of devices per chip along with a gradual reduction in operating voltage lead to an increase in the effective *Soft Error Rate* (SER) of circuits over the past few years [76, 53]. Additionally, it has been shown that the SRAM arrays are the major contributor to the overall system SER (by more than 95% [56]), hence, applying an effective technique to mitigate the errors in these components is of decisive importance.

In previous technology nodes, a soft error caused by a single particle strike in an SRAM array was very likely to only affect a single cell. This kind of errors are called *Single Event Upsets* (SEUs) and could be corrected by a low-cost single error correction technique [53, 29].

With smaller device geometries in nanoscale technology nodes, a single high energy particle strike might affect several adjacent cells resulting in *Multiple Bit Upsets* (MBUs). Recent experiments show that not only the ratio of MBUs to SEUs increases, but also the average number of affected cells by an MBU incident grows [76, 103]. Therefore, an effective MBU correction technique has to be used in memory arrays to ensure their reliability with minimum cost.

A variety of error-correction-codes (ECC) have been proposed to protect SRAM cache memories against soft errors [43, 82, 117, 155, 132, 106, 84]. However, the majority of existing MBU correction techniques are only able to correct a predefined number of errors per word. The area, latency, and power overheads of these techniques exponentially grow by increasing the MBU size. Moreover, due to high complexity and overhead of ECC encoding/decoding, it is hard to implement such techniques in current processors with affordable cost [82, 132]. Hence, to protect caches against MBUs, it is essential to reduce the complexity and overheads of the error protection techniques.

In this chapter, we first present Flexible Fault-Tolerant Cache (FFT-Cache) [22, 23], a cache architecture that uses a flexible defect map to efficiently tolerate the large number of faults when operating in the near threshold region. FFT-Cache specifies a portion of faulty cache blocks as redundancy and remaps other faulty cache word-lines and blocks to them. This is accomplished by using either block-level or line-level (set-level) replication in the same set or between two or more sets. In the remainder of this chapter, we refer to every physical cache word-line, which may contain multiple cache blocks as a line or set. FFT-Cache leverages its flexible defect map to configure the remapping of faulty data blocks either inside the same set or among different sets. FFT-Cache configuration is done in two phases: first we categorize the cache lines based on degree of conflict between their blocks, and then we use a graph coloring algorithm to optimize the faulty block remapping and reduce the granularity of redundancy replacement. Using this approach, FFT-Cache initially attempts

to replicate faulty blocks inside each line; and otherwise it attempts to replicate faulty blocks among different lines. Our FFT-Cache approach significantly saves power consumption, while incurring only a small (5%) performance degradation and a small (13%) area overhead when operating in near-threshold voltages.

In the second part of this chapter, we propose an error protection approach for cache memories using an erasure coding technique [18]. In this approach, considering the fact that error detection can be done at much lower cost than error correction [112], we employ interleaved parity codes per-word to detect errors in each cache word or block. The interleaving distance is set based on the distribution of actual MBU patterns and their respective probabilities obtained from a detailed technology dependent analysis. For error correction, we adapt the notion of traditional erasure coding to define an *Embedded Erasure Coding* (EEC) approach that performs coding across multiple cache blocks in the same row. Therefore, it reserves a cache way to store erasure check bits for the remaining ways. Moreover, it is implemented in a configurable way that depending on the desired performance/reliability, the last way in different parts (banks) of a cache can be used as erasure storage or normal data storage. We evaluate the proposed scheme for both L1 and L2 cache of a high-performance processor and also for the L1 cache of an embedded processor. Our simulation results for L1 and L2 caches of the high-performance processor show that EEC detects and corrects 100% of injected MBUs with less than 3% performance and 4% energy overheads. We also compare the reliability and area overhead of our scheme against common 1-dimensional and 2-dimensional error coding approaches. Our simulation results show that EEC provides higher reliability, faster error recovery, and less area and energy overheads compared to the state-of-the-art techniques.

In summary, three main advantages of our proposed technique over existing ones are:

- Cost-efficient and easy to implement in current processors.

- Configurability to trade performance versus reliability by using more/less cache ways as error correction storage.
- Much lower error recovery time compared to similar techniques.

These features make it scalable to larger caches in multicore processors and also for protecting against permanent faults. For example, in adaptive low power systems where the voltage is aggressively scaled to trade performance versus energy efficiency, our proposed technique is extremely useful.

## 2.2 Related Work

Several fault-tolerant techniques have been proposed at multiple levels of abstraction to improve the cache yield and/or lower the minimum achievable voltage scaling bound. In general, we can categorize resilient SRAM cache design efforts into three main groups as we explore below.

### 2.2.1 Circuit-level Techniques

The root of most SRAM reliability problems is the cell noise margin. At low supply voltages, noise margins are reduced, increasing susceptibility to data corruption caused by environmental factors described earlier. Furthermore, variability in cell noise margins requires a statistical approach to designing a reliable memory array and choice of minimum supply voltage, which must be increased to maintain yield under large variations [2]. Engineers have designed larger memory cells using more transistors and/or larger transistors to increase mean noise margins and/or reduce margin variability, but these come at the cost of reduced area efficiency and sometimes power. Besides the conventional 6T SRAM cell, sev-

eral other designs, including 8T SRAM cell [42, 44], 10T SRAM cell [38], and 11T SRAM cell [111] have been proposed. All of these SRAM cells improve read stability, though the stability of the inverter pair remains unchanged. Most of these cells incur a large area overhead that poses a significant limitation for performance and power consumption of caches. For instance, [90] proposed a Schmidt trigger based 10T SRAM cell with inherent tolerance towards process variation using a feedback-based mechanism. However, this SRAM cell requires a 100% increase in the area and about 42% increase in the access time for low voltage operation.

### 2.2.2 Error Coding Techniques

A wide range of ECC techniques have been proposed to protect memory arrays against soft errors [112]. Single-error correction double-error detection (SECDED) is proven as an effective mechanism for handling SEUs [43]. However, in a high-failure rate situation, simple coding schemes like SECDED [43] are not practical because of the strict bound on the number of tolerable faults in each protected data chunk [82]. As technology advanced, a variety of more complex coding schemes with higher level of protection have been proposed to protect memories against multi-bit errors [117, 46, 161, 155, 141]. Double error correction triple error detection (DECDED), two-dimensional ECC (2D-ECC) [82], multiple-bit segmented ECC (MS-ECC) [46], Hi-ECC [155], variable-strength ECC (VS-ECC) [4], and Memory Mapped ECC [161] are some of the more notable schemes. Besides common codes such as Hamming [67] and Hsiao [73], other strong codes such as BCH [155], OLSC [46], and Reed Solomon [96] have also been used to gain strong error detection. However, ECC techniques generally come at high cost due to significant memory storage and encoding/decoding logic overheads [82, 132]. Despite this, ECC remains a popular method for memory resilience due to its effectiveness against soft errors, and the lack of involvement from other layers of abstraction. In addition, another approach to reduce memory vulnerability to soft errors is

to use a bit-interleaved memory with SECDED to correct small-scale physically-contiguous multi-bit errors [104, 121, 55]. However, power, area, and delay overheads grow significantly as the interleaving factor increases beyond four, depending on the memory design [6].

Typically, ECC schemes used in caches and memory rely on codes that detect and correct errors since the positions of the errors are not known a priori. On the other hand, *Era-  
sure coding* is a well-known technique for correcting errors when the position of an error is known [128]. By separating error detection from error correction in common error coding approaches used in memory arrays, erasure codes can be leveraged to recover erroneous parts as an alternative to common error correction technique. Some schemes inspired by erasure coding have been recently proposed to protect cache memories against soft errors [82, 106, 84].

### 2.2.3 Architecture-Level Techniques

Several architectural techniques have also been proposed to improve reliability of on-chip cache by using either redundancy or cache resizing. Earlier efforts on fault-tolerant cache design use various cache down-sizing techniques by disabling a faulty line or block of cache. Ozdemir et al. [122] proposed Yield-Aware cache in which they developed multiple techniques that turn off either cache ways or horizontal regions of the cache which cause delay violation and/or have excessive leakage. Agarwal et al. [2] proposed a fault tolerant cache architecture in which the column multiplexers are programmed to select a non-faulty block in the same row, if the accessed block is faulty. Similarly, PADed cache [143] uses programmable address decoders that are programmed to select non-faulty blocks as replacements of faulty blocks. Sasan et al. [138, 137] proposed a number of cache architectures in which the error-prone part of the cache is fixed using either a separate redundancy cache or parts of the same cache. RDC-cache [138] replicates a faulty word by another clean word in the last way of next cache bank. Other efforts, such as In-Cache Replication (ICR) [166]

and Multi-Copy Cache (MC2) [41], use data replication to improve reliability. Schemes such as Replication Cache [165] and ZerehCache [10] use external spare caches. Similarly, variants of fault-grouping and fault remapping have been used to tolerate faulty cache blocks without adding any spare elements, but by using other parts of the cache, such as GRP2 [131], RDC-Cache [138], Abella [1], Archipelago [9], and FFT-Cache [22]. Wilkerson et al. [156] propose two schemes called Word-disable (WDIS) and Bit-fix (BFIX) that also could fall under this category. The buddy cache [87] pairs up two non-functional blocks in a cache line to yield one functional block. A similar idea was proposed independently in [131]. The salvage cache [86] improves on this technique by using a single non-functional block to repair several others in the same line. However, all of these methods are not efficient in the near-threshold region with high fault probabilities. Indeed at such a low voltage, the cache would be effectively downgraded to just a fraction (e.g., 30%) of its original size, which results in a large performance degradation in terms of IPC across standard benchmarks.

Some hybrid schemes combine multiple techniques mentioned earlier to minimize the costs of memory protection. [169] minimizes area overhead through joint optimization of cell size, redundancy, and ECC; and Ndai [119] performs circuit-architecture co-design for memory yield improvement. More recent architectural schemes for cache resilience address newer challenges for multi and many-core platforms, such as scalability [24, 19], variation in fault behaviors [46], non-uniform memory access latency [24], limited shared redundancy [20], low overhead multi-VDD support [63], and reliability-driven error protection [126].



## 2.3 FFT-Cache

### 2.3.1 Proposed Architecture

In this section, we first describe the proposed FFT-Cache architecture that uses a Flexible Defect Map (FDM) to efficiently tolerate SRAM failures. Next, we present the cache configuration that includes FDM generation and configuration stages to configure the architecture for fault-tolerant data access.

#### FFT-Cache Organization

The FFT-Cache architecture has two banks of data that can be accessed concurrently, and they include multiple cache lines, with each line including multiple blocks. FFT-Cache achieves fault-tolerance by using a portion of the faulty cache blocks as redundancy to tolerate other faulty cache lines and blocks. Using this approach, FFT-Cache tries to sacrifice a minimal number of cache lines to minimize performance degradation and tolerate the maximum amount of defects at near-threshold operating voltage. This is done by using either block-level or line-level replication in the same set or between two sets. The information of faulty locations is kept in a Flexible Defect Map (FDM) which is then used to configure the faulty block remapping. To replicate the faulty subblocks in a line (called a host line), our scheme tries to find a faulty block in the same line or in another line that has no conflict with other blocks in the host line. We refer to such a block as a Target block. Depending on whether the Target block is in the same line (host line) or in another line, it called Local or Global Target block, respectively. FFT-Cache always tries to find a local target block. If FFT-Cache cannot find a local target block, it searches for a global target block. Finally, if it fails to find either a local or global target block, it tries to find another faulty line (called a target line) that has no conflict with the host line. It then sacrifices the target

line to replicate all faulty blocks of the host line. Thus, based on the earlier discussion, the sacrificial target line/block could be one of: 1) Local Target Block, 2) Global Target Block, or 3) Target Line. Note that a local target block can be accessed in the same cycle as the host line, and does not require any additional access overhead. This is not true for a global target block or a target line, for which two consecutive accesses are required if the global target block or target line are in the same bank as the host line. In order to access the host line and global target line/block in parallel, and to minimize the access latency overhead, the host line and target line should be in different banks. Also, since target blocks/lines do not store any independent data, they are considered non-functional. Therefore, the target lines are not addressable as data lines and thus they are removed from the address scope of the cache. This could impact performance as it reduces the effective size of the cache. A major advantage of FFT-Cache over other fault-tolerant methods is that we minimize the number of non-functional and target lines by initially attempting to find a local target block to sacrifice. If a local target block is not found, FFT-Cache then attempts to find a global target block. Finally if the first two attempts are not successful, FFT-Cache sacrifices a different cache line as a target line, as other fault-tolerant methods do. In our fault-tolerant cache architecture, each cache access in low power mode first accesses the FDM. Based on the fault information of the accessed line, the target block/line may be accessed from another bank (in case of a global target block or target line). Then based on the location of target block/line retrieved from the FDM, one or two levels of MUXing are used to compose a fault free block by choosing appropriate subblocks from both host and target blocks (lines). Figure 2.1 outlines the flowchart for accesses using the FFT-Cache.

### **FFT-Cache Configuration**

We now describe the configuration process for FFT-Cache. Initially, a raw defect map is generated at boot time: using the cache memory Built-In Self-Test (BIST) unit, the L1

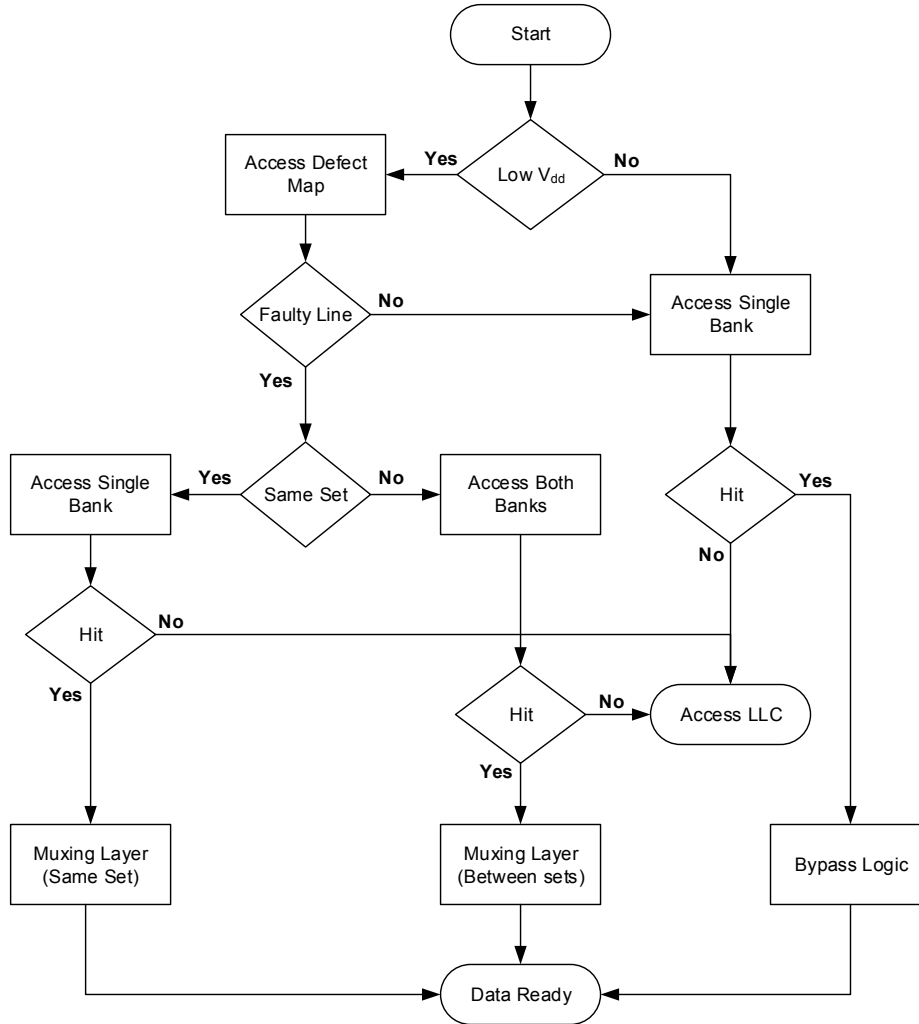


Figure 2.1: FFT-Cache Access Flowchart.

and L2cache(s) are tested under low voltage conditions. The output of the BIST is used to initialize the FDM. If there are multiple operating points for different combinations of voltage, temperature and frequency, the BIST operation is repeated for each of these settings. The obtained defect map is then modified and processed to be usable with FFT-Cache. Updating the FDM is done at full voltage, using a simple algorithm that we explain next. The configuration information can be stored on the hard-drive and is written to the FDM at the next system boot-up. In addition, in order to protect the defect map and the tag arrays, we use the well-studied 8T SRAM cell [152] that has about 30% area overhead for these relatively small arrays in comparison with 6T SRAM cells. These 8T SRAM cells are

able to meet the target voltage in this work for the aforementioned cache structures without failing. An example of an FDM entry for a cache with associativity of 4 is represented in Figure 2.2.

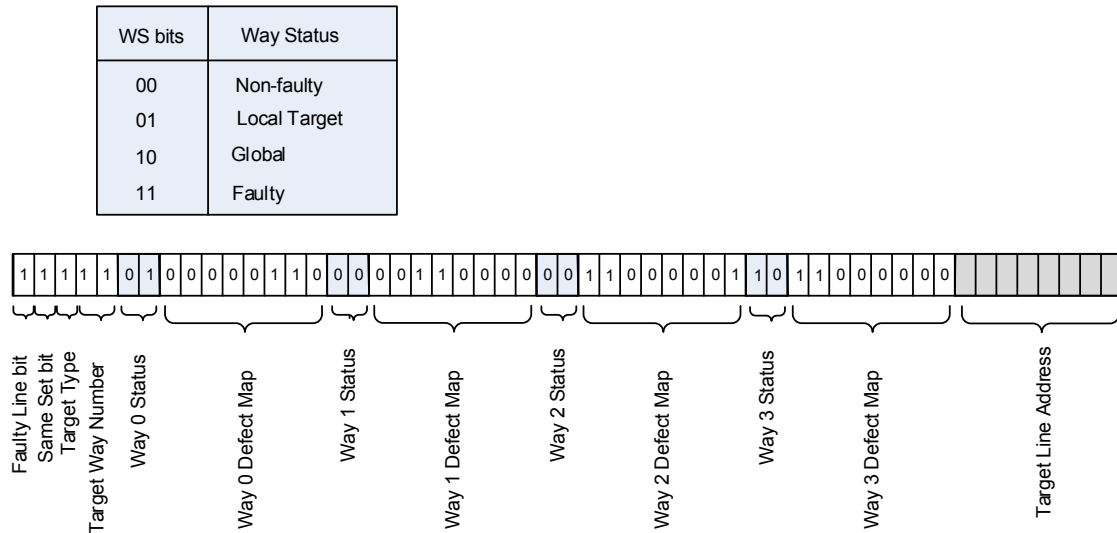


Figure 2.2: Details of a row of FDM.

Each FDM entry includes multiple configuration bits, the defect map of each way (block), status of each block and the address of the Target line. Each bit in the defect map section represents the faulty state of a subblock. Way Status bits represent the status of each way or block in a cache line. The status of each cache block can assume one of the following: 1) Non-Faulty, 2) Faulty, 3) Local Target, and 4) Global. Initially the status of all blocks in the cache is Non-faulty, representing the absence of any faulty subblocks. If a block contains at least one faulty subblock, its status will be Faulty. A block that is used as a target block for other blocks in a line gets the status of Local Target. A block that has a conflict with other blocks in same set cannot be used as a local target block. It gets the status of Global and might be used as a Global target block. We define the Max Global Block (MGB) parameter as the maximum number of blocks in each line that can be sacrificed (set as global blocks) for replication to achieve fault tolerance. For example, in an 8-way set associative cache, if we set MGB=4, then in the worst case we can sacrifice (set as global block) up to 4 blocks of each faulty line for replication. If we extend it to the whole cache, in the worst case we

sacrifice at most half of the cache for fault-tolerance. We present the algorithms for FDM initialization and configuration in Algorithms 1 and 2.

---

**Algorithm 1.** FDM initialization algorithm

---

**Input:**

*FDM*: empty defect map array

**Output:**

*FDM*: initialized defect map array

```

1: // Step1: Run BIST and find faulty cache lines at a subblock level and fill defect map sections of each entry;
2: for i = 1 to |FDM| do
3:   FDM[i].Faulty = False;
4:   FDM[i].SameSet = True;
5:   FDM[i].tLineAddr = i;
6:   for j = 1 to nAssoc do
7:     if FDM[i].block[j].nFSblocks > 0 then
8:       FDM[i].WayStatus[j] = Faulty;
9:       FDM[i].nFblocks++;
10:    else
11:      FDM[i].WayStatus[j] = NonFaulty;
12:    end if
13:  end for
14:  if FDM[i].nFblocks > 0 then
15:    FDM[i].Faulty = True;
16:  end if
17:  if FDM[i].nFblocks < MGB then
18:    FDM[i].SameSet = False;
19:  end if
20:  if FDM[i].nConflicts > 1 then
21:    FDM[i].SameSet = False;
22:  else
23:    if FDM[i].nConflictingBlocks <= MGB then
24:      set the status of conflicting blocks as Global Block
25:      group other blocks and set one of them as Target Block
26:    else
27:      FDM[i].SameSet = False
28:    end if
29:  end if
30: end for
31: return FDM;

```

---

After completion of the FDM initialization algorithm, we run the FDM configuration algorithm.

We use graph coloring to optimize the selection algorithm of both remote target block and remote target line, for the Step 2 and Step 3 of the FDM configuration algorithm [85]. Graph coloring is a NP-complete problem and there are many heuristics and transformations available to generate a feasible solution. Accordingly, we use a heuristic graph-coloring algorithm that is a modification of the Saturation Degree Ordering (SDO) algorithm [120].

---

**Algorithm 2.** FDM configuration algorithm

---

**Input:***FDM*: initialized defect map array**Output:***FDM*: configured defect map array

```
1: // Step1: Traverse the faulty rows of FDM and based on the conflicts between faulty blocks      inside
each row, categorize the FDM entries in different groups;
2: for i = 1 to |FDM| do
3:   FDM[i].Faulty = False;
4:   FDM[i].SameSet = True;
5:   FDM[i].tLineAddr = i;
6:   if FDM[i].Faulty then
7:     if FDM[i].nFblocks < MGB then
8:       FDM[i].SetStatus = Min_Faulty;
9:       set the status of faulty blocks to Global Block;
10:    else
11:     if FDM[i].nConflicts == 0 then
12:       FDM[i].SetStatus = No_Conflict;
13:       set the status of one of the faulty blocks to Local Target Block;
14:     else if FDM[i].nConflicts == 1 then
15:       FDM[i].SetStatus = Low_Conflict;
16:       make the status of the block that has conflict with other blocks as Global Block;
17:     else
18:       FDM[i].SetStatus = High_Conflict;
19:     end if
20:   end if
21: end if
22: end for
23: // Step2: Assign the lines in Low_Conflict group
24: For the lines in Low_Conflict group, run the graph coloring algorithm to find a Global      Target Block for
each line.
25: // Step3: Assign the lines in High_Conflict group
26: For the lines in High_Conflict group, run the graph coloring algorithm to find a similar      line from other
bank to make it as the Global Target Line.
31: return FDM;
```

---

We name a graph coloring problem solvable, if for a graph  $G$  we can find an integer  $K \geq 0$  such that the nodes of  $G$  can be colored with  $K$  colors while no edge exists between the same colored nodes. We construct a graph based on the conflicts between lines of different entries in the FDM. Each node in this graph represents either a Global Block (Block Node) or a faulty entry (representing an entire line) from low conflict or high conflict groups (Line Node) in the FDM. The edges represent a conflict between a pair of blocks or between a block and a line. For example, two nodes connected by an edge represent two blocks, two lines, or one block and a line that have conflict. Of course a pair of nodes connected by an edge represents a conflict, thus it is not possible to set one of these nodes as a Remote target block for the other one.

We modify the above graph coloring algorithm based on the following constraints: 1) We force the algorithm to color nodes from at least two different banks. 2) We force the algorithm to first pick up block nodes and try to color them with line nodes of other banks. Now, we apply the modified graph coloring algorithm to our graph to find a solution such that neighboring nodes are not assigned the same color. Therefore, after completion of the coloring algorithm, nodes with the same color are guaranteed to have no edges between them; implying that the corresponding cache lines/blocks have no conflicts between them. We set all nodes with the same color in a group and try to set one of them as a Remote Target for other nodes in the group. As a result, if a block node has the same color as one or more set nodes in other banks, we set it as their Remote Target Block. If a line node has the same color as one or more line nodes in other banks, we set all of its blocks as a Remote target block for the blocks in other lines. Since the graph coloring algorithm is run during boot time, the algorithms overhead is amortized within the boot time overhead and is not incurred during run-time. Based on our measurements on a machine with Intel CoreTM2 Duo 2GHz processor and 2GB memory, the overhead varies from 1 ms up to 10 ms, depending on cache configuration and fault rate.

Figure 2.3 shows an example of the FDM configuration for a given distribution of faults in 10 sets of a 2-banked 4-way set associative cache with 4 sub-blocks in each block and MGL=1. As shown in the figure, Set 1 is clean, without any faulty blocks. Set 2 is a member of the min-faulty group while its faulty block is configured as a Remote Target Block for Set 9. Set 7 is another member of this group and its single faulty block is set as a Global Block. Set 3 is an example of a no-conflict group in which the first block (Way 1) is set as a Local Target Block to be sacrificed for fault-tolerance of the other faulty blocks in the set. Set 6 is another member of this group with one of its blocks (Way 4) set as a Local Target Block. Set 4 is a member of the low-conflict group which one of its Blocks (Way 1) is set as a Global Block, since it has a conflict with other blocks. The first block of Set 9 is set as Remote Target Block for this set (Set 4). Set 5 is a member of the high conflict group with two conflicts

between its blocks (Way 0 has conflict with Way 2 and Way 1 with Way 3). All blocks of this set are configured as Remote Target Block for both low-conflict Sets 8 and 10.

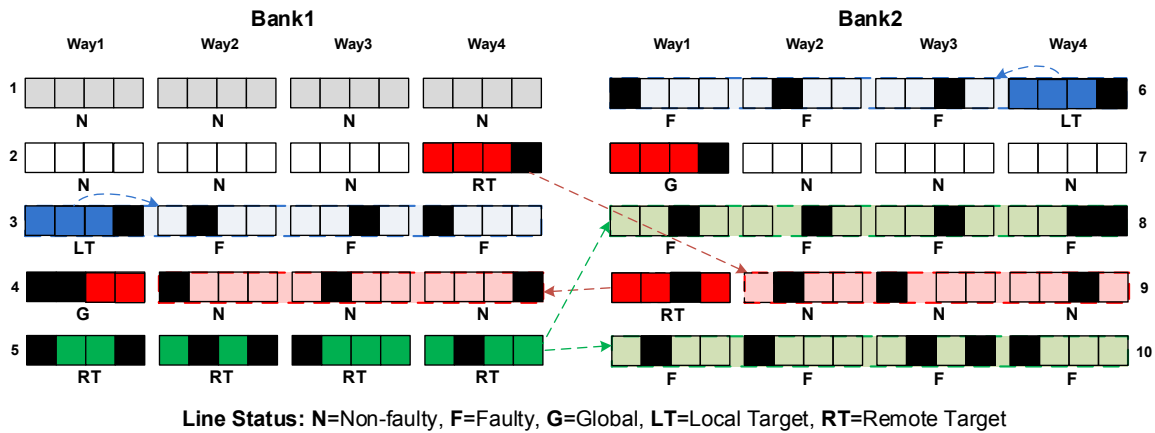


Figure 2.3: An example of FDM configuration and remapping for a given distribution of faults in a 4-way set associative cache.

## Architecture Details

We now present the architecture of the FFT-Cache. We begin with the architecture of a conventional 4-way set associative cache with two banks (labeled CC) in Figure 2.4(a). Based on the tag match results, one of the blocks from Way0 to Way3 from either Bank0 or Bank1 is being selected. The data is transferred to/from cache arrays using multiplexers as indicated in the figure. Figure 2.4(b) shows the architecture of the proposed FFT-Cache. Lets assume the cache is divided into two banks, with each bank containing four ways (blocks) that are further divided into 2 subblocks. The new modules (MUXs and FDM) added to the conventional cache are highlighted in the figure. The additional multiplexer network would allow the cache to compose the final fault-free line from any of the sub-blocks in any of the cache ways in each one of cache banks, based on its FDM data. The FFT-Cache requires two additional levels of multiplexing to compose the final fault-free line, based on either multiple lines within a set or between two or more sets in different banks. The first multiplexing layer (In-Cache MUXing) is added on the cache side to perform the remapping within a set. Note



that this layer replaces the original block-level multiplexer available in set-associative caches with multiple smaller subblock-level multiplexers. The second multiplexing layer (In-Core MUXing) is added on the core side to perform remapping between sets in original (local) and target (remote) banks.

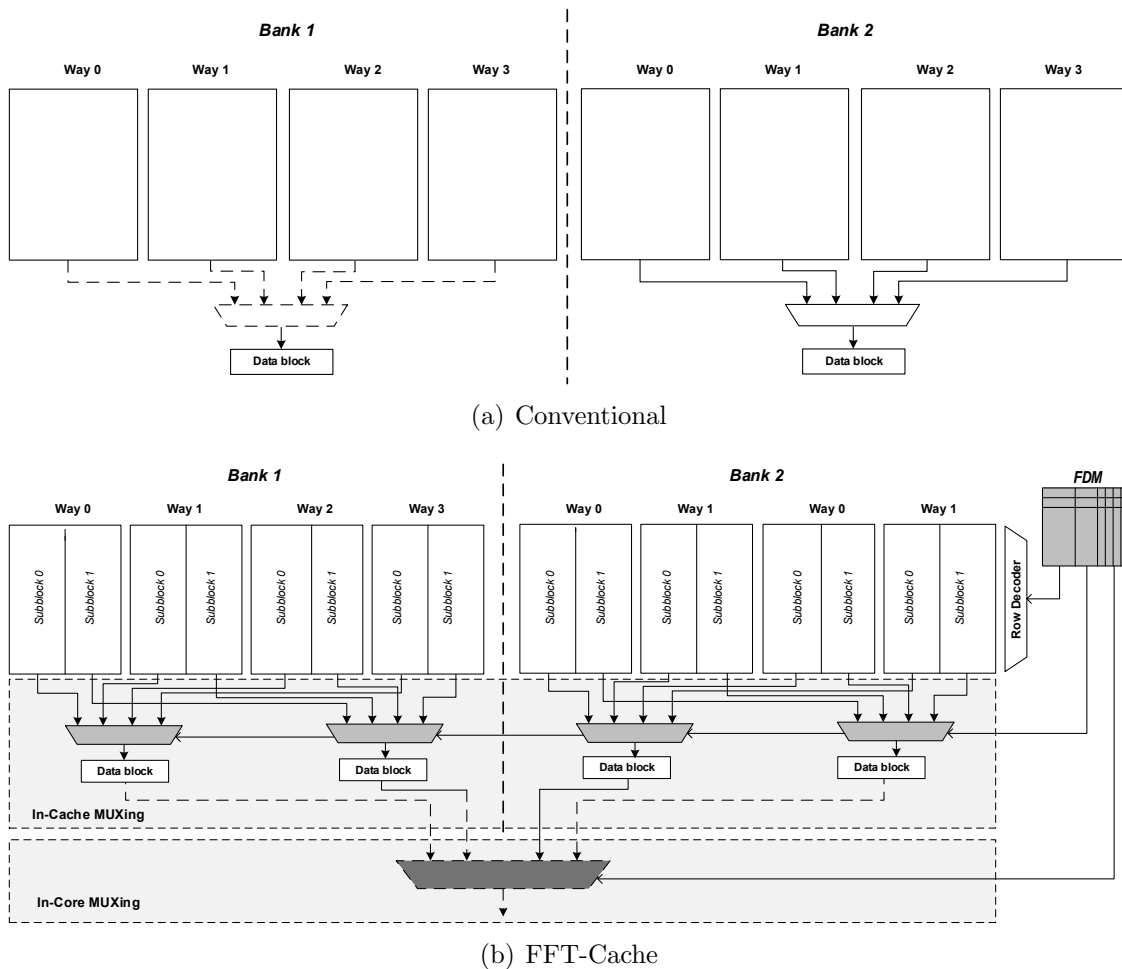


Figure 2.4: Architecture details of (a) a conventional 4-way set associative cache, (b) the proposed FFT-Cache with FDM and 2 sub-blocks per block.

The total number of  $n - to - 1$  subblock-level multiplexers on the cache side is:  $k \times b$  where  $k$  is the number of subblocks in a block,  $n$  is the number of ways (set associativity) and  $b$  is the number of banks. For instance for a 2 bank, 64KB, 4-way set associative cache, with each way with 2 subblocks, a total of 4 4-to-1 multiplexers are required on the cache side. The size of FDM equals to the multiplication of number of cache lines by the FDM entry

size.

## Hardware Implementation

For a quantitative comparison, we synthesized the MUXing layer and output logic for FFT-Cache as well as the multiplexer and output driver of the conventional cache (CC) using Synopsys Design Compiler for TSMC 90nm standard cell library for both L1 and L2 caches. The area and delay of various multiplexers (MUX2, MUX4, MUX32) are used to estimate the overall area/delay overhead of the MUXing network in FFT-Cache and the CC in nominal Vdd. We found that the delay of FFT-Cache output logic increased by only 5% compared to CC output MUX network while area and power consumption are increased by only 2% compared to a CC MUX network. Recall that the FFT-Cache architecture replaces CCs output MUX with FFT-Cache MUXing layer and output logic; thus we expect that the proposed mechanism will only result in a minimal increase of the cache delay (estimated at < 5%).

### 2.3.2 Evaluation

This section evaluates the effectiveness of the FFT-Cache architecture in reducing power consumption of the processor while keeping overheads as low as possible. Before presenting the experimental results, we describe our methodology and experimental setup, and outline the exploration space for our experiments.

#### Simulation Setup

Table 2.1 outlines our experimental setup for the baseline processor configuration. The processor is configured with a 64K 4-way set associative L1 cache. The architecture was

Table 2.1: Processor Configuration

| Parameter        | Value  |
|------------------|--|
| Core             | Alpha, 4-way issue OOO, 96-entry ROB                           |
| Load/store queue | 32   |
| L1 I/D Cache     | 64kB split I/D, 2 banks, 4-way assoc, 64B lines, 2 cycles, LRU |
| L2 Cache         | 2MB, 2 banks, 8-way assoc, 128B lines, 20 cycles, LRU          |
| Memory           | 300 cycles   |
| Technology       | 90nm   |

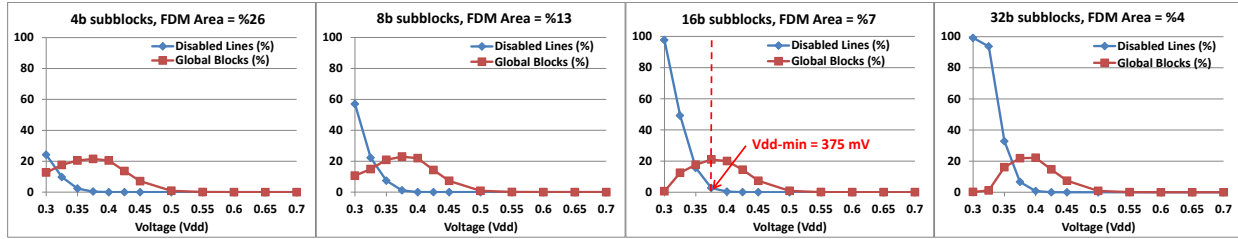
simulated using an extensively modified version of SimpleScalar 4.0 [14] using SPEC2K benchmarks. Benchmarks were compiled with the -O4 flag using the Compaq compiler targeting the Alpha 21264 processor. The benchmarks were fastforwarded for 3 billion instructions, then fully simulated for 4 billion instructions using the reference data sets. We used CACTI6.5 [116] to evaluate area, power and delay of both L1 and L2 caches and their related FDMs. The Synopsys standard industrial tool-chain (with TSMC 90nm technology library) was used to evaluate the overheads of the MUXing layer components (i.e., MUXes, comparators, MUXes selection logic, etc.). The load/store latency of 2 cycles is assumed to be broken into actual cache access taking place in cycle 1, while the bus access takes only a part of the next cycle. From our discussion in Section 2.3.1, the cache delay is increased only slightly ( $< 5\%$ ) for nominal Vdd. However, considering the increase in logic delay due to Vdd scaling, we conservatively assume that in the worst case, the latency of the cache would be increased by one full cycle for L1 and two cycles for L2. These extra cycles are considered based on our timing analysis for both muxing layer and FDM access which are on the critical path of the cache access.

## Design Space Exploration

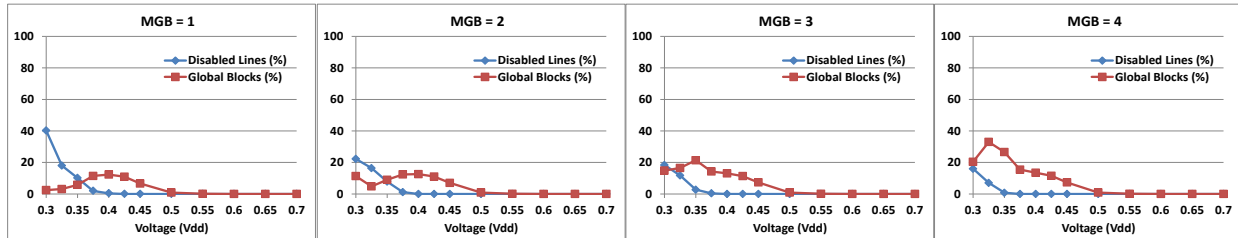
We study the impact of various FFT-Cache configuration parameters including subblock size and MGB on the number of target lines/blocks (non-functional cache part). In addition, we

study the impact of cache associativity on FFT-Cache functionality. Note that since MGB is decided by cache associativity (i.e., half of cache blocks in a line can be a global block candidate), it makes a lot of sense to study the impact of cache associativity on the size of non-functional cache part. For a given set of cache parameters (e.g., associativity, subblock size, MGB, etc.), a Monte Carlo simulation with 1000 iterations is performed using our FDM configuration algorithm described in Section 2.3.1 to identify the portion of the cache that should be disabled while achieving a 99% yield. In other words, probability of failure of the cache must be below 0.001 when operating in low-power mode.

Figure 2.5 and Figure 2.6 present the design space exploration of FFT-Cache for L1 and L2 caches, respectively. We present the results for different associativity (4 and 8) and various subblock sizes (4, 8, 16 and 32) in these figures. As evident in Figures 8 and 9, decreasing Vdd increases the size of cache non-functional part. It is notable that for very low voltage (below 400mv), the number of global blocks decreases. Overall, the effective size of cache (cache size - non-functional part) decreases as the voltage is lowered. We can also observe from the figure that decreasing the size of subblocks, increases the area overhead of the FDM. Decreasing the size of subblocks also reduces the size of cache non-functional part. Increasing MGB, increases the number of non-functional blocks only slightly while it significantly reduces the number of non-functional lines. In fact increasing the MGB helps the FDM configuration algorithm to find a global target block rather than sacrificing a target line. During the process of finding the FFT-Cache minimum achievable operating voltage (Vdd-min), we limit the size of cache non-functional part and the overhead of the FDM table as described below. The number of target lines/blocks (i.e., the size of non-functional cache part) determines the performance loss of FFT-Cache. To limit the performance loss we assume the relative size of non-functional part to be less than 25% of cache size. To minimize the implementation overhead we assume the FDM size to be less than 10% of cache size. This assumption requires the subblock size to be 16 bits or higher (32 bits and 64 bits). Based on these assumptions we find the minimum achievable operating voltage. This



(a) Percentage of disabled lines and global blocks for 4-way L1 (Max global block = 1) while varying Vdd and subblock size



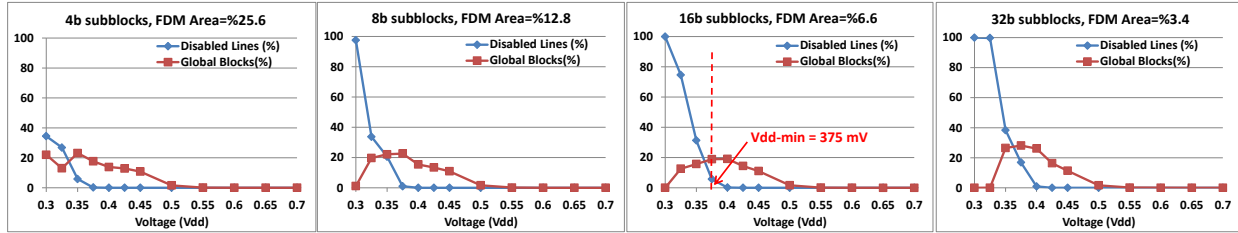
(b) Percentage of disabled lines and global blocks for 8-way L1 (4b subblocks) while varying Vdd and Max global block

Figure 2.5: The effect of changing one design parameter of L1 FFT-Cache while fixing other parameters for different Vdd values.

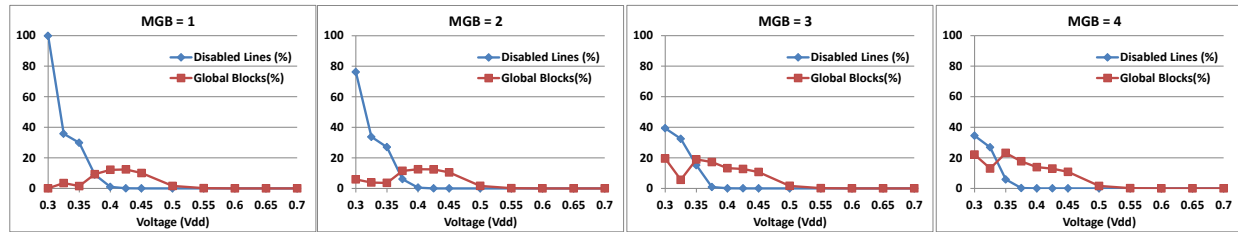
is shown in Fig. 8 and Fig. 9. Using FFT-Cache scheme the minimum operating voltage for L1 and L2 caches is 375mv.

## Performance Overhead

In Figure 2.7 we report the impact of FFT-Cache on processor performance when running SPEC2K benchmarks. The relative performance degradation in terms of IPC (instruction per cycle) is reported for a 64KB, 4-way set associative L1 FFT-Cache (Fig. 14.a) and 2MB, 8-way set associative L2 FFT-Cache (Fig. 14.b). We also report the performance degradation when the FFT-Cache scheme is deployed simultaneously in L1 and L2 (Fig. 14.c). Furthermore, we report the breakdown of performance drop, either due to increasing in cache access delay (from 2 to 3 cycles for L1 and 20 to 22 cycles for L2) or reduction in cache effective size. The results are acquired for the minimum voltage configuration (MGB=1, subblock size=16 for L1 and MGB=4, subblock size=16 for L2). On average, performance drops by 2.2% for L1 cache and 1% for L2 cache. For L1 cache the additional



(a) Percentage of disabled lines and global blocks for 8-way L2 (Max global block = 4) while varying Vdd and subblock size



(b) Percentage of disabled lines and global blocks for 8-way L2 (4b subblocks) while varying Vdd and Max global block

Figure 2.6: The effect of changing one design parameter of L2 FFT-Cache while fixing other parameters for different Vdd values.

delay of accessing the cache is responsible for the majority of performance loss. The impact of additional delay on performance is lower for L2 cache mainly due to the large L2 cache access delay (2 cycles delay overhead compared to 20 cycles L2 cache access delay). The results also indicate that the performance degradation for both L1 and L2 varies significantly across different benchmarks. The highest performance loss is observed in bzip2 and gzip benchmarks (more than 5% IPC loss). In fact these are high IPC benchmarks. In these benchmarks 1 cycle additional delay of L1 cache access in addition to reduction of L1 cache effective size by 20% is not tolerated. In many benchmark almost no performance loss is reported. These benchmarks include facerec, galgel and lucas. Our investigation indicated that while in these benchmarks the cache miss rate increased slightly due to cache effective size reduction, the nature of out-of-order execution helped the benchmark to maintain the performance. For L2 cache the performance degradation is lower. The largest performance drop is 8% and is for ammp benchmark. Finally a 3% performance loss is observed when FFT-Cache scheme is deployed in both L1 and L2 caches.

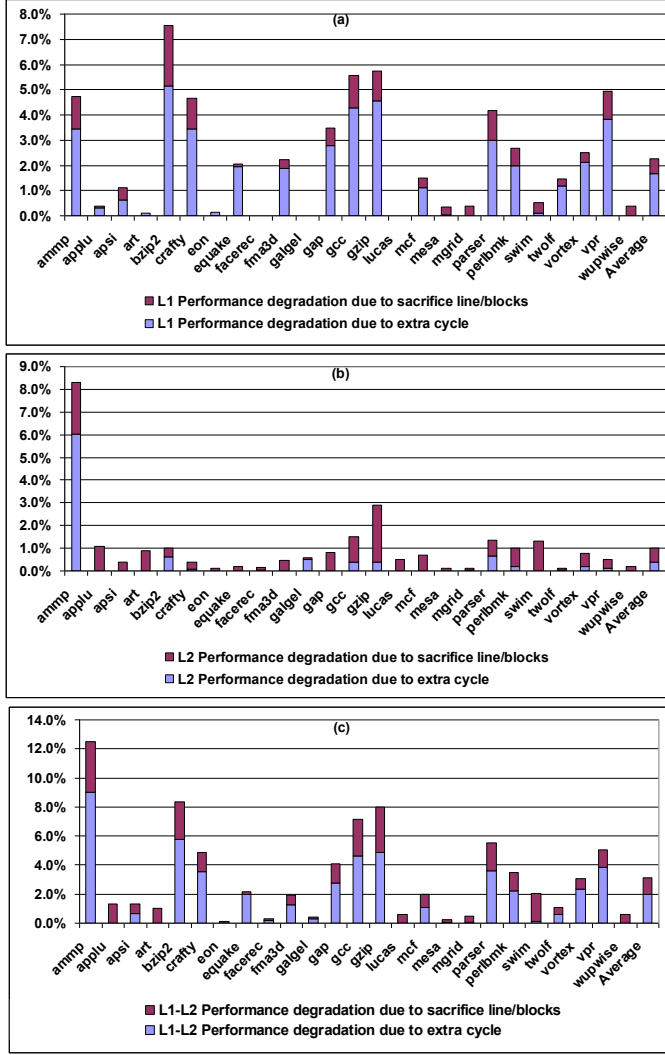


Figure 2.7: Processor performance degradation when applying FFT-Cache for (a) L1 (b) L2 and (c) L1 and L2 at the same time.

## Power and Area Overhead

Figure 2.8 summarizes the overhead of our scheme for both L1 and L2 caches. The power overhead in this figure is for high-power mode (nominal Vdd). We account for the overheads of using 8T SRAM cells [152] for protecting the tag and defect map arrays in low-power mode. To reduce the effect of leakage and dynamic power consumption of FDM in high-power mode, we assume clock gating and power gating is applied in the FDM array. Therefore, the main source of dynamic power in nominal Vdd relates to bypass MUXs. As it showed in this figure

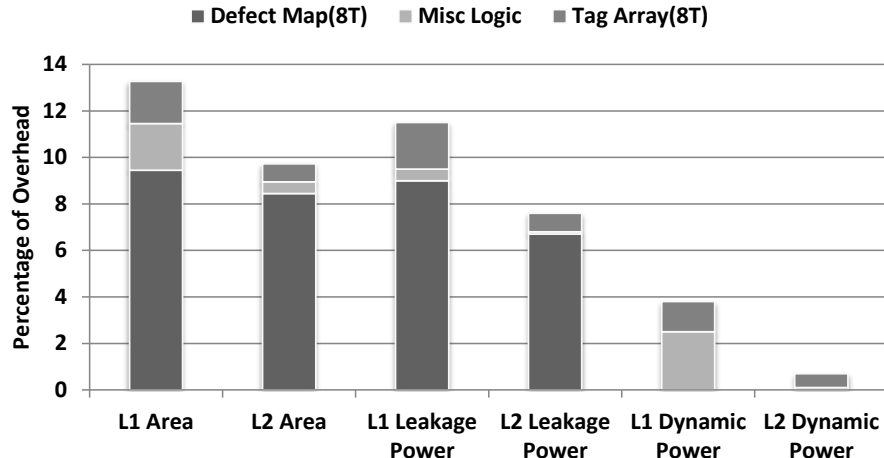


Figure 2.8: Power and Area overheads of FFT-Cache.

it is less than 3% for L1, but is trivial for L2. As evident in Fig. 15, the defect map area is a major component of area overhead for both L1 and L2. The total area overhead for L1 is 13% and for L2 is less than 10%. Also, the defect map is the major component of leakage power in both L1 and L2. Based on our CACTI results in both nominal Vdd (660 mV) and Vdd-min (375 mV), we achieve 66% dynamic and 48% leakage power reduction in L1 cache and 80% dynamic and 42% leakage power reduction in L2 cache.

## Comparison to State-of-the-art Methods

In order to illustrate the effectiveness of FFT-Cache, first we compare the efficacy of our methodology in fault re-mapping against multiple state-of-the-art remapping-based fault-tolerant techniques proposed in recent years. Then, we perform a quantitative comparison of FFT-Cache to four well-known alternative cache protection schemes.

1. *Comparison to Recent Remapping Techniques:* In this section we compare FFT-Cache against three state-of-the-art remapping techniques include RDC-Cache [138], Salvage Cache [86], and Ansaris work [10]. Figure 2.9 shows the effective cache size for each of these techniques across various voltage points. We use the failure rates reported in [137]



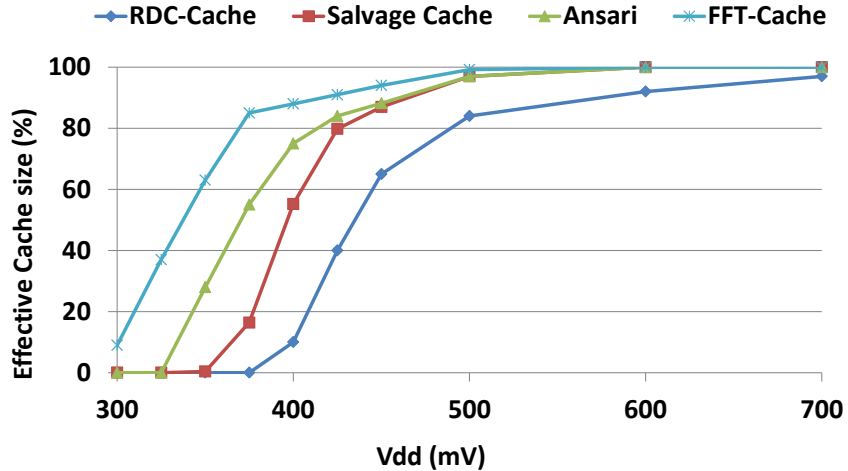


Figure 2.9: Comparison of effective cache size for different fault-tolerant techniques.

for 65nm technology node. The results reported in this figure are based on a 1000-run Monte Carlo simulation for a 8MB 8-way set associative LLC cache with two banks and 64 bytes block size. To have a fair comparison, for all these techniques including FFT-Cache, we assume fault map area overhead is at most 9% of the cache size (the area overhead of FFT-cache with 16-bit sub-blocks). Based on this assumption and to meet the overhead limit, the Ansaris method selects 16-bit sub-block size. The sub-block size for RDC-cache and Salvage-cache are 128-bit and 64-bit, respectively. The results show that in ultra-low voltage region (below 400mv) as the failure rate increases, FFT-Cache results in much smaller cache capacity loss.

2. *Quantitative Comparison to Alternative Methods:* In this section we compare our scheme to four state-of-the-art works including ZerehCache [8], Wilkerson et al. [156], 10T SRAM cell [38], and Ansari et al. [10]. This is just a quantitative comparison to compare the overheads of different schemes while has a rough comparison of their min-Vdd. We obtained these overhead results either directly from the related papers (when a direct comparison was possible), or by trying to recalibrate the results to enable a fair comparison for the same technology node. We calculated the Vdd-min for ZerehCache and updated the Vdd-min of Wilkersons scheme using our cache configuration and fault model in 90

Table 2.2: Comparison of different cache protection schemes

| Scheme          | Vdd-min<br>(mV) | L1 Cache             |                       | L2 Cache             |                      | Norm.<br>IPC |
|-----------------|-----------------|----------------------|-----------------------|----------------------|----------------------|--------------|
|                 |                 | Area<br>over.<br>(%) | Power<br>over.<br>(%) | Area<br>over.<br>(%) | Area<br>over.<br>(%) |              |
| 6T cell         | 660             | 0                    | 0                     | 0                    | 0                    | 1            |
| ZerehCache[8]   | 430             | 16                   | 15                    | 8                    | 12                   | 0.97         |
| Wilkerson [156] | 420             | 15                   | 60                    | 8                    | 20                   | 0.89         |
| Ansari [10]     | 420             | 14                   | 19                    | 5                    | 4                    | 0.95         |
| 10T cell [38]   | 380             | 66                   | 24                    | 66                   | 24                   | 1.0          |
| FFT-Cache       | 375             | 13                   | 16                    | 10                   | 8                    | 0.95         |

nm. All other overheads are obtained directly from the papers. Note that, except 10T cell and Wilkersons work that use 65 nm technology, all other schemes including FFT-Cache use 90 nm. A 380mVdd- min for the 10T cell was derived in 65nm and it is clear that in 90nm this value would be higher. Table 2.2 summarizes this comparison based on the minimum achievable Vdd, area and power overheads for both L1 and L2 caches, and normalized IPC. In this table, different techniques are sorted based on the minimum achievable Vdd-min, when targeting 99.9% yield.

Overall, our proposed FFT-Cache achieves the lowest operating voltage (375mv) and the highest power reduction compared to all other techniques. The closest techniques to ours are 10T cell, Ansari, and Wilkerson. 10T cell achieves almost similar power reduction to FFT-Cache but incurs a 66% area overhead. Wilkersons work has a significant power overhead and also it suffers an 11% performance degradation. The Ansari technique incurs slightly lower power and area overhead just for L1 cache compared to FFT-Cache but it does not reduce operating voltage below 420mw and thus achieves lower power reduction. Overall, the flexible defect map of FFT-Cache along with high configurability and high flexibility allow it to tolerate higher failure rates compared to other similar techniques.

## 2.4 EEC: Embedded Erasure Coding

### 2.4.1 Preliminaries

This section provides a general background on MBUs, introduces erasure coding, and reviews related work on ECC schemes.

#### Multiple Bit Upsets

A realistic quantification of the proposed MBU correction technique requires detailed information on possible MBU patterns and their occurrence rate over the SRAM array. In order to achieve this, a 3D-TCAD simulation is performed using a commercial soft error evaluation tool (similar to the one presented in [55]). In this experiment, the neutron-induced MBU patterns and their occurrence probabilities are obtained for a 45 nm SRAM memory working in a terrestrial environment. The distribution of MBU sizes is demonstrated in Figure 2.10. As shown in this figure, almost half of the soft errors in the employed 45 nm SRAM memory are MBUs. It also depicts particular MBU patterns with high occurrence probabilities. The experiment shows that the largest MBU pattern affects 24 bits. Moreover, the ratio of MBUs as well as the largest MBU size will further increase in smaller technology nodes [76]. Hence, depending on the employed technology in SRAM memories, an effective coding technique with appropriate correction capabilities is required to be able to detect and correct the largest possible MBU incidents in the target memory.

#### Erasure Coding

At system level, erasure coding [128] has been very successful in disk error protection [125], error correction in FPGA configuration frames [130] as well as providing reliable data storage

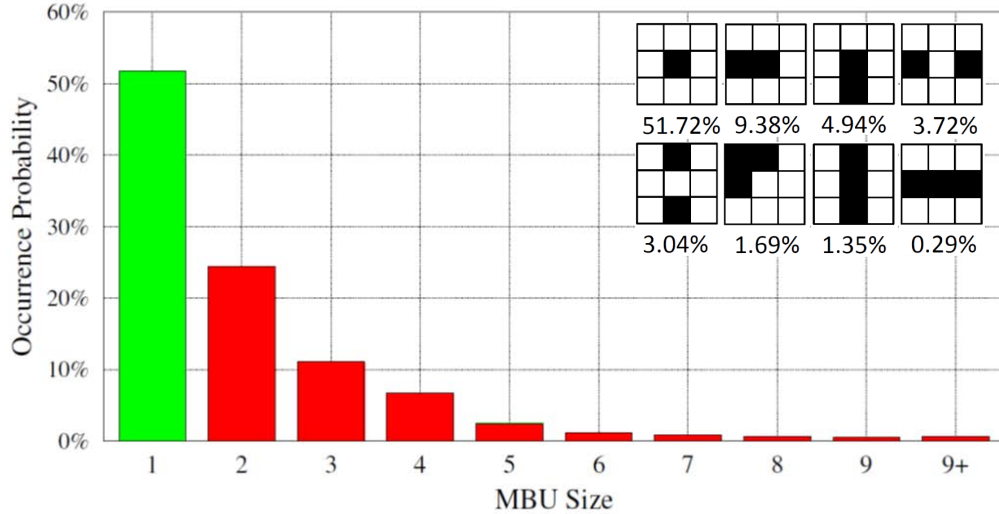


Figure 2.10: MBU size distribution and particular patterns with high occurrence probabilities in a 45 nm SRAM Array

for the storage/distributed systems domain [66, 74]. A variety of erasure coding techniques with different properties such as recovery coverage, encoding/decoding complexity, and area overhead are proposed in the literature [128]. In general, an optimal erasure code is a coding technique which transforms  $m$  blocks into  $m + n$  blocks by adding  $n$  redundant blocks. This transformation or coding is done in a way that in case of erasure (loss of data), the original blocks can be recovered from each possible set of arbitrary  $m$  blocks among the  $m + n$  coded blocks (see Figure 2.11). The recovery coverage of an erasure code is specified by the number of erasures that could be tolerated. For an optimal erasure code, the recovery coverage is equal to the number of redundant blocks, called erasure blocks (i.e.  $n$ ). The area overhead of an erasure code is defined as the ratio of redundant blocks to original data blocks which equals to  $n/m$  for the optimal erasure codes.

The erasure codes are designed to recover the original blocks when a subset of blocks is not available (i.e. erased). Although this coding approach is not designed to detect or correct errors, once an error is localized in particular blocks using a detection technique, the original blocks could be reconstructed using an erasure code.

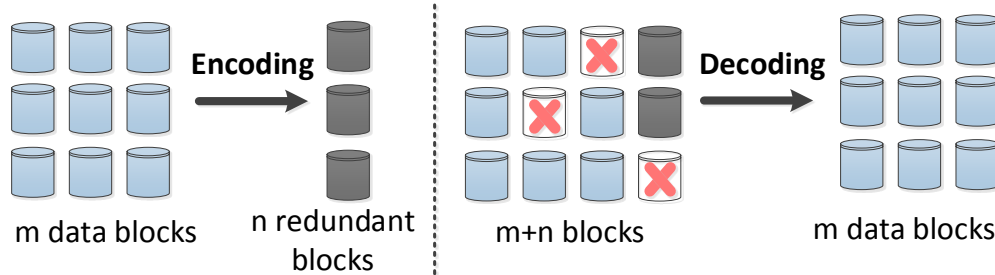


Figure 2.11: Basic erasure coding technique.

In contrast with all previous approaches, we first perform a detailed *technology dependent analysis* to extract MBU patterns and their respective probabilities. Then, based on our observation we try to minimize the overheads of MBU protection by optimizing the interleaving distance of parity checking and making the error recovery process less complicated and faster compared to similar approaches to reduce system-level costs in case of high error rates.

## 2.4.2 Proposed EEC

To enable correcting a larger ratio of MBUs with affordable complexity and latency overheads, we propose the EEC approach. The main idea behind this work is to use the concept of erasure coding for error correction by reconstructing the contents of the erroneous cache blocks within each cache set. The light-weight interleaved per-word parity codes are applied for error detection. For error correction, we adapt the notion of traditional erasure coding to define a technique that performs coding across multiple cache ways in a set-associative cache. It reserves a cache way to store erasure check bits for the remaining ways. In this technique, we trade off cache capacity for a low-cost MBU correction with fast error recovery without any significant hardware modification and overhead. Moreover, the proposed technique is implemented in a reconfigurable way that depending on the desired level of performance/reliability, the reserved way in different parts (banks) of the cache can be considered as either erasure storage or normal data storage. In contrast to similar schemes

[82, 106, 84], our proposed scheme does not require any significant change to the existing cache architecture of processors to add redundant rows.

## Parity-based Erasure Coding

There are a variety of erasure codes with different attributes in the literature. In the context of on-chip memories, an appropriate erasure code should be selected that meets the memory constraints. Since the soft error occurrence rate is relatively small and read/write operations in cache arrays are continuously performed during the runtime, it is less likely to have more than one erroneous cache block per cache row at the same time [94]. On the other hand, the encoding time of the erasure code is critical in our design as it affects the write delay. Also a high decoding latency delays the error recovery process. Thus, depending on the granularity of error coding and the number of erasure blocks, there are multiple trade-offs between area, reliability, and performance.

In this work, we consider a parity-based erasure code [128] with only one redundant (parity) block. This coding technique satisfies our requirements as it has short decoding and encoding latencies. The parity based erasure code is an optimal erasure code which has only one redundant block ( $n = 1$ ;  $m + 1$  erasure code). This technique can recover the erased (erroneous) block from all cases with only one erasure. Figure 2.12 represents a sample erasure code with  $m = 4$ . As depicted in this figure, the redundant block contains the parity of data stored in all other blocks. In case of an erasure, the redundant block can be used as the parity of other  $m$  blocks to recover the contents of the erased block. Note that the factor  $m$  determines the trade-off between the encoding/decoding time and the area overhead. In this work, for a  $k$ -way set-associative cache,  $m$  is equal to  $k - 1$ .

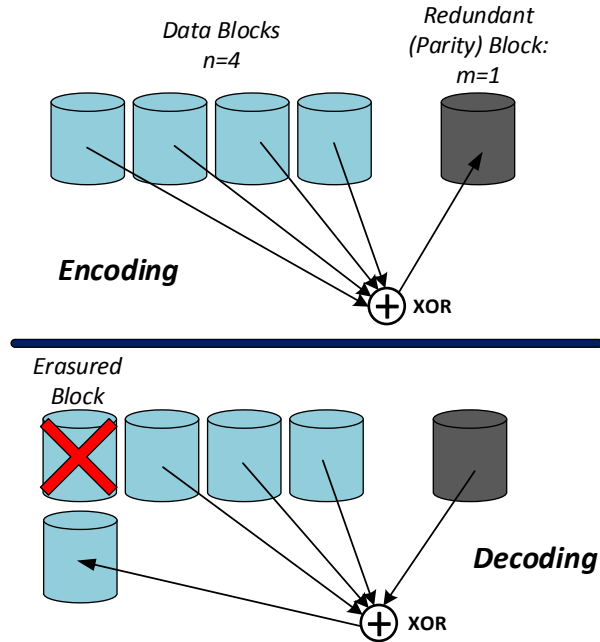


Figure 2.12: A sample parity-based erasure code with one redundant block.

### Parity-based MBU detection

We employ a light-weight bit-interleaved parity code for error detection. We consider  $n$ -way interleaved parity bits per word which is computed by XORing the bits whose distance is  $n$  (e.g., for  $n=8$   $\text{Parity\_bit}[i] = \text{data\_bit}[i] \oplus \text{data\_bit}[i+8] \oplus \text{data\_bit}[i+16] \oplus \dots$ ). In this way, we can detect up to  $n$ -bit contiguous errors (MBUs) in each cache word. The optimal value of  $n$  will be extracted based on the technology dependent MBU patterns (see Section 2.4.1).

### EEC Operation

Figure 4.17(b) depicts the architecture of a 4-way set-associative L1 cache equipped with EEC and the process for updating the erasure codes on writes. It keeps an 8-way interleaved parity per block using 8 parity bits per word. It also reserves the last way of the data array for storage of the erasure codes. Note that the latency and area overhead of this parity coding is similar to byte-parity coding (common in timing-critical L1 caches) as well as SECDED

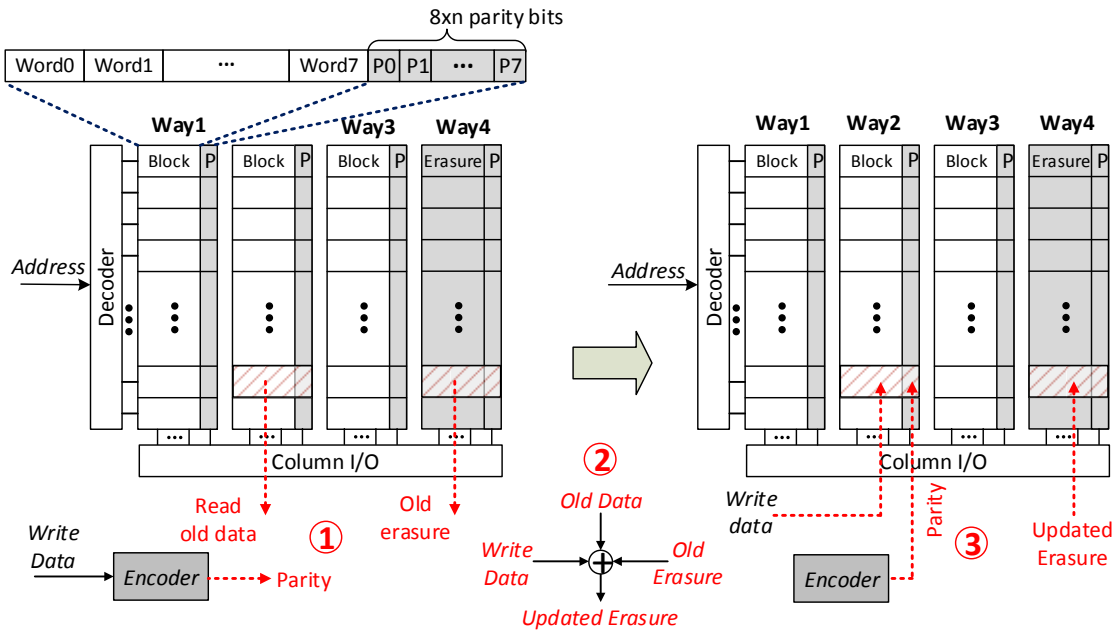


Figure 2.13: A 4-way L1 cache architecture protected with EEC that shows the updated write operation in three steps.

coding [82].

During normal operation (no error is detected), every cache read checks the parity bits associated with the accessed word. On every write, parity and erasure codes are updated. As depicted in Figure 4.17(b), the update of the erasure codes is a three-step process that requires first reading the old data and erasure code, XORing them with the new data, and then writing both the new data and updated erasure code back to the cache. Thus, every write operation is converted to a "read-before-write" operation to read the old data before writing the new data. These three steps are pipelined and the update of the erasure code is done either along with cache data write or after that depending on the cache architecture. Depending on the architecture of the cache (i.e. sequential or parallel) and the number of cache ports, this operation can be done as fast as a single cache access and as slow as four cache accesses.

As shown in Figure 4.17(b), the L1 cache has a fast and parallel access, so both data and



erasure code can be accessed in parallel. But in case of caches with a sequential architecture, we need to first read the old data, then read the erasure data during the next access. In this case, the XOR operation (step 2 in the figure) is done in two steps and an additional register is required to keep the intermediate value. Finally, it writes back the new data and updated erasure code consequently. So to have the minimum impact on performance, we apply this approach to fast L1 caches with parallel access and/or lower level caches with sequential access.

In case the processor has two separate read and write ports, those read and write operations can be performed in parallel. However, this may cause some read port contention in out-of-order processors that leads to an increased load latency. This problem can be addressed to improve the performance penalty by using different techniques such as cache port stealing [93] or cycle-stealing [106]. For in-order processors with a single port, the extra read increases the cache write latency. However, there exist some approaches such as port stealing [93] or thread scheduling [88] to hide the extra access latency for the L1 cache.

## **Error Recovery**

Once an error is detected by checking the parity bits, the cache controller initiates the recovery process. The controller reads all data blocks in the set of the accessed block including the erasure block. Then, it XORs the value of all blocks except the block that contains the erroneous word. The result of the XOR operation is the correct value of the faulty block, which is then written back to the accessed block. Note that the recovery operation requires accessing every cache block in the set. In fast L1 caches, since all blocks of the set are accessed in parallel, the recovery is very fast. However, L2 caches usually have a sequential architecture in which tag checking is done first and then only one block is accessed at a time. In this case, the cache controller needs to access the blocks of each set one by one which increases the latency of error recovery up to the latency of accessing all the blocks in the

set. During each cache write-back, the controller also checks the dirty bit of the block. If the evicted block is clean, it does not call the recovery process.

## **Reconfigurability**

In contrast to conventional erasure coding approaches, EEC does not need to add extra resources as redundant (erasure) blocks (mentioned in Section 2.4.1). Instead, by changing the configuration of the cache, EEC can reserve a part of the cache (i.e one way) as redundancy. The primary benefit of the EEC reconfigurability is that cache capacity can be easily reduced to suit the required erasure blocks, while only minor modifications to the cache controller and/or software layers are needed. So reliability can be improved with small performance impact. Hence, we add an extra bit to the tag bits of each row which indicates whether that row reserved a block for erasure codes. On the other hand, if the reliability requirements are low, the reconfigurability allows the processor to use all cache ways for the normal operation, and by that means to minimize the performance degradation.

### **2.4.3 Evaluation**

In this section we present different sets of results to demonstrate the evaluation of three major contributions of our approach – simplicity, configurability, and lower error recovery time – mentioned in previous Section, compared to similar approaches.

#### **RTL Implementation**

We implemented the EEC approach by modifying the HDL source code of the Leon3 processor [60]. Leon3 is a simple embedded in-order processor that has only L1 data and instruction caches with the same read and write latency of 1 cycle. We modified its data

cache controller and cache array modules to configure the last way for erasure code storage. We also modified the cache controller to implement the "read-before-write" operation. Since the Leon3 L1 cache has only one data port, we needed to modify the cache controller to perform an extra read before each write operation. Also, since Leon3 has a single cycle load/store operation, we considered one extra cycle to update the erasure code and write it back. For error recovery, since Leon3 has a fast cache, all the blocks are accessed in parallel. Hence, it takes only two cycles to recover the erroneous block. Table 4.5 shows the details of the simulated processor. We simulated the modified Leon3 processor by running multiple MiBench benchmarks including *stringsearch*, *basicmath*, *bitcount*, and *qsort*. The execution time results showed no more than 1% performance overhead. We also synthesized it using the *Nandgate* 45 nm technology. The synthesis results showed less than 5.8% logic area overhead and an increased latency of 0.4%.

### Cycle-accurate Simulation

We used the Gem5 cycle-accurate simulator [34] to model EEC for a single-core high-performance processor. We added EEC to both the L1 data and L2 cache of the Alpha processor which is an out-of-order processor running at 3 GHz. In this processor, for both cache levels, read and write latency are independent and can use different values. Thus, if our proposed EEC technique is employed, the read access latency remains the same as the normal value, while the write latency is doubled (due to the additional read operation for every write access). In case of L1, we considered a fast cache in which both tag check and data access are performed in parallel. So all data blocks including erasure check bits are read in parallel and the error recovery latency is equal to two cache accesses. In contrast, the L2 cache has a sequential architecture, so the tag checking and data access are done separately. In this case, the cache blocks inside a set can be accessed one by one. So the write latency is equal to the latency of two writes and two reads. However, since the rate

Table 2.3: Simulated System Parameters

| Parameter        | Processor   |  |
|------------------|---|--|
| Core             | Out-of-order  | Leon3 (in-order)                               |
| ISA              | ALPHA   | SPARC  |
| Functional units | 2 integer ALUs, 2 FP ALUs                                 | 1 integer ALU, 1 FP ALU                        |
| Issue width      | 4 instructions / cycle                                    | 1 instructions / cycle                         |
| L1 I/D Cache     | 32kB split I/D, 4-way assoc, 64B blocks, 3 cycles, 1-port | 4KB I\$, 16KB D\$, 32B blocks, 1 cycle, 1-port |
| L2 Cache         | 512kB unified, 8-way assoc, 64B blocks, 7 cycles, 1 port  | no L2 cache                                    |
| Clock Frequency  | 3 GHz   | 800 MHz  |
| Technology       | 45nm  |  |

of write to L2 is much lower than L1, the extra L2 write latency has a negligible impact on performance and dynamic energy overheads. Also, the duration of error recovery is longer and is equal to the latency of one L2 cache access times the degree of associativity. However, it is still much faster than similar approaches which require access to multiple cache rows. We added a block-size register to L2 to keep the intermediate data due to XOR of different cache blocks.

We selected six benchmarks from SPEC2000 suite to evaluate the proposed EEC approach on the Alpha processor. The benchmarks are fast-forwarded for one billion instructions as warm-up, then simulated for five billion instructions. Table 4.5 shows the details of the simulation systems.

## Performance and Energy Analysis

Since our proposed technique not only reduces the number of available cache ways by one, but also increases the cache write latency, it affects the processor performance, as shown in Figure 2.14. However, as only the latencies for the write accesses are affected and not those

for read operations, the overall performance impact of our proposed technique is very small. This is due to the fact that the number of writes are much lower than the reads and not in the critical path of the application. Also, EEC trades cache capacity and associativity which mainly affects the miss rate in highly memory-intensive applications. On average, over all benchmarks the performance is only reduced by less than 2% for the L1-Cache and less than 3% for the L2-Cache of the Alpha microprocessor simulated by Gem5.

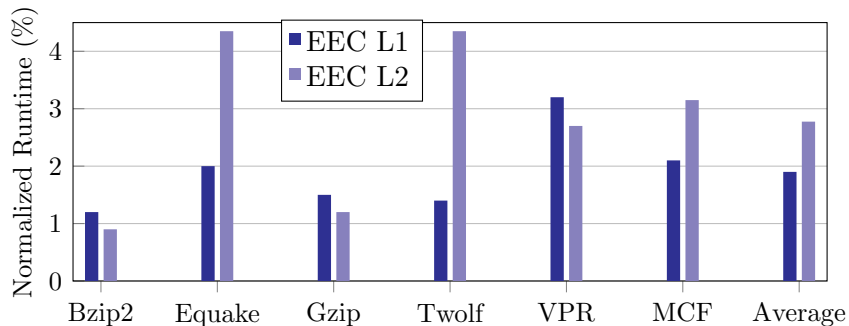


Figure 2.14: Performance overhead of EEC for L1 and L2 caches on Alpha processor

Based on these results, we also evaluated the energy impact of the EEC technique for the Alpha processor. To this end, we employed CACTI [116] to extract the per-access energy for read and write operations as well as leakage power. We obtained the overall energy consumption of the microprocessor with and without EEC according to the Gem5 statistics about the number of cache accesses and runtime. Here, similar to other approaches, we consider a cache equipped with SECDED as the baseline. Since all the overheads are normalized to the baseline, the major contributors to the power and performance overheads are due to the extra read access for L1 and read/write for L2. According to this evaluation, the dynamic energy consumption increases by 3.9% and 1.7% for applying ECC on L1 and L2 caches, respectively. In both cases, the reason for the increase in energy consumption is the fact that EEC adds a read operation for each write access in L1 and two reads and one more write for L2. However, as leakage is dominant in nowadays technology nodes and the ratio of write operations to the total number of cache accesses is just around 20%, the increase in write energy has just a small impact on the overall energy consumption. Here, we do not

consider the leakage power of erasure data storage as explicit overhead, because our scheme does not impose extra resources to the system and the cache ways that used for erasure data directly affect the performance of the system. However, if we consider the reduced capacity cache with reserved ways as the baseline, we may have up to 25 % leakage power overhead for the 4-way L1 and 12.5 % for the 8-way L2.

## Comparison with other schemes

We obtained the MBU correction coverage of different schemes including the proposed EEC, 2D parity cache [82], and SECDED considering the actual MBU patterns of the employed 45 nm SRAM technology. In this regard, we injected one million MBUs in an SRAM array equipped with these coding techniques and then computed the error detection and correction probabilities. When an MBU pattern spans across multiple words, it is considered as detected/corrected if all affected words are detected/corrected by the employed coding technique. Table 2.4 shows the results of this experiment. In this table, 2D parity cache ( $i$ ) means that this technique is equipped with  $i$  horizontal and  $i$  vertical interleaved parity bits. Also, EEC( $j$ ) indicates that  $j$  horizontal interleaved parity bits are used for error detection. As it can be seen in this table, all errors detected by EEC are successfully corrected by this technique, while some of the detected errors by 2D parity cache cannot be corrected as they affect more than one word affiliated with one redundant vertical parity line. On the other hand, the amount of *Detected Unrecoverable Errors (DUE)* in EEC is independent of the number of parity bits and is always zero. However, in case of the 2D parity cache, it depends on the number of parity bits used and can be up to 17 %.

The area overhead of each technique is also compared in terms of the coding area which is defined as the amount of extra code bits that is required by each scheme. As it can be observed from the table, the coding area overhead of EEC is less than 2D coding in all cases and less than SECDED in first four cases. Note that the amount of code storage in EEC

Table 2.4: Reliability and coding area comparison of different techniques

| Technique           | Detection | Correction | DUE    | Coding Area |
|---------------------|-----------|------------|--------|-------------|
| SECDED              | 97.34%    | 65.95%     | 31.39% | 12.5%       |
| 2D Parity Cache (1) | 69.23%    | 52.33%     | 17.10% | 3.22%       |
| 2D Parity Cache (2) | 91.43%    | 78.27%     | 13.16% | 6.44%       |
| 2D Parity Cache (3) | 99.68%    | 97.37%     | 2.31%  | 9.67%       |
| 2D Parity Cache (4) | 99.98%    | 99.37%     | 0.61%  | 12.89%      |
| 2D Parity Cache (5) | 99.99%<   | 99.69%     | 0.3%   | 16.11%      |
| 2D Parity Cache (6) | 100.00%   | 99.98%     | 0.02%  | 19.34%      |
| 2D Parity Cache (7) | 100.00%   | 100.00%    | 0      | 22.56%      |
| EEC (1)             | 69.23%    | 69.23%     | 0      | 3.12%       |
| EEC (2)             | 91.43%    | 91.43%     | 0      | 6.25%       |
| EEC (3)             | 99.68%    | 99.68%     | 0      | 9.37%       |
| EEC (4)             | 99.98%    | 99.98%     | 0      | 12.5%       |
| EEC (5)             | 99.99%<   | 99.99%<    | 0      | 15.62%      |
| EEC (6)             | 100.00%   | 100.00%    | 0      | 18.73%      |

depends on the cache associativity, EEC configuration, and level of protection. For example, in a 4-way cache with fully EEC block protection, the area overhead can be as high as 25%. Since we trade the cache capacity instead of adding any extra elements (spare cache/rows) to prepare the code storage, this is an implicit area overhead with much less cost compared to similar approaches that impose explicit costly overhead to the system.

### Applicability to Permanent Faults

The promising features of EEC such as reconfigurability, short recovery time, and ability to correct sparse errors, give the ability that EEC can also be easily applied for protecting cache memories against permanent faults. Here, we consider the high bit rate faults due to process variation in caches working at near-threshold operation voltages. Using our analytical models and the derived fault model and bit error rates similar to [63], we were able to compare our approach with similar approaches such as 2D error coding. Our analytical results show that while 2D error coding can correct a bit error rate of  $10^{-5}$ , EEC corrects up to  $10^{-3}$  without degrading the yield. Moreover, the EEC error correction is much faster than 2D coding (i.e.,

about ten times) which minimizes the impact on performance and dynamic energy. Due to the lack of space, we consider any further detailed discussion and analysis in this regard as future work.



## 2.5 Summary

In first part of this chapter, we proposed FFT-Cache, a fault-tolerant cache architecture which has a flexible defect map to configure its architecture to achieve significant reduction in power consumption through aggressive voltage scaling, while maintaining high error reliability. FFT-Cache uses a portion of faulty cache blocks as redundancy to tolerate other faulty cache lines and blocks. This is accomplished by using either block-level or line-level replication in the same set or between two or more sets. FFT-Cache has an efficient configuration algorithm that categorizes the cache lines based on degree of conflict between their blocks, to reduce the granularity of redundancy replacement. Using our approach, the operational voltage of the cache is reduced down to 375mV in 90nm technology. This achieves 66% and 80% dynamic power reduction for L1 and L2 caches, respectively. FFT-Cache reduces the leakage power of L1 and L2 caches by 48% and 42%, respectively. This significant power saving comes with a small 5% performance loss and 13% area overhead.

In second part of this chapter, we presented embedded erasure coding to protect cache memories against MBUs at low latency and power overheads. This scheme employs fast bit-interleaved parity coding as error detection in combination with optimal parity-based erasure coding as error correction over set-associative cache memories. This technique reserves a part or the whole of a cache way to store the redundant check bits. Its promising features including simple architecture, reconfigurability, and fast error recovery make it also an appropriate technique for protecting against permanent faults in voltage-scalable caches. We evaluated the proposed approach via architecture-level simulation of a high-performance out-of-order processor (Alpha) as well as RTL implementation of an embedded in-order processor (Leon3). Our experimental results on both L1 and L2 of Alpha processor showed that EEC provides same reliability with faster error recovery compared to similar approaches while incurs less than 3% performance and 4% dynamic energy overheads. Also, the results on L1 cache of Leon3 showed no more than 1% performance and less than 5.8% logic area overhead.

# Chapter 3

## Scalable Fault-tolerant Cache Design in CMPs

### 3.1 Introduction

Moore's Law scaling continues to provide increased transistor counts available to each processor generation. In recent generations, those transistors are primarily devoted to increased core counts. However, the increase in transistors is not accompanied by an increased power budget. Thus, we are approaching an architectural era that some have labeled the dark silicon era [58], because we can configure more transistors than we can afford to turn on and power at once. We prefer to think of this as an era where the tradeoff between area (transistor count) and power is fundamentally changed – transistors become all but free, while power is critically precious. Thus, optimizations that trade area for power are easily justified.

In modern designs, the last level cache (LLC) is typically a shared cache, configured to be large enough to effectively cover the data of all private caches, and store enough data to

protect the off-chip memory and interconnect from the vast majority of on-chip accesses for all cores. As a result, the LLC is typically the largest single consumer of silicon area and one of the largest consumers of power on the processor. This makes it a prime target to apply the new power/area tradeoff. This chapter describes a technique to sacrifice a moderate amount of cache area for dramatic decrease in power.

Aggressive voltage scaling is one of the most effective techniques to reduce processor power consumption. However, aggressive voltage scaling exponentially increases the impact of process variation on SRAM cells, resulting in an exponential increase in the fault rate. This introduces a trade-off between cache yield and minimum achievable voltage  $V_{cc}$ [2]. In the absence of robust error correction strategies (e.g., beyond ECC), we can tolerate few errors and have little opportunity to push down the voltage. In order to improve the cache yield and/or lower the minimum achievable voltage scaling bound, several fault tolerant mechanisms have been suggested [9, 156, 155, 46, 138, 86]. An effective fault hiding mechanism, then, can enable much more aggressive cache voltage scaling. If that mechanism keeps the cache capacity loss due to failures small, we can increase the raw capacity of the cache to keep the effective capacity the same as the original (nominal voltage) cache, enabling a dramatic decrease in power with the primary cost being increased cache area – thus, we fully exploit the new area/power tradeoff.

We show in this chapter that the efficacy of the cache fault masking strategy can have a significant effect on LLC performance and effective capacity, particularly in the case of NUCA LLCs, an inevitable direction as the number of cores and size of LLC grow [58, 68]. However, prior fault protection efforts have focused mostly on private L1/L2 caches in uni-core processors and explore policies that do not scale well to a large multi-bank, non-uniform latency design. In a NUCA design, access latency, network bandwidth, network power, and effective capacity are all sensitive to the exact location of the spare block (redundancy) used for fault-tolerance of a faulty block, relative to the requesting core and the original requested

block.

We present RESCUE, a scalable fault-tolerant design to improve the reliability of a large multi-banked NUCA cache with non-uniform latencies, distributed banks, and multiple access points in a multi-core/many-core architecture. The RESCUE architecture uses a scalable and flexible fault remapping technique to ensure that all cache space not being used for redundant remapping is available for storing useful data.

In RESCUE, each faulty line either serves as a repair line (target line) for one or more other lines, or retains its original mapping and is paired with a repair line to be used to store and reconstruct a complete line. We examine a number of cache line remapping policies that attempt to manage the sometimes conflicting goals of minimizing latency, minimizing network traffic, and maximizing cache hit rate (minimizing lost capacity). It is critical that the remapping policy be sensitive to the NUCA architecture, since access time scales with the maximum distance to each of the source lines, while network traffic scales with the combined distance from the requesting core and the two lines. The effective cache capacity, conversely, is maximized when we place few restrictions on the mapping policy, creating tension between our goals. Overall, however, if we can minimize lost capacity and performance overheads in the presence of frequent failures, we can moderately over-provision the cache (and possibly the interconnect) while still maintaining significant power reduction via aggressive voltage scaling.

This chapter extends REMEDIATE, our scalable fault-tolerant architecture for low-power design of shared NUCA LLC in Tiled CMPs that was first introduced in [24]. While the main goal in REMEDIATE is to develop a remapping-based fault-tolerant scheme for protection of shared LLC in Tile-based CMPs while operating in low voltages, the main goal in RESCUE is to propose a design trend (aggressive voltage scaling + cache over-provisioning) that uses different fault remapping heuristics with salable implementation for shared multi-bank LLC in non-tiled-based CMPs to reduce power while exploring a large design space with multiple

dimensions and performing multiple sensitivity analysis. Moreover, we proposed two new remapping policies optimized for NUCA caches in RESCUE which have various sensitivity analysis based on them. Also, we modified the LLC access flow in RESCUE to remove the access to fault map from the critical path of non-faulty block accesses.

This chapter examines the effectiveness of the proposed design under a wide variety of assumptions, examining different technology targets, multicore design points, cache configurations, and workload makeups. Applying RESCUE for different core-cache configurations over various technology nodes results in 5 to 60% power reduction. Also, by applying RESCUE we can scale the voltage below 400 mV and reclaim more than 55% of lost cache capacity (without RESCUE remapping, all cache space is lost at that voltage level).

## 3.2 Related Work and Motivation

As we discussed in previous chapter, there is a large body of work on fault-tolerant and error-resilient cache design. However, most of these techniques are proposed for single-core processors which rely on two banks and a single fault map that tracks the fault-tolerance information of both banks. Hence, these existing techniques fail to remain efficient at high failure rates in NUCA caches with a large number of banks. Wang et. al [154] proposed a utility-driven address remapping technique at a coarser-grain (bank level) to tackle the capacity loss in NUCA cache of NoC-based CMP architectures.

Unfortunately all these previous efforts are neither scalable nor flexible to be leveraged for protection of NUCA caches with distributed banks in CMP architectures. Since CMP architectures have more than one core that can access one or different distributed banks simultaneously, fault mapping and protection needs to be scalable and distributed. Furthermore, existing fault-tolerant techniques typically use a single fault map and a centralized

fault protection scheme which cannot be scaled for large NUCA CMP architectures: as a result each access needs to go through the fault map first, and no parallel cache accesses are allowed. Another difference for NUCA architectures is that the access latency, failure rate, and criticality of distributed banks are different depending on the location of each cache bank and type of NUCA (SNUCA or DNUCA). Therefore, the fault-tolerant technique needs to be scalable and configurable to address these issues. However, all previous fault-tolerant methods have been designed for at most two banks within a UCA architecture which are not configurable. Moreover, they use a unified remapping technique to protect all faulty blocks without considering differences among different banks.

In contrast to previous efforts, RESCUE leverages a scalable architecture allows efficient remapping across multiple banks in a NUCA CMP. It uses a flexible fault mapping and fault protection scheme which results in higher cache reliability. RESCUE uses a flexible fine-grained remapping technique, using graph coloring to match the repair blocks/lines to a group of original sets. It allows the implementation of multiple heuristics to remap in a non-uniform multi-bank cache architecture, and this flexibility enables the architect to effectively trade off power, performance, and energy efficiency.

## 3.3 RESCUE

### 3.3.1 Baseline Architecture

Before we outline the details of the RESCUE architecture, this section describes our baseline architecture. This is because the technique is tightly connected to the actual cache and interconnect configuration. We study various core and cache configurations in different technologies to understand the limits of aggressive voltage scaling as we face ever-increasing power constraints. Each parameter is chosen to represent current design trends or project future design trends as the technology scales.

Esmailzadeh, et al. [58] study the limits of core scaling and have derived an optimal number of cores that fit in the chip’s area and power budget for below 22nm technologies. We use the outcome of their studies to configure our experiments across technologies (Table 3.1). For the number of cores in each technology generation, we use the results in [58], reporting the number of optimal cores for power/performance trade-off in each technology node. The optimal number of cores we assume in this work and for each technologies is 4, 6, 8, 16, and 32 for 65, 45, 32, 22, and 16nm, respectively. For the technology scaling model we use the ITRS road maps projection [77] and the Predictive Technology Model (PTM) [13] that provides the voltage, area, power, and frequency scaling factors at technology nodes from 65 nm to 16nm.

Our baseline LLC architecture is a tiled, shared, non uniform access cache. We focus on the NUCA design as it is an inevitable design choice in order to reduce the effects of wire delays in future technologies. We assume total LLC capacity scales with the number of cores. To study cache scaling and the appropriate size of LLC for future designs, we consider empirical data from a large set of processors. As shown in figure 3.1, an average projection for the size of cache is 2MB per core. We study the average case as well as an over-provisioned cache

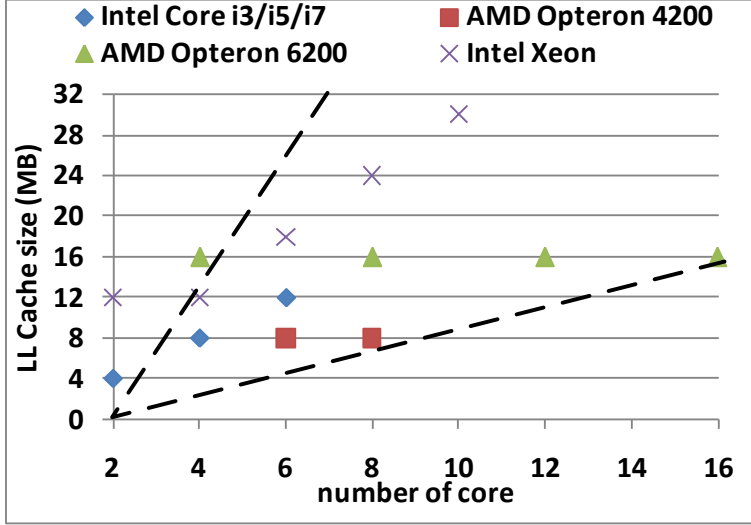


Figure 3.1: Empirical data for the cache size per core across various CMP architecture.

size with 4MB per core.

The baseline design assumes a Non-Uniform Cache Architecture (NUCA) last-level cache. We considered exclusive caches with parallel LLC access. Each LLC bank includes multiple cache lines, within a set (lines with the same index) all mapped to the same bank. As shown in Figure 3.2, the processors and cache banks are organized into a two-dimensional mesh topology. We assume that there are I/O pads (connecting to the off-chip memory) along the two vertical sides. L1 and L2 caches are kept consistent using a directory coherence mechanism.

We consider two NUCA baseline designs, modeling both static (CMP-SNUCA) and dynamic (CMP-DNUCA) policies [30]. The former is derived from Kim, et al.’s SNUCA-2 design [81]. CMP-SNUCA statically partitions the address space across cache banks, which are connected via a 2D mesh interconnect. In static mapping, a fixed hash function uses the lower bits of a block address to select the bank. L2 access latency is proportional to the distance from the issuing L1 cache to the L2 cache bank. By allowing non-uniform hit latencies, static mapping can reduce hit latencies of traditional monolithic cache designs, which fix the latency to the longest path [81]. Because a block can be placed into only one bank, its L2



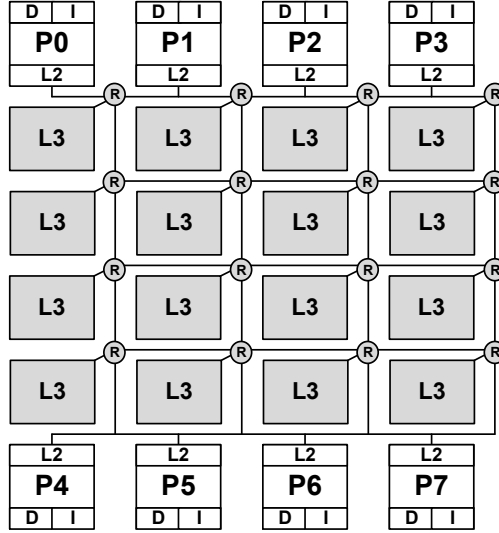


Figure 3.2: Baseline 8 processor CMP structure.

access latency is essentially decided by its address. Dynamic mapping (CMP-DNUCA) [75] addresses the weakness of static mapping by allowing a block to go to one of multiple candidate banks. With proper placement and migration policies, D-NUCA enables the cache to place frequently accessed blocks in the banks closest to the CPU.

Table 3.1: Architectural Specification

| Parameter             | Value  |
|-----------------------|--|
| Integrated Technology | 65nm,45nm,32nm,22nm,16nm   |
| Cores                 | 2,4,6,8,16,32  |
| Issue,Commit width    | 4 (out-of-order)   |
| Functional units      | 4 int/ldst 2 fp  |
| L1 cache              | 32KB, 4-way assoc, 2cyc access   |
| L2 cache (priv)       | 512KB, 4-way assoc, 15cyc access   |
| L3 cache (shared)     | NUCA, 1MB/Bank, 2MB/core, 8-way assoc, 30cyc   |
| L3 miss penalty       | 250 cyc  |
| Memory                | Three 1GB DDR3, 4 128-bit channels<br>2-DIMMs/channel, 2-ranks/DIMM                                  |
| Interconnect          | 2D mesh NoC (NxN) 32-byte links (2 flits/access)<br>1-cycle link latency, 2-cycle router, XY routing |
| Frequency             | 3GHz   |
| Vdd                   | 0.9V-0.3V  |

### 3.3.2 RESCUE Overview

We now present the RESCUE hardware framework for remapping of faulty cache sub-blocks to increase available cache capacity, which extends in several ways prior work on cache block remapping [9, 22, 86, 10]. The prior remapping techniques are mainly proposed for, and evaluated on, a uniform access cache in a single core architecture, and thus do not address several challenges arising from a large multi-bank NUCA LLC.

RESCUE divides each line of the LLC into multiple sub-blocks, which define the granularity at which failures are identified and remapped. A sub-block is labeled faulty if it has at least one faulty bit, as determined by Built-In Self Test (BIST) analysis during boot-up of the system. Thus, a small sub-block minimizes the number of bits lost due to a single fault, but increases the hardware overhead of reconstructing blocks. Instead of maintaining explicit additional cache space to replace faulty lines, we exploit lines already marked as faulty to provide redundant storage to mask faults in other lines. This has a critical advantage. While a static redundancy technique is based on worst-case estimates of faults, our approach sacrifices no more space than is absolutely needed for redundancy. This is particularly useful for a cache that might be used at multiple voltage levels, as we automatically adjust the level of redundancy to the errors manifested at a particular voltage.

RESCUE attempts to sacrifice the minimal number of cache lines and tolerate the maximum amount of defects. This is done by using line-level replication in the same set or among multiple sets. In RESCUE, the information of faulty locations in each LLC bank is kept in a Distributed Fault Map (DFM) which is then used to configure the address mapping. If two lines have faults in the same sub-block, we say they conflict and one cannot be used to mask faults in the other. When a line is detected as faulty (e.g., at boot-up using BIST), the system attempts to remap non-faulty portions of the line (Original line) to another line (called the Target line) that has already been marked as faulty and does not conflict with the

original line. It then sacrifices and disables the target line to replicate all faulty sub-blocks of the original line. Thus, the correct line can be reconstructed from a combination of the two lines (original and target).

Even without the additional complexities of the NUCA remapping introduced in this chapter, the selection of which lines to leave as original lines, which lines to scavenge as targets, and which target lines to assign to a faulty original line is a difficult optimization problem. Every time a line is disabled and marked for scavenging, it reduces the total capacity of the cache. Note that if a line has 4 sub-blocks, and assuming one faulty sub-block per faulty line, a target line could serve as a target for up to three original lines. In many cases, a good choice of the target is another line in the same set (referred to as a *Local* target line), since both original and target lines can be read in a single access. Barring this case, we should always select target lines from a different bank, so that both lines can be read without two serialized accesses to the same bank. We refer to this target line as a *Remote* target line. Because the two lines (original line and target line) are known to not have sub-block conflicts, and we know which sub-blocks contain errors, the cache controller can always reconstruct the original line using single bank remapping MUXing techniques. Figure 3.3 shows both the conventional and modified architectures of a 2-way set associative cache with four banks. One or two levels of multiplexing is used to compose a fault free block by choosing appropriate sub-blocks from both original and target lines, as shown in Figure 3.3(b). Note that the highlighted blocks are added to the original design. We now describe the configuration process for RESCUE-Cache.

In a RESCUE cache architecture, each cache access in low power mode (e.g., at a voltage level expected to be prone to errors) accesses both the original bank and DFM simultaneously. Figure 3.4 outlines the flowchart for accesses using the RESCUE-Cache. Based on the fault information of the accessed line, we either read one or two lines from one set in one bank (within set remapping), or one line from the original bank and one line from the target bank

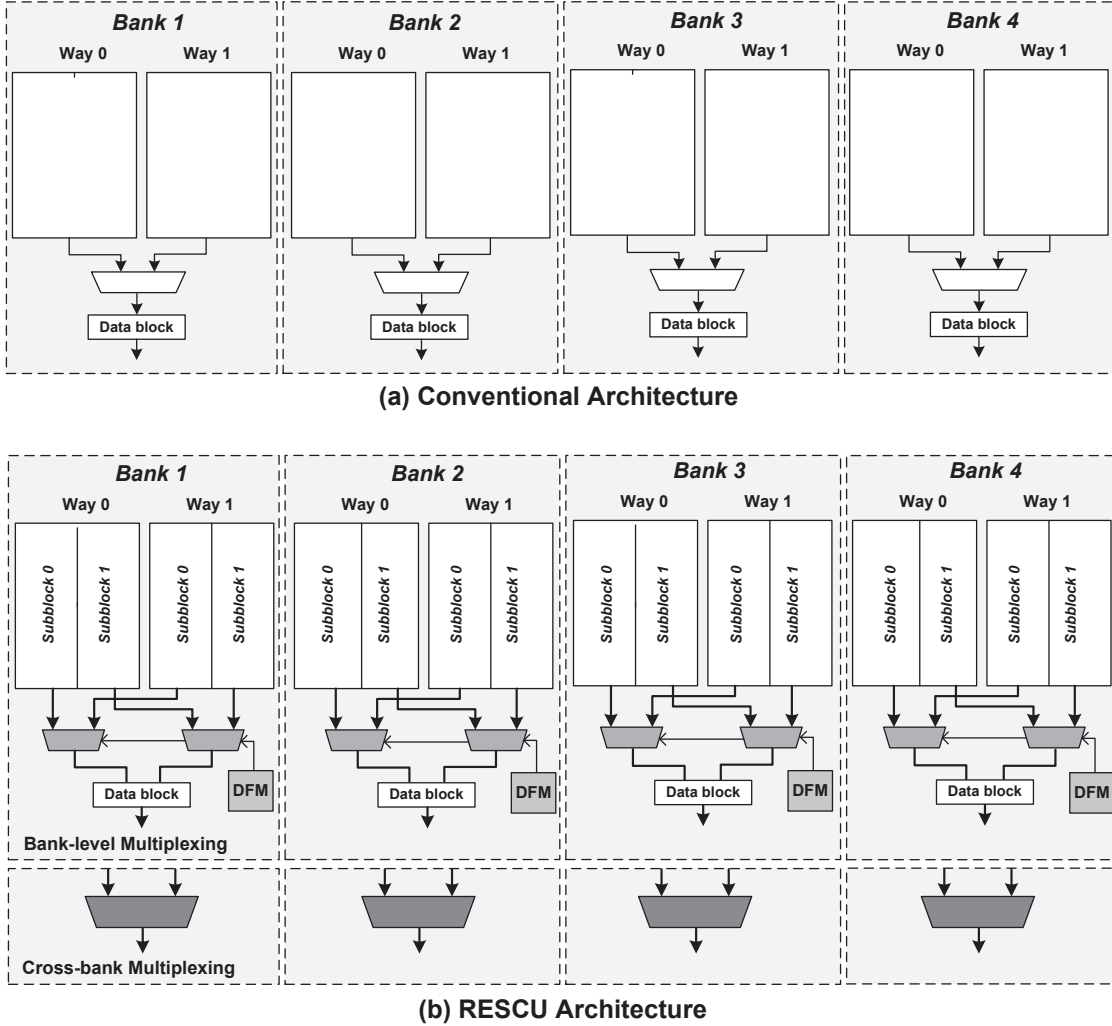


Figure 3.3: Architecture details of (a) a conventional 4-banked 2-way set associative cache, (b) the proposed RESCU with DFM and 2 sub-blocks per line.

(remote set remapping). In the latter case, the latency of the access is the maximum of the two bank access latencies. The DFM lookup is only in the critical timing path when the original line is faulty and is repaired with a remote line. In case of a faulty line access, one or two levels of multiplexing is used to compose a fault free block by choosing appropriate sub-blocks from both original and target lines, as shown in Figure 3.3(b). Note that the highlighted blocks are added to the original design. The configuration process is described in the next section.

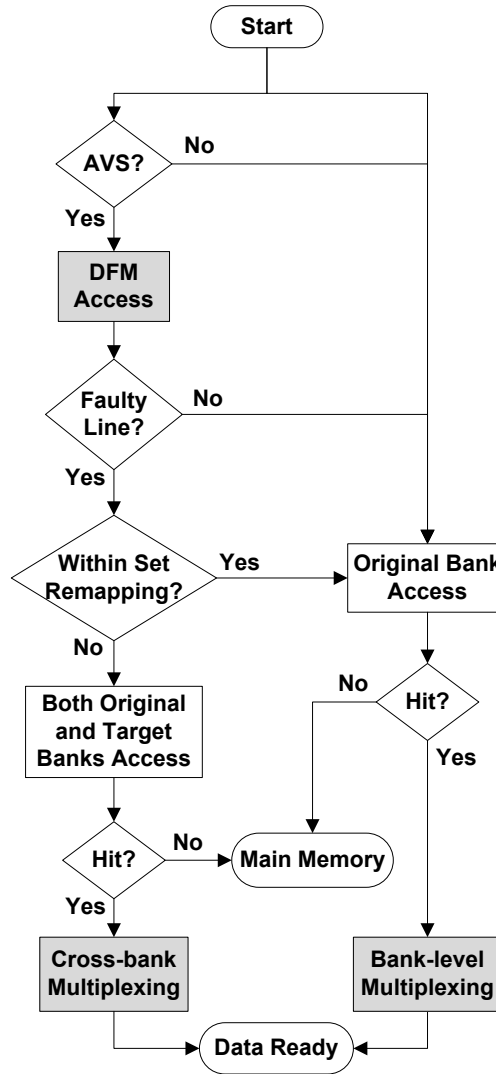


Figure 3.4: RESCUE cache access flowchart.

### Rescue DFM configuration

Initially, a raw defect map is generated at boot time; using the memory BIST unit, the LLC is tested under low voltage conditions. The output of the BIST is used to initialize the DFM for each LLC bank. If there are multiple operating points for different combinations of voltage, temperature and frequency, the BIST operation is repeated for each of these settings. The obtained defect map is then modified and updated with remapping data to enable the RESCUE operation. Updating the DFM is done at full voltage, using a graph coloring algorithm that we explain next. In addition, in order to protect the defect map and

the tag arrays, we use the well studied 8T SRAM cells [44] which scale  $V_{dd}$  down to 300 mV without failing, but incurs about 30% area overhead for these relatively small arrays.

The DFM of each bank has one entry for each cache set inside that bank. Each DFM entry includes multiple configuration bits, the defect map of each line in the set, status of each line, and the address of the target line, if any. Each bit in the defect map section represents the fault state of a sub-block. Line status bits represent the status of each line in a cache set. The status of each cache line can assume one of the following: 1) Non-Faulty, 2) Faulty, 3) Global, 4) Local Target, and 5) Remote Target. Initially, the status of all lines in the cache is Non-Faulty, representing the absence of any faulty sub-blocks. If a line contains at least one faulty sub-block, its status will be Faulty. A line used as a target line for other lines in the same set gets the status of Local Target. A line that has a conflict with other lines in a set and cannot be used as a local target line, gets the status of Global. A Global line can be used as a target line of another set, and then becomes a Remote Target. We define the Max Global Line (MGL) parameter as the maximum number of lines in a set that can be set as Global lines; the remaining lines can then be composed as a group of lines without conflict (no-conflict), which allows them to find a Global line and set it as their Remote Target line.

We use graph coloring to optimize the selection algorithm of a remote target line, for the stage 3 and 4 of the above algorithm [85]. Graph coloring is an NP-complete problem and there are many heuristics and transformation algorithms to make its solving feasible. We use a heuristic graph-coloring algorithm which is a modification of Saturation Degree Ordering (SDO) [120]. We name a graph coloring problem solvable if for a graph  $G$  we can find an integer  $K > 0$  such that the nodes of  $G$  can be colored with  $K$  colors while no edge exists between the same colored nodes. We construct a graph based on the conflicts between lines of different entries in all DFMs. Each node in this graph represents either a Global line (line node) or a faulty entry (representing an entire one set) from low conflict or high conflict groups (set node) in the DFMs. The edges represent a conflict between a pair of lines or

```

Begin DFM initialization algorithm
  1. Run BIST and find faulty cache lines at sub-block
    level and fill defect map sections of each set (DFM entry).
  2. For lines with at least one faulty sub-block, set the status
    of line as Faulty and for other lines with no faulty sub-
    block, set their status as Non-faulty.
  3. For sets with at least one faulty line mark their status
    as faulty.
  4. Based on the conflicts between faulty lines in
    each set, categorize them into four groups:
    - min-faulty if the number of faulty lines in a set is
      lower than the predefined MGL
    - no-conflict if there is no conflict between faulty lines
      in the set
    - low-conflict if the number of conflicts between the
      lines within a set is lower than the predefined MGL
    - high-conflict if the number of conflicts between
      lines within a set is higher that the predefine MGL
End DFM initialization algorithm

```

Figure 3.5: DFM initialization algorithm.

between a line and a set. For example, the two nodes connected by an edge represent two lines, two sets, or one line and a set that have conflict. Note that it is not possible to set one of these nodes as a Remote target line for the other one.

We modify the above graph coloring algorithm based on the following constraints: 1) We force the algorithm to color nodes from at least two different banks. 2) We force the algorithm to first pick up line nodes and try to color them with set nodes of other banks.

We apply the modified graph coloring algorithm to our graph to find a solution such that neighboring nodes are not assigned the same color. Therefore, after completion of the coloring algorithm, nodes with the same color are guaranteed to have no edges between them, implying that the corresponding cache sets/lines have no conflicts between them. We set all nodes with the same color in a group and try to set one of them as Remote Target for other nodes in the group. As a result, if a line node has the same color as one or more set nodes in other banks, we set it as their Remote Target line. If a set node has the same color as one or more

*Begin DFM configuration algorithm*

Traverse the faulty entries of the DFM:

1. For the entries in the min-faulty group just set the status of faulty lines to Global.
2. For the entries in the no-conflict group make one of their faulty lines as **Local Target** for other faulty lines in the same set.
3. For the entries in the low-conflict group, make the status of the line that has conflict with other lines as Global. For the remaining lines in the set (that have no conflict) find a Global line from a remote bank and set it as their **Remote Target**.
4. The entries in the high-conflict group are divided into sub-groups (according to the graph coloring algorithm). For the entries in each sub-group, select one of them and make all of its lines as **Remote Targets** for other entries in the sub-group.

*End DFM configuration algorithm*

Figure 3.6: DFM configuration algorithm.

set nodes in other banks, we set all of its lines as a Remote target line for lines in other sets.

## Reconstructing Cache Lines

Figure 3.3(b) illustrates the architecture of the RESCUE-Cache controller which is responsible for reconstructing correct lines. The new modules (MUXs and DFM) added to the conventional cache are highlighted in the figure. The additional multiplexer network would allow us to compose the final fault-free line from any of the sub-blocks in any of the cache ways in each one of four banks, based on its DFM data. Note that two levels of multiplexing are added to compose the final fault-free line, based on either multiple lines within a set or between two or more sets in different banks. The first layer is added on the cache bank side to perform the within-set remapping (Bank-level Multiplexing). Note that this layer replaces the original block-level multiplexer, available in set-associative caches. The second layer is added on the core side to perform remapping between sets in different banks (Cross-bank Multiplexing).



The total number of n-to-1 and 2-to-1 multiplexers required to compose the final fault-free line are  $k \times b$  and  $k \times c$ , respectively, where  $k$  is the number of sub-blocks in each line,  $n$  is the number of ways (set associativity),  $b$  is the number of banks, and  $c$  is the number of cores. For instance, for an 8-core processor with a 4-bank, 4-way set associative cache, with 64-byte blocks composed of 32 16-bit sub-blocks, a total of 384 (128 4-to-1 and 256 2-to-1) 16-bit multiplexers are required. For the described cache, the total DFM size will be less than 7% of total cache area. For a quantitative comparison, we synthesized the multiplexing layer and output logic for the RESCUE, as well as the multiplexer and output driver of a conventional cache (CC) using the Synopsys Design Compiler for TSMC 45nm standard cell library, for a 16MB cache. The area and delay of various multiplexers are used to estimate the overall area/delay overhead of the multiplexing network in RESCUE and the CC under nominal Vdd. We found that the delay of RESCUE-Cache output logic increased by only 5% compared to the conventional cache output multiplexing network while area and power consumption are increased by only 2%. These overheads are all modeled in our results.

## Remapping Comparison with Recent Techniques

In order to illustrate the effectiveness of RESCUE remapping for fault recovery, we compare its fault remapping ability against other state-of-the-art remapping techniques proposed to improve the fault tolerance of on-chip cache in low voltage operation. These techniques include RDC-Cache [138], Salvage Cache [86], Ansari’s scheme in [10], and Archipelago [9].

To compare these techniques with RESCUE we report in Figure 3.7 the effective cache size for each of these techniques across various voltage points. We use the failure rates based on PTM models in 45 nm [13]. The results reported in Figure 3.7(a) are based on a 1000-run Monte Carlo simulation for an 8MB 8-way set associative LLC with two banks and 64 byte block size. To have a fair comparison, for all these techniques including RESCUE, we assume fault map area overhead is at most 7% of the cache size (the area overhead of RESCUE with

16b block size). Based on this assumption and to meet the overhead limit, Archipelago algorithm selects 32-bit sub-block size. The sub-block size for RDC-cache and Salvage-cache are 128-bit and 64-bit respectively. The results show that in ultra-low voltage region (below 400 mV) as the failure rate increases, the more flexible methods like Archipelago and RESCUE result in smaller cache capacity loss.

Figure 3.7(b) presents the results for the same cache configuration but double the number of banks (total four banks) to evaluate the effect of increasing the number of banks on the effective cache size. Some of the techniques, like RESCUE, benefit from increasing the number of banks while others like RDC do not. In fact, for the RDC-Cache and Salvage cache the effective cache size for the two bank and four bank cases are fairly similar. For the Salvage cache this is mainly due to intra-bank remapping, where additional banks do not provide any more opportunity for remapping. Note that Salvage cache only relies on intra-bank remapping. The RDC-cache relies on a chain topology among banks, and therefore can at most apply remapping between two banks. For Ansari and Archipelago the improvement is small when moving from two to four banks, only a 5% effective cache size improvement across different voltage points. For the RESCUE we achieve up to 13% improvement in effective cache size compared to the two banks case. In addition, in the Ansari and Archipelago techniques, the fault map size depends on the fault rate and it would be increase noticeably as we reduce the voltage below 375mV.

At voltage points below 375mV, to achieve the same effective cache size as RESCUE, Ansari and Archipelago need to rely on a sub-block size of 4-bit or lower, which incurs an overhead of more than 20%. This is due to the fact that Archipelago and Ansari's fault map size is proportional to the number of original sets in a group that have the same target set. Because of such overheads, Archipelago places an upper bound on the maximum clique size. This makes Archipelago less effective for large caches with more than two banks. However, RESCUE addresses Archipelago scalability issues. The fault map is independent of fault

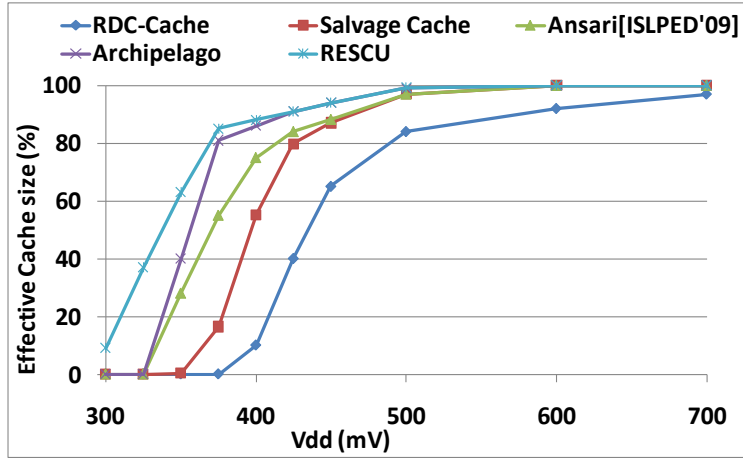
rate and is distributed to support multiple access point in a NUCA LLC with distributed banks.

RESCUE provides noticeably improved flexibility, allowing all banks to participate in remapping. This has an additional benefit – it also allows us to tune RESCUE to account for the difficult tradeoffs presented by a multi-banked non-uniform access cache. Running the LLC at an aggressively low voltage and frequency can have significant power advantages, but there is a cost (both in latency and capacity lost). When errors are frequent, bank accesses increase, interconnect traffic increases, and cache access latency increases as well. A good block remapping strategy should minimize these costs. This is the topic of the next section.

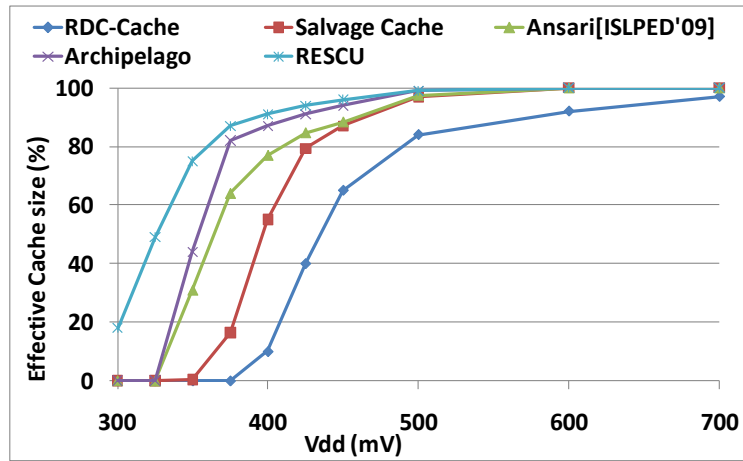
### **3.3.3 RESCUE Remapping Policies**

RESCUE leverages already faulty cache lines for redundancy to replace faults in other lines. In this way, the RESCUE cache automatically adapts the level of redundancy (and as a result, the amount of lost cache space) to the actual rate of failure. In a uniform cache architecture (UCA) with multiple banks and uniform latency, the baseline fault-tolerant cache configuration method presented in the previous section would work well. In such a scenario the choice of the RESCUE cache location (specifically, the bank for the target line) would have little impact on latency or interconnect bandwidth. However, as core counts and LLC sizes continue to grow, unified LLC designs become unrealistic. A NUCA design allows access time to scale with the average (or better) distance to cache banks, rather than the maximum distance, and enables use of a scalable interconnect to reach the cache banks.

There are several challenges to applying RESCUE in a distributed multiple-access-point NUCA environment. The mapping of lines to cache banks in a NUCA architecture has a significant impact on performance. Adding RESCUE remapping on top of that adds yet another dimension to the problem. Thus, the RESCUE remapping policy must balance



(a)



(b)

Figure 3.7: Effective cache size for different techniques with the (a) two banks and (b) four banks LLC.

several conflicting goals, including the desire to:

1. Minimize the maximum distance from core to bank. Cache latency will be determined by the maximum distance to the original and repair (target) banks. We cannot always predict what core will access the data, but if the repair bank is near the original bank, then it is more likely the maximum distance will not be inflated significantly.
2. Minimize the total distance from core to banks. Power and traffic congestion (and, as a result, latency) are minimized by minimizing the total number of hops from the requesting

core to each of the two banks. If we assume good NUCA mapping (bank close to requesting core), then remapping to a nearby bank likely minimizes the total hop count. When the NUCA mapping is not highly effective (e.g., SNUCA), remapping to a central bank can minimize the average hop count to the target line.

3. Maximize total cache capacity. The more freedom we have to select target lines, and the original lines that map to those repair (target) lines, the fewer total lines will be sacrificed.

Since the cache bank access pattern depends on the application that each core is running and dynamically changes with program behavior at run-time, a one-solution-fits-all approach may not deliver optimal results. Instead, we propose several heuristics for RESCUE remapping policies aimed at achieving a different balance of these goals. First, we design two algorithms that recognize the difference between SNUCA and DNUCA architectures – namely, the fact that with DNUCA you can assume the current mapping is a good one, while with SNUCA you cannot. The third policy attempts to minimize traffic by exploiting adjacency and the fourth places highest priority on preserving cache space.

### **DNUCA Optimized Policy (Minimum Latency - Moderate Capacity - Moderate Traffic)**

This remapping scheme is targeted at the Dynamic NUCA architecture. If the baseline DNUCA cache placement strategy is effective, data will be gradually promoted to banks closer to the cores that access it. We refer to these banks as core-neighbor banks and they are highlighted in Figure 3.8(a). As a result, the current location of a cache line becomes an indicator of what core is likely to access it. More importantly, the banks that are nearest the cores are more critical to performance than those in the middle. Therefore, in this scheme, we give highest priority to finding a repair (target) line in banks other than the core-neighbor banks. This preserves space in the edge banks, sacrificing space in the center.

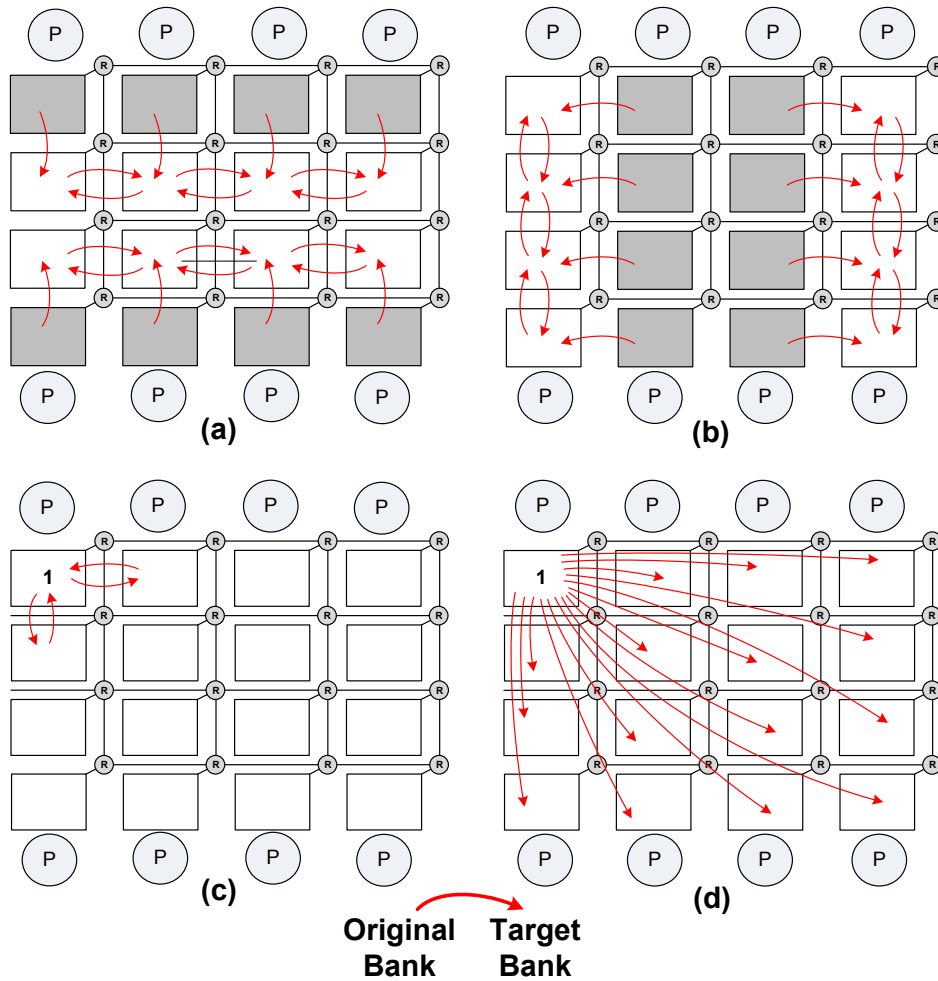


Figure 3.8: Example showing the (a)DNUCA optimized scheme with remapping limited to one bank with 1 hop (b)SNUCA optimized scheme with remapping limited to one bank with 1 hop (c)Adjacent mapping scheme (d)Global mapping scheme.

Figure 3.8(a) shows an example of remapping carried out using this scheme where the faulty lines in the core-neighbor banks are remapped to a target line in other (central) banks. In addition, for banks other than the core-neighbor banks, the priority is given to remap their faulty lines to each other as shown in the figure (preferably with one hop to minimize the latency).

### **SNUCA Optimized Policy (Moderate Latency - Moderate Capacity - Moderate Traffic)**

This remapping scheme assumes a Static NUCA architecture. In SNUCA the original mapping of data into banks is statically determined, based on the block index. In such architectures, access to the address space and all memory banks are the same for all cores. Therefore, unlike DNUCA, we have no information to indicate which core is likely to access a cache line. But there is an asymmetry which we can exploit in an n-by-n grid, as the average distance from all cores to a bank are not the same. So, the central banks shown in the figure that have minimum average distance from all cores are more important than the peripheral ones. Thus, we seek to preserve cache space in the center of the cache in this scheme.

Adjacency still matters in this scheme, because the average *max* distance to two banks is always minimized when those two banks are adjacent. However, this policy is more focused on minimizing the total number of hops (decreasing power and interconnect traffic). Figure 3.8(b) shows an example of remapping carried out using this scheme, with adjacency defined as 1 hop. As shown in the figure, in this scheme the preference is given for remapping the faulty lines in the central 8 banks to their adjacent outer banks. In addition for banks other than the central banks, the priority is given to remap their faulty lines to each other as shown in the figure (again preferably with one hop).

### **Adjacent Mapping Policy (Moderate Latency - Moderate Capacity - Minimum Traffic)**

In this remapping policy, the highest priority for placing the target line is given to adjacent banks. By doing so, it minimizes the maximum latency to either bank. Otherwise, it is less constrained than the prior heuristics. Figure 3.8(c) shows a simplified example involving only one bank, and again defining adjacency as one hop.

## **Global Mapping Policy (Maximum Latency - Maximum Capacity - Maximum Traffic)**

All of the other remapping policies potentially sacrifice cache capacity (i.e., prevent the technique from finding the optimal mapping by constraining it) for latency or traffic considerations. However, when lost capacity induces a miss, it can have far greater impact on both performance and power than a suboptimal line repair mapping. This remapping policy thus imposes no location restrictions that would constrain the mapper. However, this policy gives the priority to the adjacent banks and if the target line is not found in adjacent banks, it examines other banks. Figure 3.8(d) shows that any bank can be a source for a repair line of any bank.

### **3.3.4 Evaluation**

#### **Simulation Setup**

We use the M5 simulator [35] for our performance simulations. The M5 simulator supports four different CPU models to provide simulation platforms for detailed cycle accurate simulations. The memory instructions are modeled through the detailed memory hierarchy and network model mentioned in Table 3.1. The memory hierarchy uses a two-level directory-based MESI cache coherence protocol. To simulate accurate behavior of NUCA architecture and memory network we have extended M5 to include the HORNET [98] NoC simulator which simulates NoC architectures with high accuracy and detailed latency, network traffic, and power results. To enable power analysis, HORNET combines a dynamic power model based on ORION 2.0 with a leakage power model. This tool-set configuration provides a detailed memory-system model that enables us to model the NUCA cache architecture. Furthermore, it accurately models the network contention introduced by the



simulated mechanisms.

For the choice of core we study a high-end architecture which is an aggressive out-of-order Alpha ISA superscalar processor with issue width of 4, deep pipelining, and a complex branch predictor. Each core contains a split first-level (separate data and instructions) and a private second-level cache. The third level of the memory hierarchy is the NUCA cache. Table 3.1 summarizes the configuration parameters used in our studies.

We need a model of process variation, otherwise our SPICE models would show every cell failing at the same voltage level. To model the read/write time distribution as a result of process variation, we model an independent Gaussian distribution characterizing the  $V_{th}$  fluctuations of each transistor [133]. The circuit under test is a standard six transistor SRAM memory bit cell. The SPICE models used for the simulation were obtained from the Predictive Technology Model (PTM) [13]. Due to the random and uniform distribution of  $V_{th}$ , it is expected and verified by Monte-Carlo simulation that the read/write time to the cache follows a "Gaussian like" distribution. However as the supply voltage to the memory changes, the characteristic of read/write distribution is expected to change. We repeated 1K Monte-Carlo Simulation for each voltage point. By fitting the obtained iteration points at each voltage to the closest Gaussian distribution, we produce the expected mean and standard deviation.

The access latencies of the memory components are based on models made with the CACTI 6.5 [116] modeling tool, this being the first version of CACTI that enables NUCA caches to be modeled. We also used CACTI 6.5 to obtain the static power and dynamic energy per cache bank access. The dynamic energy consumed by the NUCA cache within the chip was modeled with the HORNET tool-set which uses the Orion simulator to determine the energy per bank access, the energy required to transmit a flit on the network link and the energy required to switch a flit through a network switch. The extra network traffic introduced by our proposal is also taken into account and accurately modeled in the simulator.

Table 3.2: Workload Mix, Spec2006 benchmarks are denoted with \_06

| 8-Thread Workloads | Benchmarks   |
|--------------------|--|
| WL1                | art_470, sjeng_06, perlbench_06_checkspam, mesa povray_06, omnetpp_06, eon_rushmeier, swim           |
| WL2                | bwaves_06, leslie3d_06, omnetpp_06, mesa, libquantum_06 vortex_3, crafty, perlbench_06_checkspam     |
| WL3                | crafty, h264ref_06_sss_encoder_main, hmmer_06_nph3 gap, lbm_06, parser, gcc_06_typeck, libquantum_06 |
| WL4                | galgel, apsi, lucas, gzip_log, bwaves_06, leslie3d_06 mgrid, namd_06                                 |
| WL5                | milc_06, gcc_06_typeck, gzip_log, cactusADM_06, gap lbm_06, gromacs_06, povray_06                    |
| WL6                | soplex_06_ref, fma3d, art_470, eon_rushmeier, h264ref_06 facerec, vpr_route, lucas                   |
| WL7                | sphinx3_06, bzip2_source, mgrid, parser, gobmk_06_nngs galgel, equake, astar_06_rivers               |
| WL8                | sphinx3_06 - facerec - mcf_06 - sixtrack - astar_06_rivers applu - fma3d - gromacs_06                |
| WL9                | vortex_3 - milc_06 - equake - bzip2_source - vpr_route cactusADM_06 - gobmk_06_nngs - soplex_06_ref  |

### Workload characterization

We compose multi-program workloads with  $P$  threads.  $P$  is equal to the number of cores in each CMP architecture we study (2 to 32 core CMP architectures). The applications are selected from among the SPEC2K and SPEC2006 benchmark suites, selecting representative sets of memory-intensive, compute-intensive and mixed workloads as shown in Table 3.2. A total of 32 different workloads are selected for each CMP architecture. For each application in the workload we fast-forward 2 billion instructions to skip the initialization phase and then simulate until all threads execute 2 billions instructions.

### Design Space Exploration

We study various LLC NUCA network design parameters and configurations and show how RESCUE can be effective in improving the fault tolerance for various configurations. The

main design parameters considered in our study include: remapping policy, memory cell SRAM bit failure rate for various operating voltage levels, and memory network configuration. The major network configuration parameters we considered include: the number of virtual channels (VC), number of crossbar ports (CP), and bandwidth (BW) of router links. We assume XY routing and study three network configurations based on the router parameters: 1) Low-Performance (VC=2, BW=1X, CP=1), 2) Moderate-Performance (VC=4, BW=2X, CP=2), and 3) High-Performance (VC=8, BW=4X, CP=4). We run our simulation process for six different fault rates, four proposed remapping policies, and three different network configurations. For each technology node the configuration with optimal number of cores is studied. We also examine multiple cache sizes. We show results for both SNUCA and DNUCA LLC architectures.

## RESCUE Overheads

Figure 3.9 summarizes the overheads of RESCUE for an 8-core CMP architecture with 16 MB (16 banks) LLC in 45nm technology, relative to an LLC without redundancy support. The power overhead in this figure is for the nominal Vdd (700mv). We account for the overheads of using 8T SRAM cells [44] for protecting the tag and defect map arrays in low-power mode. We assume clock gating and power gating is applied in the DFM array. Therefore, the main source of dynamic power in nominal Vdd relates to bypass MUXs. As shown in this figure it is less than 1%. The fault map area is the major component of area overhead. The total area overhead is less than 9%. In RESCUE, the DFM and MULTIPLEXING layer are on the critical path of LLC cache access. Based on our discussions in Section 3.3.2 and our timing analysis, we consider 2 additional cycles for LLC access latency overhead in low-power mode.

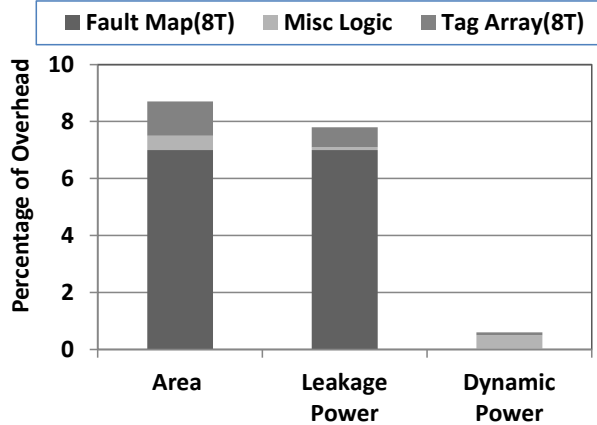


Figure 3.9: Area, leakage, and dynamic power overheads of RESCUE for an 8-core CMP with 16 LLC banks in 45nm.

### 3.3.5 Quantitative Comparison to Alternative Techniques

In order to illustrate the benefits of our design, we quantitatively compare RESCUE with the baseline 6T SRAM cell, two recent multi-bit ECC-based techniques (2D-ECC [82] and MS-ECC [46]), three recent remapping-based techniques (Ansari [10], FFT-Cache [22], and Archipelago [9]), and three other state of the art schemes (Bit-fix [156], 10T SRAM cell [38], and ZerehCache [8]). Table 3.3 summarizes this comparison - in 45nm - based on the minimum achievable Vdd, area overhead for the caches, power overhead, normalized IPC, and normalized power. In order to have a fair comparison, the remapping configuration and coding granularities are set so that the coding overheads of the ECC-based techniques are equal/comparable to disabled capacity of other methods (i.e 25%). In this table, different techniques are sorted based on their minimum achievable Vdd, when targeting 99.9% yield for on-chip caches.

Overall, RESCUE achieves the lowest operating voltage (360mv) and the highest power reduction compared to all other techniques. The closest techniques to ours are Archipelago, FFT-Cache, and 10T cell. However, 10T cell incurs a 66% area overhead and 24% power which are much more than our method overheads. Comparing to FFT-Cache and Archipelago,

Table 3.3: Comparison of different cache protection schemes

| Scheme      | Vdd-min (mV) | Area over. (%) | Power over. (%) | Norm. IPC | Power Norm. to RESCUE |
|-------------|--------------|----------------|-----------------|-----------|-----------------------|
| 6T cell     | 660          | 0              | 0               | 1.0       | 4.51                  |
| 2D-ECC      | 470          | 6.5            | 15              | 0.96      | 1.82                  |
| MS-ECC      | 440          | 6              | 10              | 0.90      | 1.61                  |
| ZerehCache  | 430          | 10.7           | 12              | 0.97      | 1.48                  |
| Bit-Fix     | 420          | 8              | 20              | 0.89      | 1.41                  |
| Ansari      | 410          | 15             | 8               | 0.96      | 1.33                  |
| 10T cell    | 380          | 66             | 24              | 1.0       | 1.24                  |
| FFT-Cache   | 375          | 10             | 8               | 0.95      | 1.21                  |
| Archipelago | 370          | 12             | 7               | 0.95      | 1.18                  |
| RESCUE      | 360          | 9              | 8               | 0.95      | 1.0                   |

RESCUE can achieve a lower Vdd and higher power saving while its overheads are less. Overall, the scalability of RESCUE for large shared NUCA caches along with efficient remapping, high configurability, and inherent flexibility allow it to tolerate higher failure rates in large CMPs compared to other similar techniques.

### 3.3.6 Experimental Results

In this section we evaluate the impact of multiple RESCUE remapping implementations on performance, power, and energy in the presence of aggressive voltage scaling. We also study whether cache over-provisioning can further change the power/performance trade-offs when RESCUE is being applied. We consider three types of workloads including Memory-intensive (MEM-WL), CPU-intensive (CPU-WL), and Mixed (MIX-WL). Our design space includes 26 different CMP core-cache configurations from 2-core to 32-core over five technology nodes from 65nm to 16nm. For each workload we report performance in terms of weighted speedup [147], normalized arithmetic mean IPC, and LLC network traffic statistics. For weighted speed up measurement we use the CMP architecture in which the LLC is operating at nominal VDD with no fault-tolerant mechanism, as a baseline. We also report

total power consumption (leakage+dynamic) of LLC memory and NoC structures. In the figures of this section we abbreviate different RESCUE mapping techniques as: AR = Adjacent Remapping, GR = Global Remapping, SN = SNUCA Optimized, and DN = DNUCA Optimized.

## Sensitivity to Voltage

In this section we study different design points on an 8-core high-performance processor at 32nm with a 16MB SNUCA LLC which uses Moderate-Performance (MP) routers for its cache network (we refer to this as the base CMP model). We study a voltage range of 0.7V to 0.3V. We refer to the voltages between 0.7V and 0.5V as **high-range** voltage, 0.5V to 0.4V as **middle-range** voltage, and 0.4V to 0.3V as **low-range** voltage. RESCUE remapping creates complex interactions and trade-offs between network traffic, useful cache size, and cache hit rate, power, energy, and performance. This section details a large suite of experiments aimed at understanding those trade-offs and identifying where RESCUE is most effective. Figure 3.10 shows the effective cache size after applying RESCUE. As shown for high probability of failure (low operating voltage), GR Policy is far more effective than all other policies in preserving cache capacity. For instance, at 0.4V, GR Policy reclaims more than 55% of lost cache capacity (without remapping, all cache is lost). GR mapping imposes no location restrictions that would constrain the RESCUE mapper. Therefore more cache capacity can be saved as more candidates for remapping are available to the RESCUE mapper.

## RESCUE Policies

Figure 3.11 shows the sensitivity of the RESCUE policies to different workloads for the base CMP model, and reports results along several dimensions of interest. The figure shows

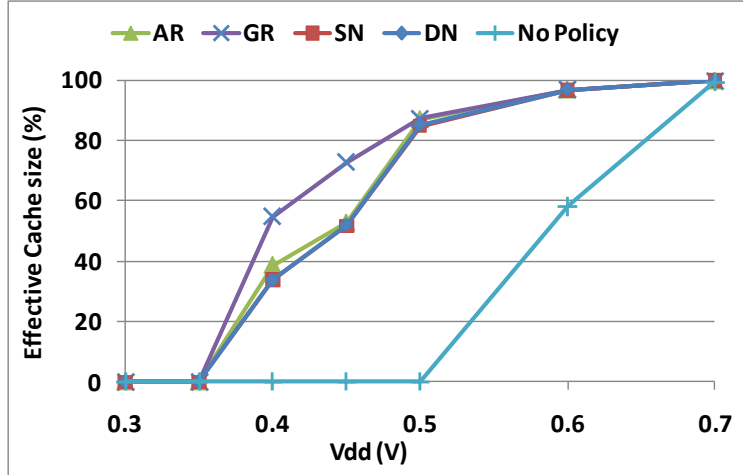


Figure 3.10: Effective cache size after applying RESCUE policies.

multiple performance and power results in terms of LLC number of accesses (1st row), average packet latency (2nd row), weighted speedup (3rd row), normalized IPC (4th row), total power consumption (5th row), and energy-delay product or EDP (6th row). A clear trend seen in the 1st two rows is the increasing LLC access rate and average packet latency as the voltage scales down below 0.60V. For high-range voltages, remapping is rarely necessary, while for low-range, so much of the cache is disabled that remapping is not much help. In the middle range, we see that remapping is active, resulting in higher traffic rates and a small increase in average latency. Note that while LLC latency is unaffected for very low voltage, the average memory access time is significantly increased, as the number of LLC misses is severely increased. In the 2nd row of figure 3.11, the SN policy, in general, has the lowest average packet latency over all three types of workload and across all voltages.

The 3rd and 4th row show the effect of RESCUE policies on performance parameters in terms of Weighted Speedup and IPC. A trend that can be seen in the 3rd and 4th rows is that as the voltages drops below 0.5V, the IPC and Speedup drop sharply as the voltage scales down. At these voltages, a large percentage of the LLC accesses are now misses and require main memory access. The 0.5V point is particularly interesting, because performance does not drop off considerably, despite the fact that this is the range where cache block

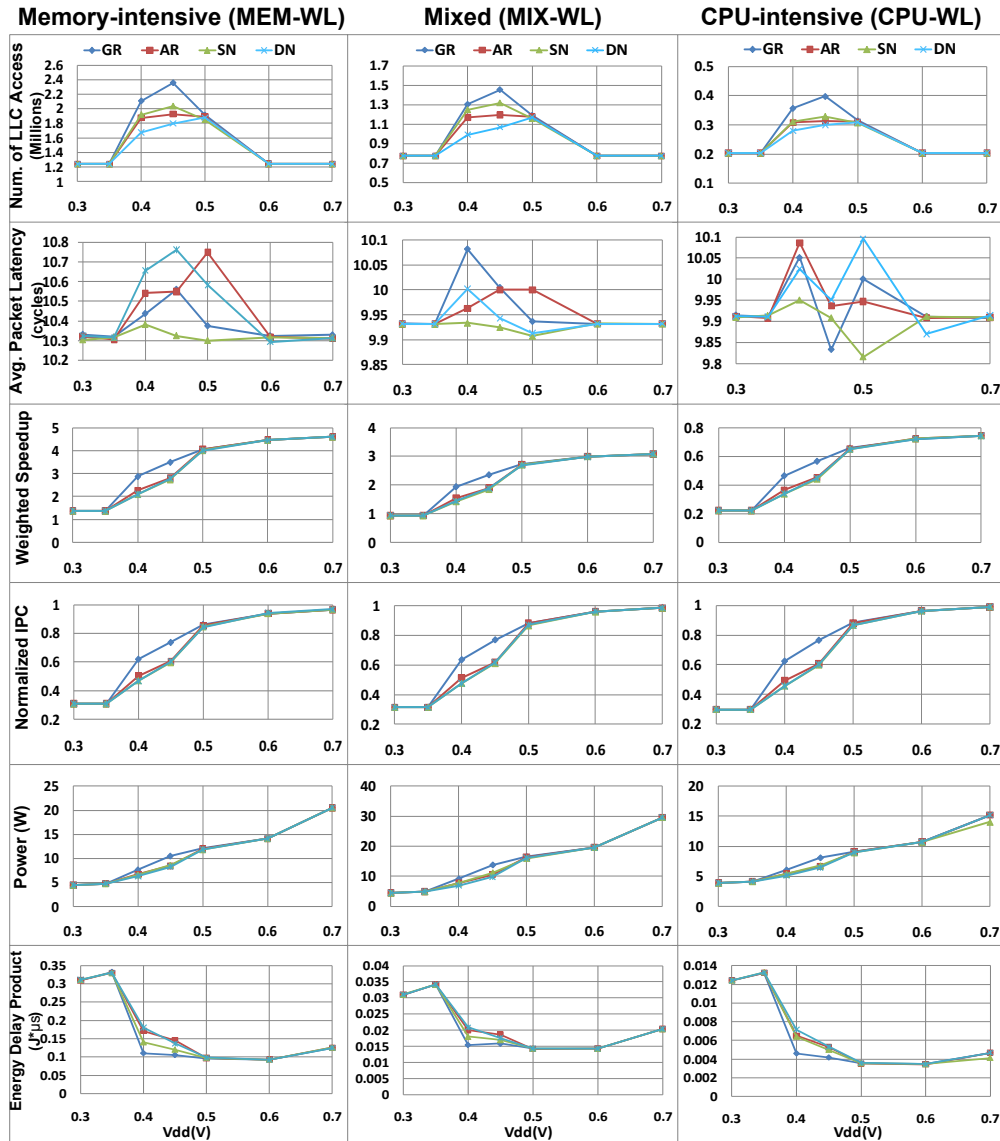


Figure 3.11: Sensitivity studies of RESCUE policies across various workloads.

failures become common and remapping activity starts becoming high. The GR policy gives the highest performance in this important mid-voltage range, and effectively extends the range over which high performance-loss is avoided. Since the GR policy achieves the lowest capacity lost among RESCUE policies, it has the lowest impact on performance parameters like Speedup and IPC.

A clear observation from the 5th and 6th rows is that, once again, the power and the EDP are sensitive to the RESCUE policies mainly in the middle range voltages. Compared to all



other policies, GR consumes the most power in the middle-range voltages, since it incurs higher LLC access rate. Despite this, because of its strong performance, the GR policy maintains the lowest EDP. In fact, we see that this policy enables us to maintain EDP near its minimum value even as we extend into aggressive voltage scaling regions.

## Network Configuration

By trading off aggressive voltage and cache failures with increased network traffic, the effectiveness of RESCUE mapping will be sensitive to the available bandwidth. To analyze the impact of network configurations on RESCUE, we report the results of our base CMP model with three different router configurations: HP, MP (our baseline), and LP as explained in Section 3.3.4. Figure 3.12 shows the effect of router configuration on performance and power results for the GR policy with the MEM-W1 workload. As we lower the voltage, we observe smaller performance impact in HP and MP network configurations (Figure 3.12b). For middle-range voltage the gap in average packet latency between low performance and high performance network is widening. This is expected, since in this voltage range, applying RESCUE remapping policies results in the highest network traffic (Figure 3.12a). Hence a low performance network can degrade performance. The power and EDP results are reported in Figures 3.12c and d respectively, and indicate the high performance router has higher power dissipation and EDP across all voltages. Despite giving up network latency due to contention, the power advantages of the least aggressive network outweighs the overall performance loss, resulting in a low EDP, and a much wider range of near-optimal EDP even into low voltage levels.

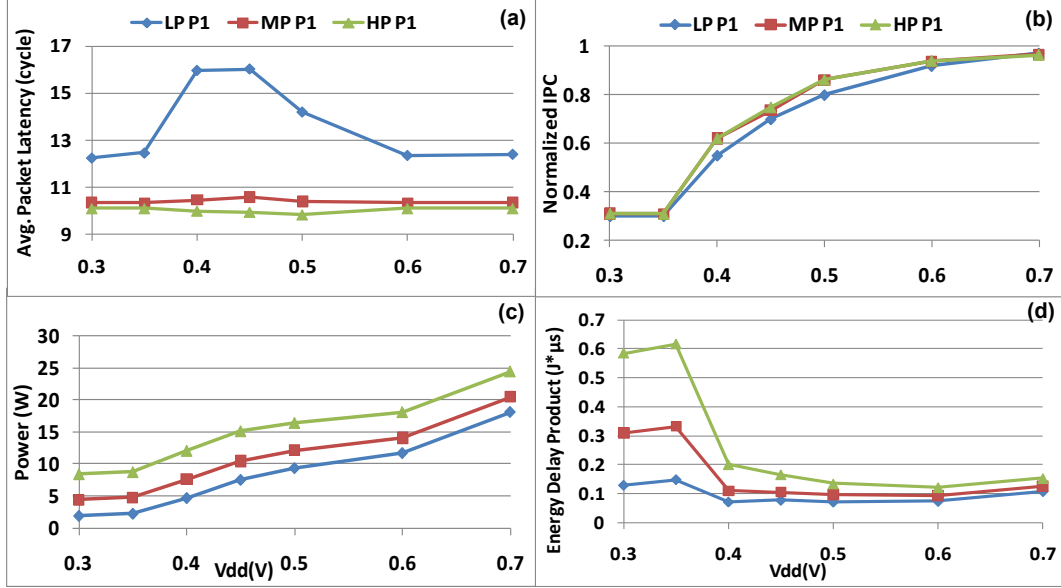


Figure 3.12: Sensitivity studies of network configuration with MEM-WL; Average Packet Latency (a), Normalized IPC (b), Total Power (c), and EDP (d).

## NUCA Policy

We expect remapping to be sensitive to the actual LLC configuration, which is why in fact we examine policies targeted at both SNUCA and DNUCA. We consider the base CMP model with two different NUCA policies (SNUCA and DNUCA) with MEM-WL. Figure 3.13(a) and (b) shows the efficiency of NUCA-sensitive policies of RESCUE. It shows that for the model with SNUCA the SN (SNUCA-Optimized) remapping policy of RESCUE has the lowest average network packet latency among other policies. Also, for the SDNUCA configuration, the DN policy (DNUCA-Optimized) has the lowest average latency among other policies. Another observation is that the latency of all policies across all voltages in the DNUCA configuration is less than the SNUCA configuration. By comparing figures 3.13 (e) and (f) we find that power consumption of all policies in SNUCA is slightly higher than DNUCA, across all voltages. Figures 3.13(g) and (h) show the effect of NUCA-sensitive policies on EDP. It shows that SN and DN policies are closer to the best performer for the middle and low range voltages for SNUCA and DNUCA configurations, respectively. Despite the traffic advantages of SN and DN, we find that GR still achieves the best performance in

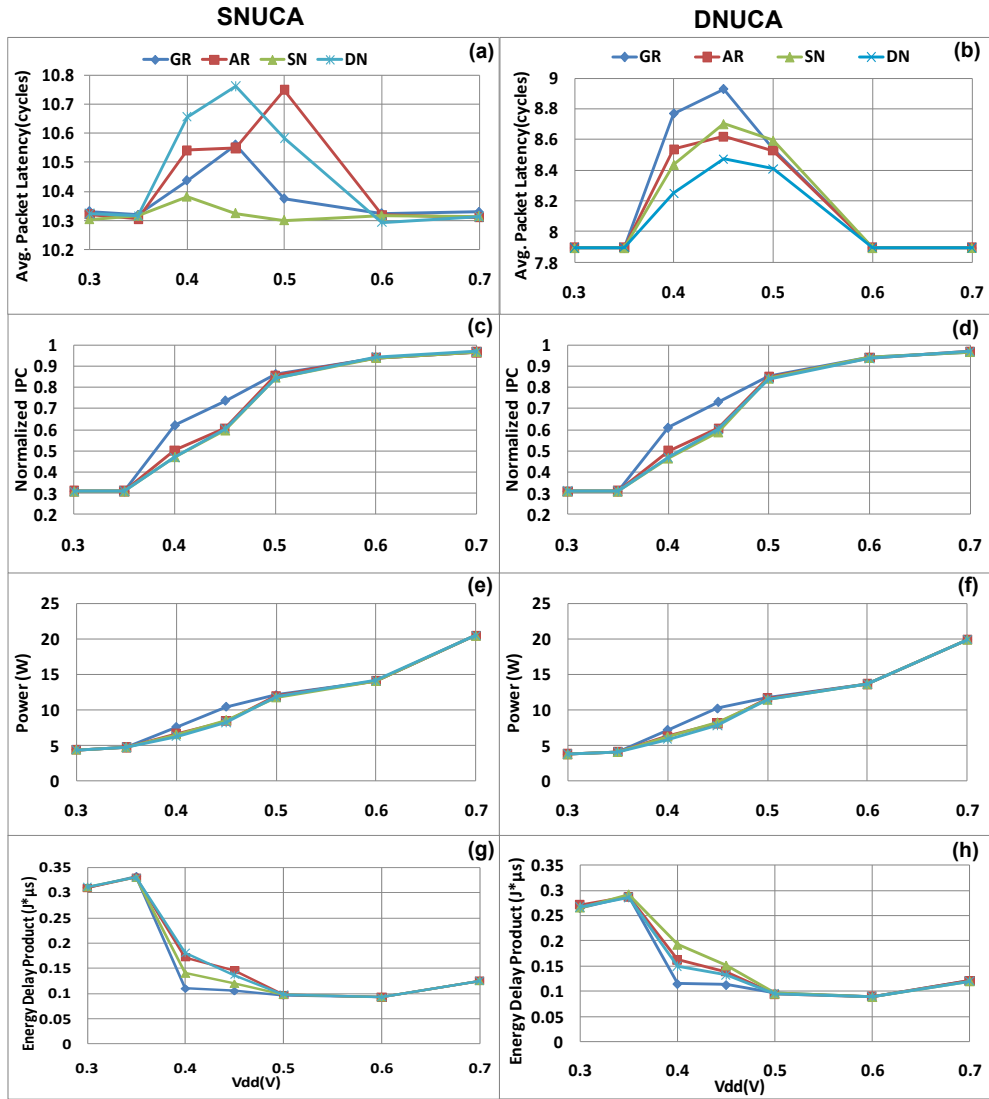


Figure 3.13: Sensitivity studies of NUCA policies with MEM-WL; Average packet latency (1st row), normalized IPC (2nd row), total power consumption(3rd row), EDP(4th row).

terms of weighted speed up and mean IPC. Our results continue to indicate that maximizing cache capacity is more important than minimizing traffic, whether we are considering raw performance or performance/energy.

### Cache size

RESCUE cache remapping creates interesting trade-offs between cache size, power, and performance. If we are designing a processor in a power-constrained environment (the common

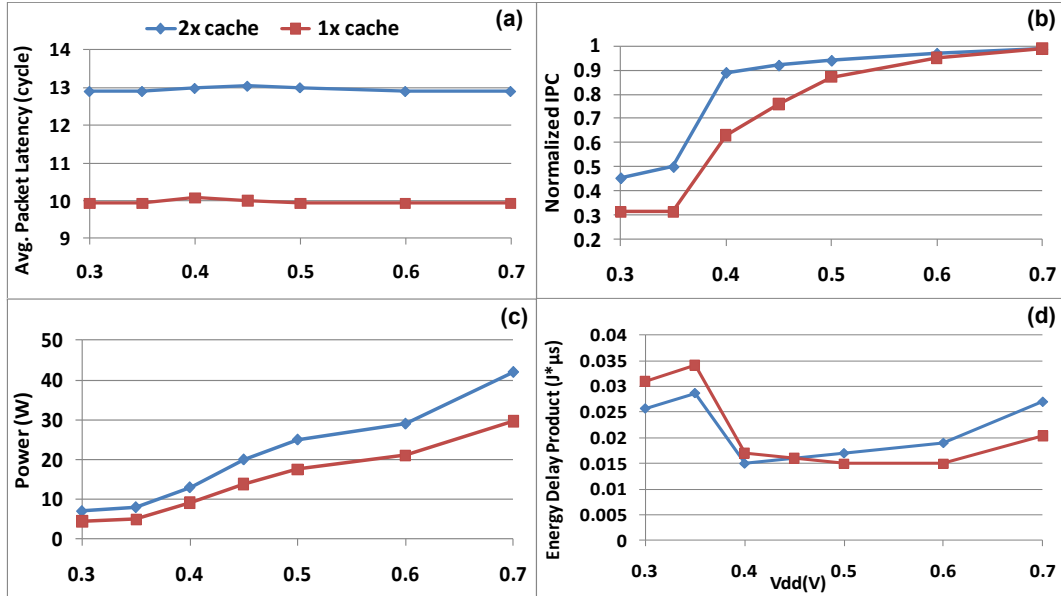


Figure 3.14: Sensitivity studies of Cache size over-provisioning with MIX-WL; Average packet latency (a), normalized IPC (b), total power consumption (c), and EDP (d).

case in this era), there may be little cost to over-provisioning the cache if it allows us to further scale down voltage. To study the effect of Cache over-provisioning on performance and power results we study two architecture; one is our baseline CMP with LLC cache size of 16MB (1X cache) and the second has the same architecture as our baseline with double LLC cache size; i.e. 32MB (2X cache). Figure 3.14 shows the results of our study for the GR policy with the MIX-Wl workload. Figure 3.14(b) compares the effect of voltage scaling on the IPC of 1x and 2x caches. It shows that the large cache allows us to maintain reasonable performance at more aggressive voltage scaling.

Figure 3.14(d) shows that for the high-range and the middle-range voltages the EDP of 2X cache is larger than 1X. The break-even point is 0.45 volt where the EDP of the two architecture are almost the same. The trend changes for the voltage below 0.45 volt, where the EDP of 2X cache is smaller than the 1X cache. Most importantly, an over-provisioned cache allows us to maintain near-optimal EDP at more aggressive voltage levels.

These results display exactly the area-power tradeoff that becomes increasingly attractive

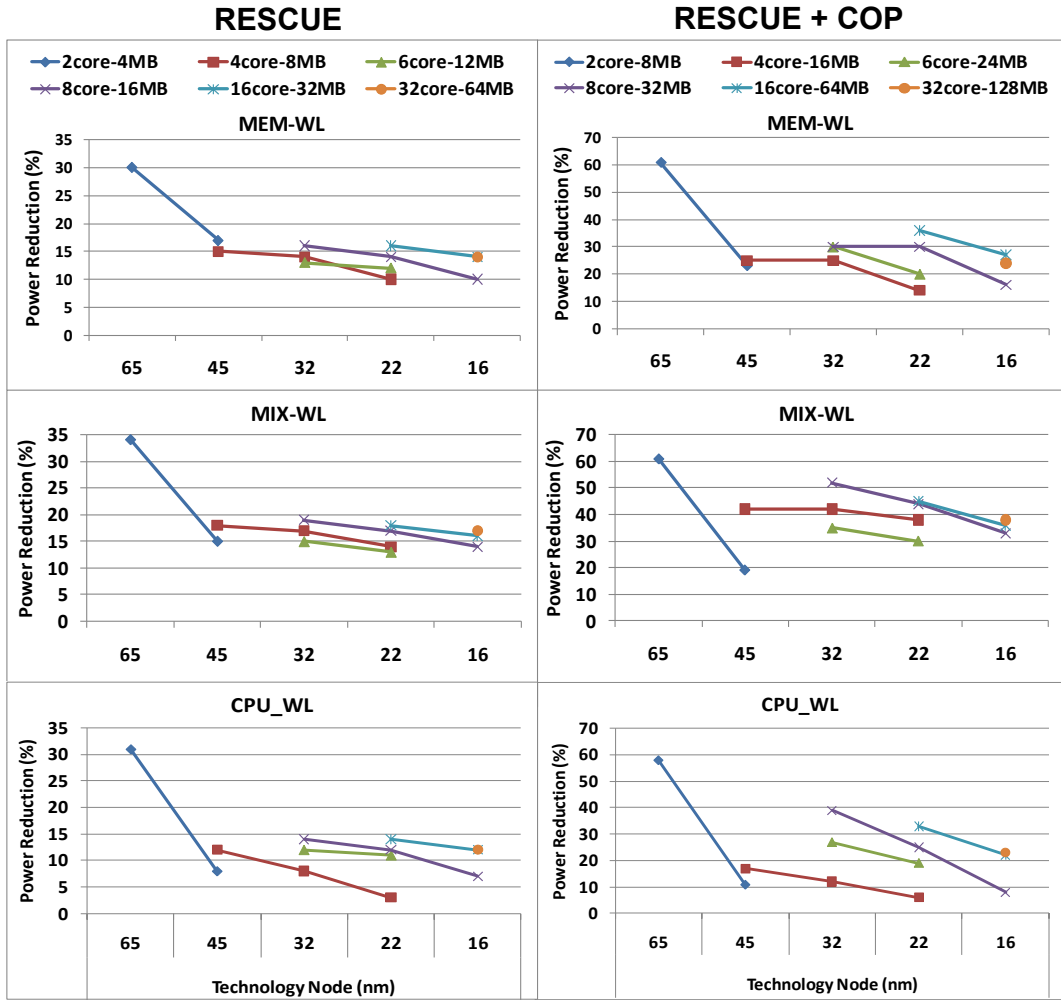


Figure 3.15: Power reduction across process technology nodes over all organizations and policies with MEM-WL, MIX-WL, and CPU-WL when applying RESCUE (left) and RESCUE+COP (right).

in future technology designs. By choosing a cache with twice the maximum capacity (twice the area), we are able to push voltage scaling from .5V to .4V with no performance loss, and actually hit a lower power point with the larger cache than the smaller cache. This architecture also gives us more flexibility. By configuring the over-provisioned cache at 0.4 V rather than the original cache at .5V, we pay no real power or performance cost in the common case (low voltage operation), but have the option of increasing voltage and getting full benefit of the larger cache when performance demands it.

## Technology

Figure 3.15 summarizes the power results to show the amount of power saving when applying RESCUE and RESCUE+COP (Cache Over-Provisioning) compared to the baseline architecture with no power savings mechanism. The figure shows that we gain the lowest amount of power saving in MEM-WL workloads and highest amount in CPU-WL in almost all core-cache configuration nodes. For CPU-intensive workloads, we can scale the LLC voltage and rarely pay the cost, but the more LLC accesses we observe in the workload, the more we pay a cost for the scaling. Comparing the figures in each row shows that for most of the core-cache nodes in all technology nodes, applying RESCUE and COP techniques together achieves 5 to 30% more power reduction than applying just RESCUE.

## 3.4 Summary

The design of NUCA LLC in CMP architectures is challenging due to the often conflicting tradeoffs of reliability, manageable power, and performance. In this chapter, we proposed RESCUE, a fault-tolerant scalable cache architectures for shared NUCA LLC in CMPs. This enables very aggressive voltage scaling to reduce the power dissipation of large last level caches in CMPs. RESCUE trades off cache capacity for power, a tradeoff that becomes increasingly attractive as we continue to follow Moore’s Law. RESCUE leverages address remapping to replicate faulty blocks of shared LLC banks with blocks from other banks. In remapping of faulty cache blocks, a single fault-free line can be reconstructed from multiple faulty lines. This chapter presents four different remapping policies with different primary goals. The proposed remapping policies leveraged for efficient selection of redundancy from different cache banks considering design challenges in a NUCA memory organization. In general, performance and energy-efficiency are highest when the mapping policies are unconstrained, minimizing the number of sacrificed cache lines. In future large, multi-bank, non-uniform cache access architectures, performance and routing power can be very sensitive to the remapping policy. Our experimental analysis shows that as cache operating voltage scales down, RESCUE increases available cache capacity and hence maintains performance even in the presence of high failure rates. We show that RESCUE is more effective in lowering power and EDP in the mid-range voltages (0.375 – 0.450 V). Our results indicate that it saves up to 60% power consumption of LLC cache in a 4x4 CMP architecture operating below 0.4V while recovering up to 55% of the faulty cache capacity with only modest performance degradation. RESCUE also shows that the trade-off of capacity vs. power can be very effective. An over-provisioned cache that is twice the original size can be more aggressively voltage scaled without sacrificing performance, resulting in a power point that is actually lower than the original cache.

# Chapter 4

## Interconnect-aware Resilient Cache Design

### 4.1 Introduction

Many-core platforms have become mainstream for the ever-increasing core count multicore systems where Last-Level Cache (LLC) is the largest shared on-chip memory component. LLCs are comprised of multiple distributed memory banks interconnected by a sophisticated communication fabric, often a Network-on-Chip (NoC). By increasing the complexity of such platforms, reliability becomes a major design challenge. A significant increase in both permanent faults and transient errors is expected due to advanced technology nodes, higher integration, voltage scaling, parametric variations, and higher power density that threaten the reliability of many-core platforms [118, 160]. Hence, both the cache banks and interconnect fabric (NoC) in LLC significantly suffer from potential fault occurrences. As the number of cores, cache layers, and memory banks increase in such platforms, there is a critical need to develop efficient, scalable, and modular fault-tolerant schemes to cope with the resultant



complexity.

There is a large body of previous work on fault-tolerant design of cache memories which have mostly focused on either single-bank or multi-bank caches with uniform access latency in a single-core architecture as explained in previous chapters. However, such schemes face severe limitations with increasing the number of cores and cache banks in emerging NoC-based multicore/manycore architectures. On one hand, most of them are not efficient, modular, and scalable enough to be leveraged for protection of shared caches with nonuniform access latency and distributed banks in such architectures. Since CMP architectures have more than one core that can access one or different distributed banks simultaneously, fault mapping and protection need to be scalable and distributed. Furthermore, existing fault-tolerant techniques typically use a single fault map and a centralized fault protection scheme which cannot be scaled for large CMP architectures: as a result each access needs to go through the fault map first, and no parallel cache accesses are allowed.

On the other hand, most of the proposed solutions are not applicable to LLC in emerging many-core platforms with error-prone interconnect [19]; they either consider reliable interconnects or assume their impact is trivial and therefore ignore them. There are only a few research efforts that address LLC fault tolerance in NoC-based CMP architectures [154, 19]. However, research on reliability of memory components in NoCs is still in its infancy [107]. In case of NoC, different fault-tolerant approaches such as Forward Error Control (FEC), Automatic Repeat Query (ARQ) [32], and multi-path routing [114] have been proposed in literature for reliable on-chip transmission. Error correcting code (ECC) approaches are very common in NoCs as well as memories to protect interconnect fabric and cache banks against errors, respectively [82, 32]. In particular, ECC schemes applied to caches suffer from high power/performance cost and lack of scalability when applied to the large LLC blocks with multiple errors. In NoCs, switch-to-switch (s2s) and end-to-end (e2e) ECC schemes are common practices [115, 123]. The choice of either an e2e or s2s ECC scheme in NoCs has

power, area, and performance trade-offs that vary with the fault rate, size, and topology of the system [141]. Another category of techniques are based on retransmission of data which impose higher packet latency and power consumption [115].

Our first observation is that almost all emerging CMPs are based on NoC platforms that provide a scalable interconnection fabric for connecting the cores, cache banks, and on-chip memory controllers. On-chip routers and links constitute this scalable communication backbone. Considering the limitations of available fault-tolerant schemes and power challenges of emerging CMPs beside NoC capabilities motivates us to leverage the interconnection fabric available in NoC-based CMPs to implement the fault-tolerant approach. Hence, the efficient use of the interconnect backbone in CMPs helps designers develop more efficient fault-tolerant schemes for protection of on-chip memories using its high flexibility, regularity, parallelism, and scalability to detect faults, correct them, and manage redundancy among different distributed memory banks.

We also observe from related work that all previous approaches consider protection of cache and interconnect in isolation of each other. Another observation is that to minimize the cost of LLC protection, cache and interconnect need to be considered together and ignoring one part makes the protection of other part inefficient. Moreover, in many-core architectures with distributed shared LLC banks where a large fraction of cache accesses are from remote cores, it is necessary to protect data blocks against errors throughout the interconnect network. Existing approaches use a direct integration of conventional error protection schemes available for both cache and NoC interconnect together in current multicore architectures. However, since such schemes are designed in isolation of each other, this approach leads to a high level of redundancy with overlapped blocks which makes it inefficient. Consequently, a unified approach that guarantees the end-to-end protection of shared cache blocks without imposing a conspicuous overhead is essential for future many-core architectures.

Our observations we mentioned above motivate us to propose two novel interconnect-aware

approaches to on-chip memory resiliency in NoC-based many-core platforms. While the first approach considers the scalability of interconnect, the second approach considers the impact of erroneous interconnect in design for memory resiliency.

As the first major contribution in this chapter, we present a novel solution at the interconnection network level for fault tolerance of LLC in NoC-based CMP architectures [19, 21]. We leverage the NoC fabric to implement a fault-tolerant scheme protecting the LLC cache banks against permanent faults in a scalable and efficient way with minimal overhead. During an LLC access to a faulty block, the components inside the network (routers) are responsible for detecting and correcting the faults and then returning the fault-free data to the corresponding core. We propose four different policies that leverage the NoC fabric in different ways to implement the basic fault-tolerant method. We then perform cycle-accurate simulations using well-known NoC benchmarks to evaluate our solutions. Our results show that some of these policies take advantage of the intrinsic parallelism of the communication mechanism in different ways, leading to a trade-off regarding the constant use of the NoC (increasing energy consumption) and lower latency. To help evaluate these trade-offs, we also perform sensitivity analysis on different policies by improving some aspects of the network-on-chip such as number of virtual channels and the use of an adaptive routing algorithm. Our results show that some policies react better than others to these modifications, which shows that different policies take advantage of the communication parallelism in different ways, allowing system designers to tune the use of specific policies. To the best of our knowledge, this is the first work that leverages the on-chip network fabric to facilitate fault-tolerant design of on-chip memory (cache) in NoC-based CMPs.

The second major contribution of this chapter is *CoDEC*, a Codesign approach to unify the error coding of cache and interconnect in many-core architectures in order to minimize the overhead of error protection. CoDEC deploys a united error coding scheme that guarantees the end-to-end protection of LLC data blocks throughout the on-chip network against both

hard and soft errors. CoDEC integrates strong multi-bit ECC in cache banks with s2s error coding of interconnect while removing their overlapped redundancy. It partitions each LLC block into multiple equally-sized segments, extends each segment with a low-cost ECC, and transmits each extended segment as a flit in NoC. The main advantage of the CoDEC approach is eliminating the overlapped redundancy of ECC logic in cache bank and that of interconnect by integration of multi-bit ECC in cache banks and single bit s2s ECC of NoC. Using this approach, CoDEC eliminates the large ECC encoder/decoder blocks from the critical path of LLC global access through the network. Our technique minimizes latency in the common case of accessing a shared LLC bank over the network, and potentially accessing a local LLC bank, while providing almost the same error protection as strong multi-bit ECC in cache blocks using a segmented per flit ECC. Our evaluation on a  $4 \times 4$  platform with mesh NoC shows that compared to a conventional approach, CoDEC improves the performance by up to 14% and around 5.5% on average. Also, our implementation results show that with respect to the baseline architecture, CoDEC improves cache access latency for global and local accesses by around 50% and 20%, respectively, with only a minimal overhead of less than 1% area and 2.5% ECC storage. To the best of our knowledge, this is the first *Codesign* approach to protect both cache and interconnect against errors in a NoC-based many-core platform.

## 4.2 Related Work

Our work represents a convergence of three main bodies of related research: fault-tolerant cache design, fault-tolerant NoC interconnect, and memory reliability in multi/many-core architectures.

### 4.2.1 Fault-tolerant Cache Design

There is a rich body of literature on design for error resiliency in cache memories as discussed in previous chapters. Several architectural techniques have also been proposed to improve reliability of on-chip caches by employing relatively sophisticated fault tolerance mechanisms, such as block/set pairing [2, 156, 131], address remapping [9], block/set-level replication [22], etc. In sum, since most of these techniques are designed for a single-core processor with one or two cache banks and uniform cache access designs, none of them considers the scalability and erroneous issues of interconnect on their approach, unlike our proposed approaches in this chapter.

### 4.2.2 Fault-tolerant NoC design

Reliability and fault-tolerance are outstanding research challenges in NoC design [107]. There are many efforts to investigate the robustness of NoCs, mostly in the areas of routing algorithms [129, 127], communication infrastructure [36, 115, 123, 141], or micro-architecture [83, 7, 71]. A fault-aware IP-core mapping to NoC routers is proposed in [105]. They address the problem of transient link failures by means of temporally and spatially redundant transmission of messages. A stochastic communication paradigm is proposed in [36] to provide a fault-tolerant communication. This approach is based on probabilistic broadcast where packets are forwarded randomly to the neighboring nodes. However, none of them protect faulty datapath inside routers. In [32], the authors deploy some error correcting schemes to achieve combined energy minimization and reliability optimization design goals. A fault-tolerant router architecture is proposed in [57] to guarantee the functionality of the NoC in the presence of faults. They detect the most tenuous components of router against different sources of faults first and then tolerate them to save power and area overhead of employing fault-tolerant techniques.

### 4.2.3 Memory Reliability in Multi/Many-cores

There are some recent efforts addressing memory resiliency in NoC-based CMP architectures. Angiolini et al. [7] presented a mechanism to maintain a reliable integrated memory subsystem for a NoC-based system. The idea is to have a reliable backup memory (in addition to the main memory) to store all the critical data. The NIs are responsible for redirecting the critical data access in case of a failure in the main memory. Although, this scenario provides some percentage of reliability to the memory, it has drawbacks like redundancy of data and not being transparent to the programmer. In another work [154], Wang et al. addresses fault-tolerance of NUCA cache in NoC-based CMPs. They proposed a utility-driven address remapping technique to tackle the capacity loss in NUCA cache of NoC-based CMP architectures. However, their address remapping technique is at the bank-level, and they have not leveraged the NoC fabric for fault-tolerance of the cache.

Our approach is different from all the related work as follows. Our approach is orthogonal to all previous work in different categories mentioned earlier. We neither propose a new cache fault-tolerant technique nor a NoC fault-tolerant technique. Here, we propose interconnect-aware approaches to LLC protection in two different ways. In first contribution, we describe how to leverage the NoC fabric to implement an available fault-tolerant cache technique in a scalable and efficient way with minimal overheads for emerging NoC-based multicore/manycore platforms. In second contribution, we consider the impact of an erroneous interconnect fabric in LLC protection and propose a codesign error coding technique to consider both cache and interconnect.

## 4.3 NoC-based Fault-tolerance of LLC

### 4.3.1 Basic Idea and Baseline Architecture

Unlike traditional fault-tolerant cache schemes, our approach migrates all of the fault-tolerant bookkeeping from the cores to the NoC fabric by modifying the network routers to perform all the required tasks of a fault-tolerant scheme. As an exemplar of a fault-tolerant scheme, we use a remapping-based technique to demonstrate the effectiveness of our proposed approach. Remapping-based schemes address permanent faults by remapping faulty blocks to other faulty/non-faulty blocks, thereby ensuring delivery of fault-free blocks. We propose four policies to configure the NoC, implement the fault-tolerant scheme, and based on each policy we add a fault map and a multiplexing layer to the appropriate NoC components. The fault map keeps information about the location of permanent faults and remapping data in the local cache bank connected to each router. The multiplexing layer performs the fault matching.

Using the NoC interconnection paradigm enables us to implement any fault-tolerant method in an efficient, modular, and scalable manner for several reasons. First, it supports modular and scalable implementation of fault-tolerant methods for emerging multi-/manycore architectures. Different policies and configurations are possible with the minimum overhead and design change. Second, using the NoC paradigm, it is easy to control access and management of redundancy resources. Also, redundancy for fault tolerance can be provided easily via the available interconnection network for any entity in the NoC. Third, adaptive and dynamic fault tolerance of memory banks can be supported via NoC components. Since the routers have a key role in communication and management of data between all core and cache nodes in a NoC-based CMP architecture, by using their network information and modifying their routing algorithm, we can easily provide adaptive fault tolerance of cache banks.

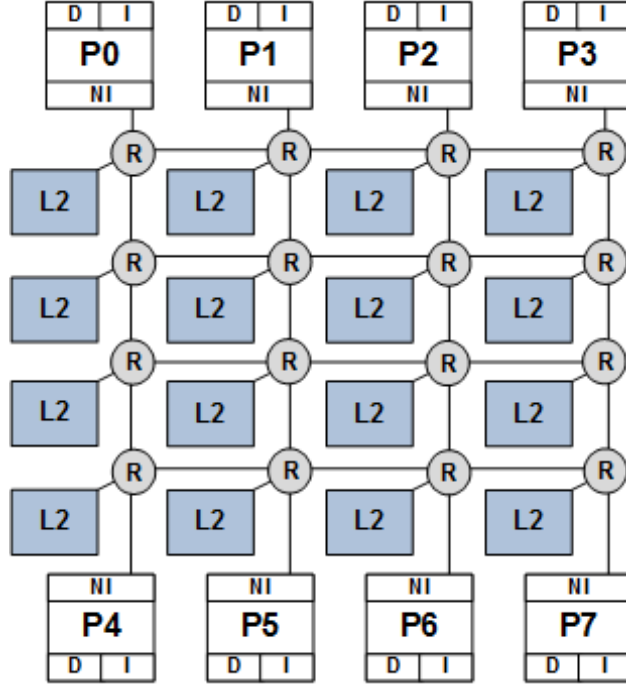


Figure 4.1: Baseline NoC architecture.

## CMP and NUCA LLC

We consider a nontiled CMP architecture as our baseline. The architecture is similar to Oracles SPARC T3-1B processor composed of 8 cores and 16 shared LLC banks [142]. Cores and cache banks are interconnected as a 2D mesh via a Network-on-Chip (NoC) architecture as shown in Figure 4.1 where the L2 cache is located in the middle (CIM) [80, 49]. Each core is composed of a processing element and private L1 data and instruction caches. Based on our cache organization, each LLC bank is a portion of a larger distributed shared LLC cache.

The baseline design assumes a Non-Uniform Cache Architecture (NUCA) [81] LLC cache. With multiple banks within the LLC cache, we have the choice of either always putting a block into a designated bank (static mapping) or allowing a block to reside in one of multiple banks (dynamic mapping). We consider static mapping in our baseline design and model a static NUCA policy for CMP architectures (CMP-SNUCA) [30]. Similar to the



original proposal (S-NUCA for uncore processors in Kim et al. [81]), CMP-SNUCA statically partitions the address space across cache banks connected via a 2D mesh interconnection network.

In static mapping, a fixed hash function uses the lower bits of a block address to select the target bank. LLC access latency is proportional to the distance from the issuing L1 cache to the LLC cache bank. By allowing nonuniform hit latencies, static mapping reduces hit latencies of traditional monolithic cache designs, which fixes the latency to the longest path [3]. Because a block can be placed into only one bank, the LLC access latency is essentially decided by the accessing block address.

## NoC Architecture

Networks-on-chip are based on conventional computer networks, and have emerged as an alternative to buses to deal with the issue of scalability and to provide a regular architecture in order to provide reusability. Indeed, because of their regular architecture, NoCs are very suitable for platform-based design. The NoC exemplar used in this work is a cycle-accurate SystemC implementation based on SoCIN (SoC Interconnection Network) [163]. SoCIN is a mesh topology scalable network based on a parametric router architecture conceived with the goal of instantiating a low-cost NoC. The main modules of this implementation of the SoCIN router are the input buffers, the arbiter, the crossbar, and the links as depicted in Figure 4.2(a). SoCIN is a wormhole packet-switched network, where each packet is divided into small units called flits (flow control unit). Each flit contains  $n + 2$  bits where  $n$  is the payload and the 2 bits are flags for begin of packet (*bop*) and end of packet (*eop*). When a flit arrives in the router it is stored in the input buffer. Once the arbiter reads the first flit (identified by the *bop* flag) it can determine through which output port the packet must go. At this point, when multiple packets need to use the same output, the arbiter must choose which one must go first. This is decided by using a simple round-robin scheduling. This

information is passed to the crossbar. If the crossbar determines that the output request is available, it begins to transmit the packet. The condition that defines the availability of the output is that the input buffer of the router connected to it needs to have enough space for at least one flit.

SoCIN implements a deterministic XY routing that avoids deadlocks. The routing is determined by the NoC address of the destination specified in the very first flit of the packet (the  $n/2$  most significant bits are the destination NoC address and the  $n/2$  less significant are the source NoC address). Once the first flit is routed through an output, the handshake protocol between the routers defines when there is enough space in the input buffer to send the next flit. This happens until the flit with the *ep* flag is routed. Therefore, all flits are routed in a pipeline fashion. Figure 4.2(b) shows the packet format in SoCIN. The flow control in SoCIN is handshake based. When the sender writes the data on the link, it signals a valid (*val*) bit. When the receiver is ready to read the data it activates the ack signal. Figure 4.2(c) depicts the link structure with two unidirectional channels (to send and receive) each one of them has  $n + 2$  bits and *val* and *ack* signals.

## Base Fault-tolerant Method

In general, all fault-tolerant schemes leverage some form of redundancy to tolerate the faulty components. As an exemplar for fault-tolerant caches, we consider remapping-based techniques as our base fault-tolerant method from the work in [22]. This method leverages a portion of faulty blocks as redundancy to mask faults in other blocks in cache memories. Here, we present a brief description of this fault-tolerant method. This method uses a BIST module during boot-up to detect permanent faults (hard errors) in cache blocks and keeps the information of the location of faulty blocks in a fault map. Then, based on this information, it configures the address remapping for faulty blocks. This method divides each block of the cache into multiple subblocks that define the granularity at which failures are

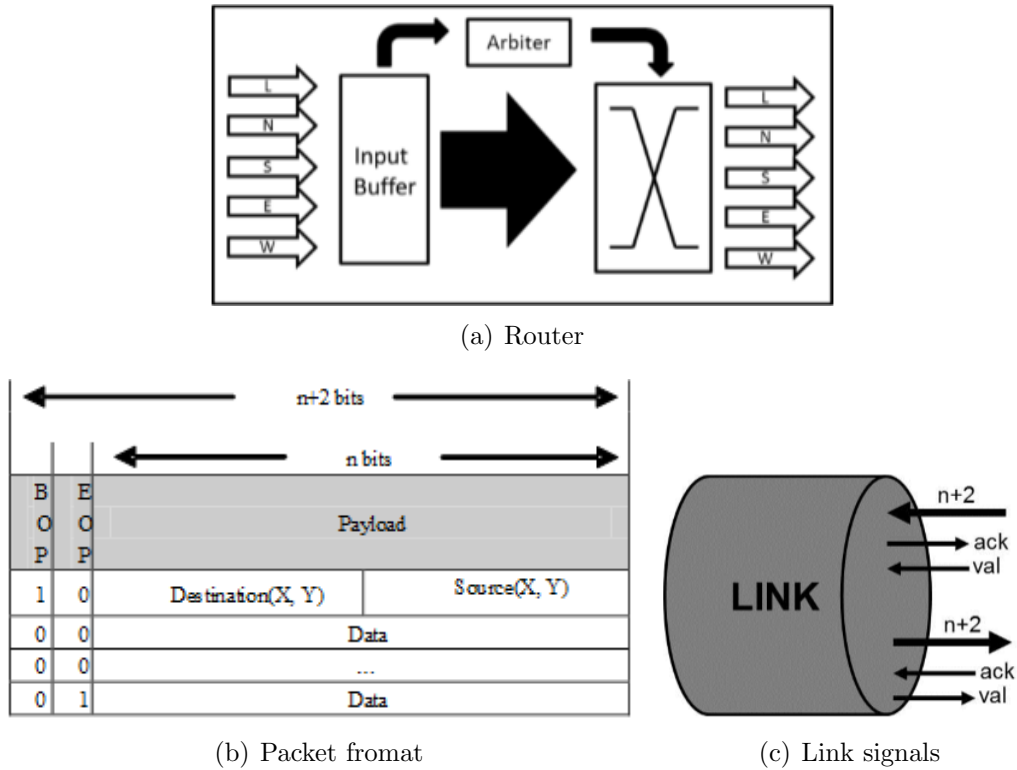


Figure 4.2: SoCIN component details.

identified and remapped. A subblock is labeled faulty if it has at least one faulty bit, as determined by BIST analysis. If two blocks have faults in the same subblock, it means they have a conflict and one cannot be used to mask faults in the other. During the configuration process if a block is labeled as faulty, the system attempts to remap faulty portions of that block (host block) to another block (called the target block) that has already been marked as faulty and does not conflict with the host block. It then sacrifices and disables the target block to replicate all faulty subblocks of the host block. Thus, the fault-free block can be reconstructed from a combination of the two blocks; that is, the host block together with the target block via leveraging a multiplexing layer.

In many cases, a good choice of the target is another block in the same set (referred to as a local target block), so both host and target blocks can be read in a single access. Barring this case, we should always select target blocks from a different bank (a target bank), so that both blocks can be read without two serialized accesses to the same bank. We refer to

this target block as a remote target block. Note that the target block can be selected from any bank in the LLC address space. Here, for each bank we limit the remapping policy to select target blocks from the adjacent banks that are within a one-hop distance in the NoC.

### 4.3.2 Proposed Architecture

We extend the base fault-tolerant method to be leveraged for a shared multi-bank NUCA cache in a CMP architecture using the NoC backbone. Recall that the basic idea is that we distribute the fault map and multiplexing layer over routers of the NoC architecture. There exists one entry in a fault map for each set. We divide each entry to two sections: map of faulty subblocks and remapping data. The first section keeps the map of faulty subblocks in the set. The second section represents the remapping data of all blocks in the set. We keep the fault map of each bank in two separate blocks. We keep the first section of the fault map in a Fault Map Block (FMB). The second block is composed of the remapping data of the fault map which is named Remapping Data Block (RDB). RDB includes one bit that indicates the faulty status of each block. We need the FMB data only to configure the address remapping and initialize the RDB. Therefore, there is no need to access this FMB data during the operation of the system, and we can keep it in main memory or the hard drive. We only need to keep the RDB data at the cache level. Note that we employ a reliable 8T SRAM cell [38] to protect RDB and also tag arrays. To implement our approach, we perform modifications in three levels to a cache access algorithm, the NoC routers, and cache banks to support the fault-tolerant method, as described in the next sections.

#### Fault-tolerant Cache Access

We modify the LLC access algorithm to support the fault-tolerant method. In our fault-tolerant cache architecture, each LLC access first accesses the RDB of the fault map. Based

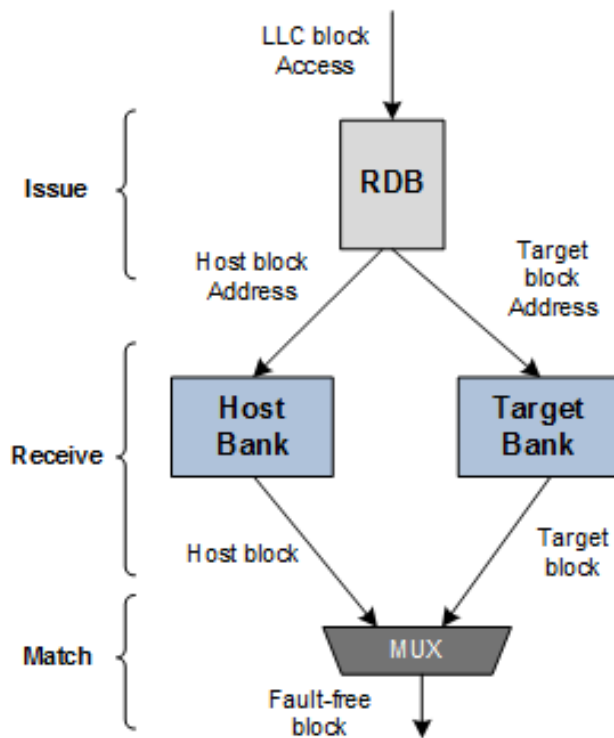


Figure 4.3: Three phases of a fault-tolerant LLC access.

on the fault status bit of the accessed block, if it is a faulty block it needs to have an extra access to a target block. In our model, we consider that this reading process from the RDB creates an overhead of one clock cycle but this only happens one time during the L2 access and only when there is an L1 cache miss. The target block can be a local one within the same set or may be accessed from another bank (in case of a remote target block). Then, based on the location of the target block retrieved from the RDB, one or two levels of multiplexing are used to compose a fault-free block by choosing appropriate subblocks from both host and target blocks.

Based on our modified cache access, we define a fault-tolerant LLC access in three phases.

1. *Issue:* Send access to both host and target blocks.
2. *Receive:* Wait until receiving both host and target blocks.
3. *Match:* Match both host and target blocks via a multiplexing layer to tolerate faulty

subblocks.

Figure 4.3 represents three stages of a fault-tolerant LLC access in our scheme. Since the proposed fault-tolerant cache architecture increases the number of accesses to the LLC banks, it has a direct impact not only on the network traffic but also on the LLC access latency. In fact, efficient implementation of a fault-tolerant cache architecture needs to consider its impact on both network traffic and latency while trying to minimize its performance and cost overheads. Considering such design tradeoffs, we propose four scalable policies to implement a fault-tolerant cache architecture based on a NoC paradigm.

We define an entity as a NoC component (core, router, or cache bank) that can perform a phase of the fault-tolerant LLC access. Based on this definition each one of the cores, routers, or cache banks can be an entity. Therefore, different phases of a fault-tolerant LLC access can be performed by different components (entities) in the NoC architecture. Based on the type and location of entities that perform different phases of a fault-tolerant LLC access, we can have different policies. However, in this architecture we assume all phases of a fault-tolerant LLC access are done in one entity (issue entity) and based on this assumption we define the four following policies.

1. **Core-Based:** This is the basic policy, where all three phases of a fault-tolerant LLC access are done by the core which sends an access to the LLC bank. In this policy, we don't perform any changes to the NoC architecture. The NoC architecture is used to send access to both host and target banks and return the host and target blocks to the issuer core. One advantage of this policy is that it potentially minimizes the network latency by sending access to both host and target block in parallel.
2. **Parallel-Issue:** In this policy, the local router of the core performs all three phases of issue, receive, and match. In this policy, we don't change the cores. Only the local routers of the cores are changed. One advantage of this policy is that only a section of routers have

to be modified. Also, it reduces the traffic between cores and NoC routers in comparison to the base policy.

3. **Host-Centric:** In this policy, the local router of the host bank performs all three phases of issue, receive, and match and then sends the final fault-free block back to the requester core. One advantage of this policy is that it minimizes the network traffic in comparison with other policies. However, its network latency highly depends on the distance between the host and target banks.
4. **Third-Node:** For this policy, an intermediary node as the third node is chosen to perform the three phases. The idea is to have a middle-way point that receives the request from the core and issues both requests to the host and target nodes in parallel. Eventually, host and target blocks will arrive at this same node. This would potentially decrease the amount of latency where both host and target blocks are concurrently accessed and route through the network in parallel; it does not serialize and lengthen the latency of the process as much as the host-centric policy. The method used to choose the intermediary node is explained in the next section.

Figure 4.4 represents different policies based on the issue entity and shows the differences between them.

### **Fault-Aware Router**

As mentioned earlier in previous Section, the router has the arbiter that reads from the input buffer in order to determine the direction to which the packet must go (namely, north, south, east, west, or local). Then it informs the crossbar which input buffer it must read from. This process is depicted in Figure 4.5(a). In order to support the fault-tolerant LLC access and the parallel-issue and host-centric policies presented in the previous section, we need to change the router. We therefore propose a novel fault-aware router to support our

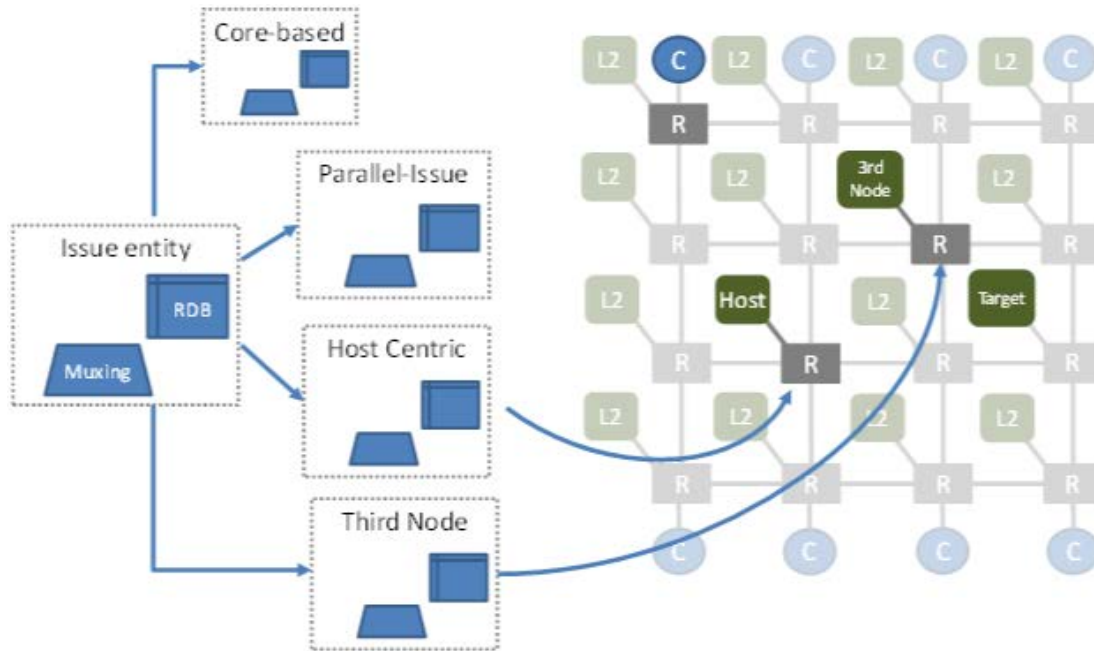


Figure 4.4: A general view of the NoC with different fault-tolerant mapping policies.

fault-tolerant LLC access via the NoC architecture, which is detailed in Figure 4.5(b). This new router has new components like an RDB, an extra buffer, and a multiplexing layer. In a situation where a data request arrives, the arbiter verifies (using the RDB) if that particular address is a faulty access (access to a faulty block). In case of a faulty access, it checks if the target block is local (within the same set) or remote (another set in another bank). In case of local target block, the cache bank handles the local remapping (more details in the next section). In case of a remote target block, the arbiter itself generates another request packet to be sent to the target bank. It keeps three pieces of information after that: (1) a flag that indicates that a matching operation is pending; (2) the direction from which the host and target blocks are expected; and (3) the address of the core that originally requested the data. When a reply packet arrives, the arbiter checks whether this is an expected packet, as part of the original data (host block) request that is pending. If so, it stores the block in the extra buffer. When the target data packet arrives, the arbiter sends it and also the contents of the extra buffer to the multiplexing layer for fault matching. The output of the multiplexing layer will be the final fault-free data packet (block) to be sent to the core that requested the



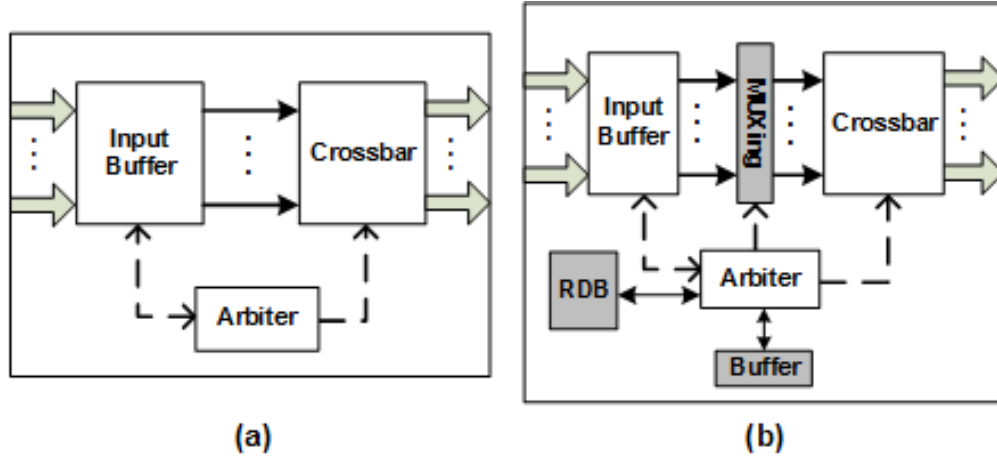


Figure 4.5: Router architecture: (a) Conventional router; (b) fault-aware router.

data. It is important to note that for regular non-faulty data requests, the multiplexing layer is bypassed. Note that based on different policies, we replace none, some, or all of the NoC routers with the new fault-aware one to implement the fault-tolerant method. In case of the core-based policy, there is no need to use the fault-aware router and all of the processes mentioned before are performed inside of the cores. For the parallel-issue policy, we replace all local routers of the cores (the routers directly connected to the cores) by the proposed fault-aware router. In case of host-centric and third node, since every router in the NoC can be an issue entity, we replace all of the routers by the fault-aware router. Additionally, for the third node, the first router (the one connected to the core) is responsible for choosing the intermediary node. This happens by generating two lists of routers from the local router of the core to the host router admitting the use of XY routing and YX routing (Figure 4.6(a)). In the same way, two additional lists are generated considering the target router as final destination (Figure 4.6(b)). Finally, the farthest router that intersects at least two of these paths is chosen as the third node (Figure 4.6(c)).

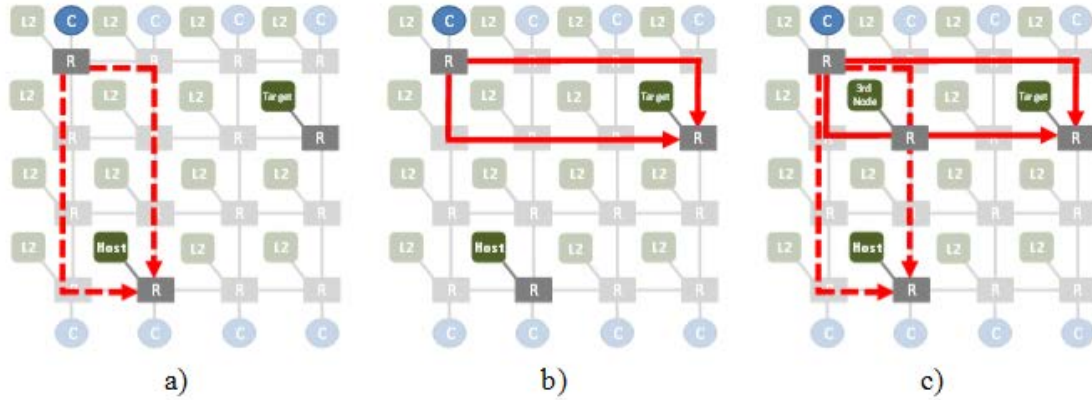


Figure 4.6: Choosing the intermediary node in the third node policy.

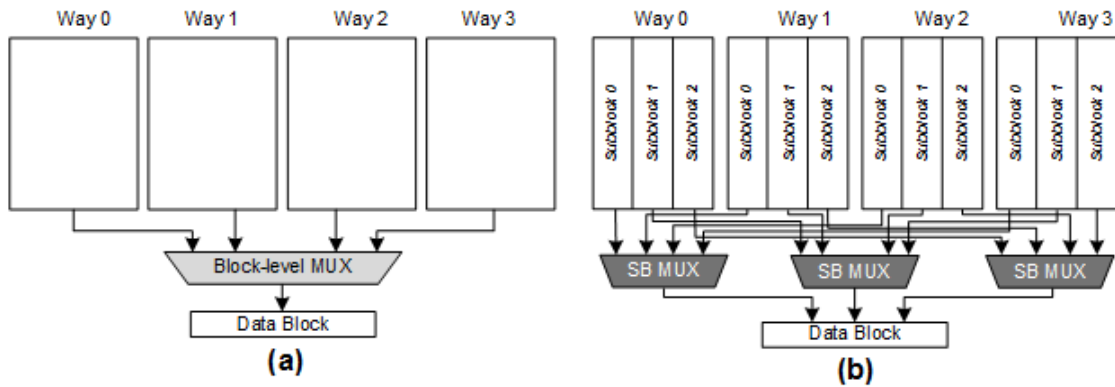


Figure 4.7: Cache bank architecture: (a) A conventional 4-way set-associative cache bank; (b) modified cache bank with three subblocks in each way.

### Fault-tolerant Cache Banks

We perform some changes in the LLC banks to support our fault-tolerant method. As mentioned earlier in Section 4.3.1, we divide each block into multiple, equally-sized subblocks. Since our LLC is a set-associative cache, each bank already has a large block-level multiplexer. We replace that multiplexer with a set of smaller ones with subblock size scale. Figure 4.7 shows a conventional 4-way set associative and the modified one with three subblocks in each way. The number of required multiplexers in each bank is equal to the number of subblocks per block.

Table 4.1: Simulation Configuration

|                    |  |
|--------------------|--|
| Processor Core     | 8 SPARC V8   |
| L1 I/D Cache       | 2 Banks, 32KB each, 4-Way, 32B Block Size, 2 cycle, LRU  |
| L2 Cache (LLC)     | shared 16MB, 16 banks, 16-way assoc, 64B lines, 10 cycle, LRU  |
| Coherence protocol | Directoy-based   |
| Memory Latency     | 250 cycles   |
| Interconnect       | NoC of 2D mesh (4×4 for 16 banks and 16 cores) , 32-byte links (2 flits per memory access), 1-cycle link latency, 2-cycle router, XY routing, Wormhole Switching, 4 phit buffer size |
| Frequency          | 1.0 GHz  |
| Technology         | 45nm   |

### 4.3.3 Evaluation

This section describes the methodology used to obtain relevant results in order to evaluate the impact of different policies of the proposed approach on performance, energy, and network metrics. In this work we have focused on the fault tolerance of on-chip memory banks using an ideal, fault-free NoC infrastructure.<sup>1</sup>

#### Simulation Setup

We use a cycle-accurate SystemC model of a NoC used in previous works [61, 91] which is a mesh-based implementation that uses an XY routing and wormhole switching with a 4-phit buffer size. Each router of this NoC is connected to a 1MB L2 cache bank. All L2 cache banks share the same address space of 16MB. Additionally, for each router in the first and last rows, there is a module generating data and instruction requests based on memory traces

---

<sup>1</sup>Of course the NoC itself also is vulnerable to both permanent and transient errors (and there is a lot of work on fault tolerance of the NoC fabric itself as described in the related work); for this evaluation we restrict our analysis to faults in the memory subsystem. Our future work will consider a unified approach that simultaneously analyzes the fault tolerance of NoC fabric together with the memory banks.

obtained from a Simics [102] simulation using 8 SparcV8 as cores. Table 4.1 summarizes the experimental setup of our architecture.

A workload of parallel programs with very distinct behaviors is created using benchmarks from the well-known SPLASH-2 [159], PARSEC [33], and a parallel version of MiBench [65] suites. Table presents a brief description for each one of them. Using the Simics simulator, we extract all memory requests (data and instructions) by executing these applications, one at a time, with 8 threads each. With these memory traces as input of our framework, we are able to extract performance and energy results. The performance results are the total number of cycles to execute all 8 threads and the energy results are obtained from Cacti 6.5 [116] for the memory subsystem and from Orion 2.0 [79] for the network-on-chip.

For the sake of this study, faulty LLC banks are modeled randomly since SRAM cell faults occur as random events. These random faults are due to the major contribution of the Random Dopant Fluctuation (RDF) to the process variation [2]. A recent work in the resilience roadmap reports the predicted probability of failure for inverters, latches, and SRAM cells as a function of the technology node [118]. Based on their results the probability of bit failure for SRAM cells can be up to  $2.6e-4$ . Here, we use the Predictive Technology Model (PTM) [13] that provides the voltage, area, power, and frequency scaling factors at 45nm technology node. We consider  $2e-4$  bit fault rate in our results which are based on the PTM model gains due to scaling the voltage down to 500mV.

## **Design Space Exploration**

We run the simulation process for our four proposed policies running different applications. Additionally, we consider a baseline case based on the original NoC with a fault-tolerant scheme similar to the one presented in Section 2.3. The restriction for this baseline is that there are no global targets. This means that the remapping for a faulty line will occur

Table 4.2: Benchmarks Description

| Benchmark Suite | Application   | Description  |
|-----------------|---------------|--|
| SPLASH          | FFT           | A complex 1-D version of a six-step FFT algorithm  |
|                 | LU            | Factors a dense matrix in the equivalent lower-upper matrix multiplication   |
| MiBench         | Patricia      | A data structure used to represent routing tables in network applications  |
|                 | Susan corners | An image recognition package. It was developed for recognizing corners and edges in Magnetic Resonance Images of the brain |
|                 | Susan edges   |  |
| Susan smoothing |               |  |
| PARSEC          | Swaptions     | Monte Carlo simulation is used to price a portfolio of swaptions   |

only by using blocks from the same bank. As a consequence, this baseline should have smaller opportunities for remapping, hence increasing the total number of disabled blocks in the system. The policy results are all normalized according to the baseline. We evaluate the impact of the proposed NoC-based fault-tolerant approach on performance, energy, and network latency and traffic.

We consider the network latency as the amount of cycles spent between issuing of the data request packet and receiving the data packet from the L2 cache bank. Network traffic, as defined in Section 4.3.2, is the amount of data that passes through the network taking into account the amount of time that the data spends passing through the NoC. The network traffic generated by each policy could create contention leading to performance loss. We can estimate network contention using the average buffer occupancy as a metric; in turn this gives us an estimate of the effect of network traffic generated by each policy. The average buffer occupancy increases as more packets are sent and also increases as more packets flow through more routers in the network. Our overall performance metric is the number of Instructions Per Cycle (IPC). An important observation is that since we are using a cycle-accurate SystemC model of a NoC, all contentions are already taken into account by the

simulator. The latency and traffic analysis shown in Section 4.3.2 did not consider the delay and energy consumption caused by eventual contention during the execution of the applications in the system. Thus, the results presented in the next section are much more precise in this matter.

Our energy metrics include the LLC total energy and the energy-delay product. The LLC total energy reflects the total amount of static and dynamic energy from the L2 cache banks and NoC (routers and links) throughout the execution of all traces. We also consider the energy overhead due to the extra traffic and also added blocks in the routers like muxing logic, RDB, and an extra buffer. The energy-delay product is the product between the energy and the total delay for each application. In this case, we consider the delay as amount of cycles taken to finish the execution of all tasks of the application.

### 4.3.4 Results

#### Network Latency and Traffic

Figure 4.8 shows network latency of the proposed policies normalized to the baseline across different benchmarks. As expected, due to the analysis presented in Section 4.3.2, the core-based and parallel-issue policies present very similar results. This happens because of the difference between these two policies in terms of traversing the NoC.

In the core-based policy, the packet must pass through one hop more than in the case of the parallel-issue policy. The host-centric policy, in contrast with our expectation, presents less latency than these two policies. Finally, the third node policy presents better results than the host-centric one for most benchmarks due to the use of an intermediary point between the core and the host and target nodes. This approach not only tries to decrease the time to access both blocks but also tends to minimize the traffic in the system since the target

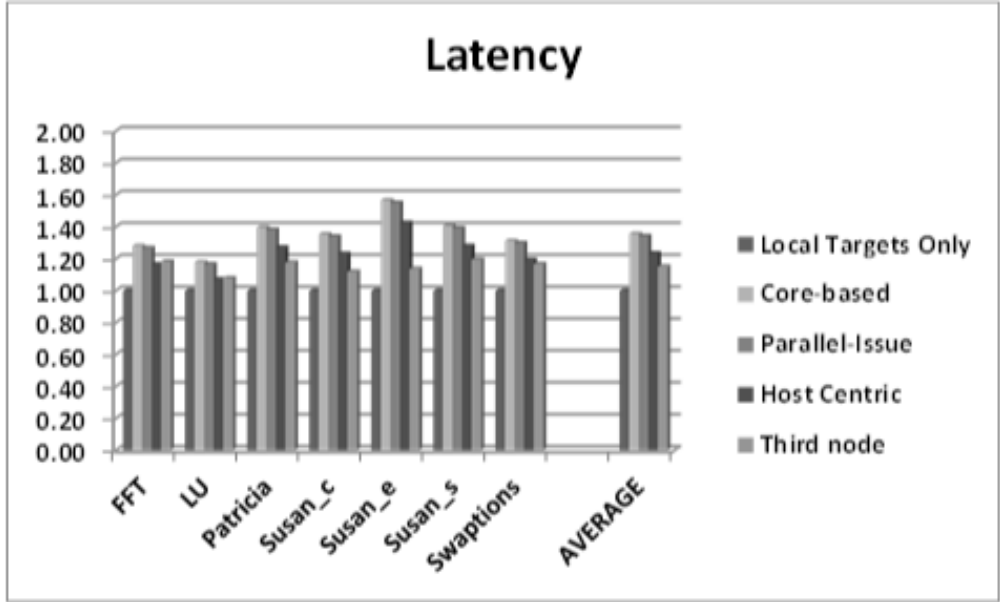


Figure 4.8: Network latency of different policies normalized to the baseline.

and host blocks do not have to travel all the way to the core. However, for the FFT and LU applications, the host-centric technique seems to have a slight advantage. This may be explained by the fact that these are smaller applications and therefore there is not much space for improvement by using the intuition behind the third node policy.

These results can be nonintuitive since the host-centric policy essentially requests the host and the target blocks sequentially, differently from the other two policies. However, it is important to note that the RDB configuration plays a key role in these results, especially for the host-centric policy. In this study, by managing the allocation of host lines and target lines in such a way that they be placed in cache banks only at most 1-hop apart each other, we create a situation where the serialization of requests in the host-centric policy does not affect the latency nor degrade the performance at all. Additionally as shown in Table 4.3, the RDB configuration generated for these experiments favors the host-centric policy since the majority of the target banks are 1-hop apart from their respective host banks. This means that because of XY routing in NoC, in most cases, cache access request and response packets in the core-based and parallel-issue policies flow through the network using almost

Table 4.3: RDB Remapping Results

|  |                |
|--|----------------|
| Global Targets                         | 17702 (54.02%) |
| Lines remapped in 1-hop banks          | 17092 (96.55%) |
| Average number of hops to target banks | 0.96           |

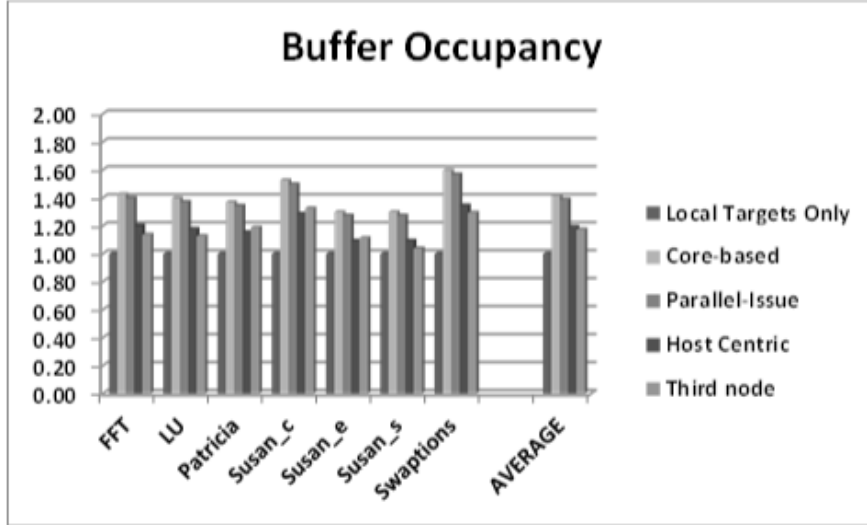


Figure 4.9: IPC of different policies normalized to the baseline.

the same set of routers as the host-centric policy. This directly affects the major advantage of these policies, which is the parallel use of the network.

In order to verify the impact of each policy on the NoC and the amount of contention they can generate, we measure the average buffer occupancy in routers. As presented in Figure 4.9 we consider the average amount of packets on each buffer in order to have an idea of the contention in the system. Therefore, the results show not only the amount of data packets in each case but also how long they stay in the NoC. In the end, these results show the potential to generate contentions that each policy may have.

Since core-based and parallel-issue policies only perform the multiplexing between the host and target lines in the last node and router of the return path, respectively, the network becomes busier and therefore they have higher average buffer occupancy. On the other hand, the host-centric policy matches the lines at the host node and, because of that, it causes less traffic and less possible contention in the network overall. Finally, for the third



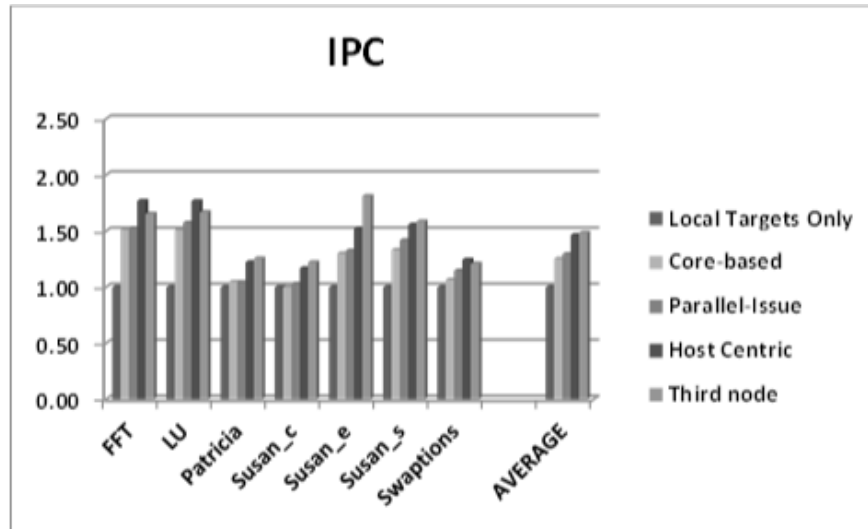


Figure 4.10: Network latency of different policies normalized to the baseline.

node policy, the results show that it presents similar results to the host-centric policy due to the fact that it tries to reduce the amount of usage of the NoC by performing the muxing phase earlier than the core-based and parallel-issue policies.

## Performance Results

Figure 4.10 represents the IPC results for different policies. It shows that with small variations in different applications, the third node policy has the best results on average but again at smaller applications the host-centric policy presents better results. Core-based and parallel-issue policies have almost the same IPC. This outcome is expected because of the similar trends observed in network latency and average buffer occupancy results.

As shown in the network latency results, some applications benefit more from the fault-tolerant method than others. This happens because some memory accesses to faulty lines do not necessarily generate L2 misses. Now some of these memory requests will generate L2 hits thanks to the remapping method. With that in mind, we can see that applications like FFT and LU have higher IPC, because they have lower network latency as shown in Figure

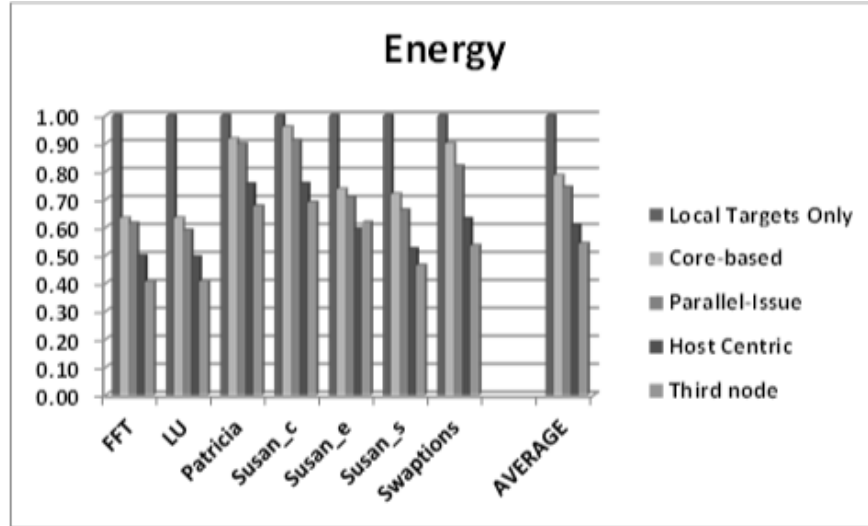


Figure 4.11: Energy of different policies normalized to the baseline.

4.8. Also, the amount of off-chip requests due to L2 misses harms the performance of the baseline more than the proposed policies.

## Energy Results

As mentioned earlier, the energy results for the cache banks and RDB arrays are obtained from Cacti. Also, we use Orion to get the energy results of NoC fabric considering extra added logic inside routers. Figure 4.11 shows the total LLC energy for different policies. As a direct consequence of less access to an off-chip memory, all policies present better energy results than the baseline. In the case of core-based and parallel-issue policies, this advantage is smaller than the host-centric policy because of the network energy. The fact that the host and target traverse through the network for more hops makes it more energy consuming in case of the core-based and parallel-issue policies, especially with more activity in their router buffers. Also, due to reasons explained before, the network latency for these two policies is higher, which affects their energy consumption as shown in Figure 4.11. For the same reasons, the energy consumption on the third node is much lower than the core-based and parallel-issue policies and therefore presents better results in most cases. Figure 4.12 shows

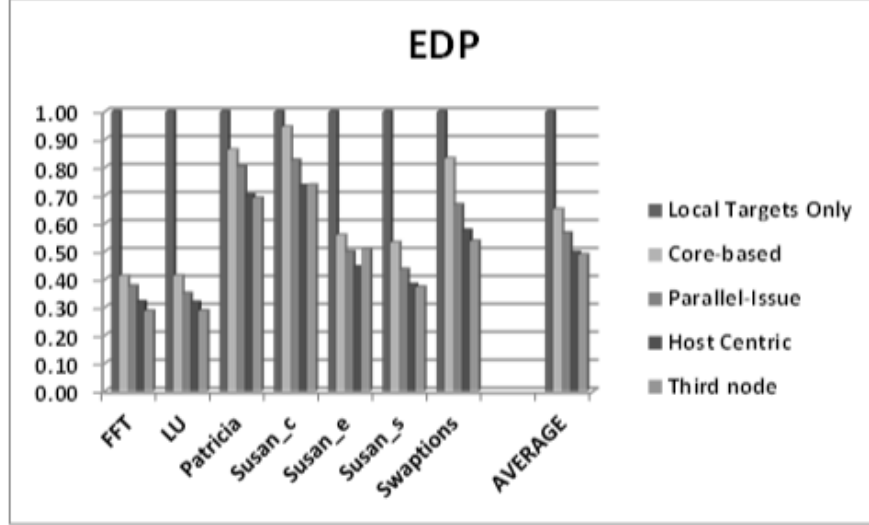


Figure 4.12: Energy-delay product of different polices normalized to the baseline.

Energy-Delay Product (EDP), which summarizes the results presented so far. The third node policy gains the best EDP because of its lesser amount of off-chip memory access, the lower amount of buffer occupancy, better performance, and lower energy in comparison with other policies. However, the host centric node also presents very similar results.

## Overheads

Since each policy requires some minor architectural changes, the solution presents some area and power overhead caused by the addition of extra components in the router as explained in Section 4.3.2. Table 4.4 summarizes the area and power overheads of the proposed architecture for different policies. We consider the overhead of the RDB, the extra buffer, the multiplexing layer, and miscellaneous logic in the issue entities. The RDB is the major contributor to area and static power overhead. In addition, in order to protect the RDB and the tag arrays, we use reliable 8T SRAM cell [38] which has about 33% area overhead for these relatively small arrays in comparison with 6T SRAM data cells.

In the case of the core-based, parallel-issue, and third node policies, each entity must have

Table 4.4: Power and Area Overhead Results

| Scheme        | Core-Based | Parallel-Issue | Host-Centric | Third-Node |
|---------------|------------|----------------|--------------|------------|
| Area          | 2.46%      | 2.46%          | 0.41%        | 2.46%      |
| Static Power  | 5.53%      | 5.53%          | 0.81%        | 5.53%      |
| Dynamic Power | 1.05%      | 1.05%          | 0.40%        | 1.05%      |

a full RDB (with entries for every line in the LLC). This is necessary since there is no limitation on which data request will arrive in the entity. Each core can access any line in any bank since they form the same address space. However, the host-centric policy is a different case. Since the issue entity is the router directly connected to the host bank, the data to be requested can only be one of the lines in that host bank. Therefore, the RDB in each router can have only the entries of the lines that the host bank holds. Hence, the overhead in all routers of the host-centric policy is equal to the size of one full RDB, while the overhead in the core-based and parallel-issue policies equals eight times (one per core) the size of a full RDB. Overall, the overhead of all policies in the proposed fault tolerance solution is minimal (less than 3% area and less than 7% power overhead), demonstrating the viability of our approach.

Regarding the delay overhead, we consider the delay to access the RDB inside the entity nodes router for both faulty and healthy lines. The delay overhead of muxing logic is negligible in comparison with that delay. However, this happens only once during each memory request and it takes only one extra cycle. From the gains presented in the IPC results we can safely assume that its overhead is compensated by the fact that the approach prevents more accesses to an off-chip memory which is very costly (typically a few hundred cycles).

## 4.4 CoDEC

### 4.4.1 Background and Motivation

#### Error Coding Overview

A common solution to address errors in memory components is applying Error Detecting Codes (EDC) and ECC techniques. Typically, EDC techniques are composed of parity bits, while the most common ECC methods employ Hamming [67] or Hsiao [73] codes with the capability of Single bit Error Correction and Double bit Error Detection (SECDED). More complex codes such as DECTED, QECPED [82] and also BCH [155] are considered once higher error detection is needed. However, they are rarely used in large cache memories because the area, delay, and power overheads of ECC grow rapidly as correction capability is increased [82]. Depending on the level of error detection/correction and complexity of implemented coding algorithm, there are multiple reliability, performance, power and area trade-offs [132].

For interconnected architectures using NoCs, if the data paths are infected by any faults, instantaneous fault detection can be provided by a combination of forward error correction (FEC) and error detection codes. ECC schemes can be implemented in the data link layer as s2s or in transport layer as e2e scheme [115]. In the s2s approach, the encoders (decoders) of the ECC are implemented in the output and/or input ports of the routers of the NoC, while in the e2e scheme, the encoders and decoders of the ECC are implemented in the network interface (NI) of the cores. When faults rates are low, robust s2s encoding schemes consume excess power which is unnecessary and can be avoided by e2e schemes. At the same time, s2s schemes can provide better fault coverage than e2e schemes when fault rates are high, especially in large hop networks. The choice of either e2e or s2s fault detection scheme in NoCs has power, area and performance trade-offs that vary with the number of CMPs and

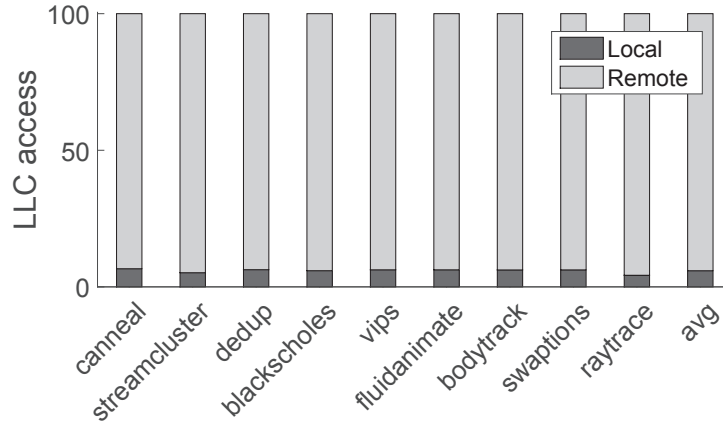


Figure 4.13: Percentage of LLC remote vs local accesses in a 16 MB shared LLC.

fault rate in the system. However, due to lack of scalability, the e2e scheme is inefficient in emerging large many-core platforms with high error rates.

## Motivation

In current multicores with large LLCs, the LLC access latency has a great impact on system performance, with many efforts to minimize the average cache access latency [75]. In such platforms LLCs are usually shared with non-uniform cache architecture (NUCA) [30]. A majority of the accesses to each LLC bank are global(remote) accesses transmitted through the network from non-local cores. Figure 4.13 shows the percentage of remote and local

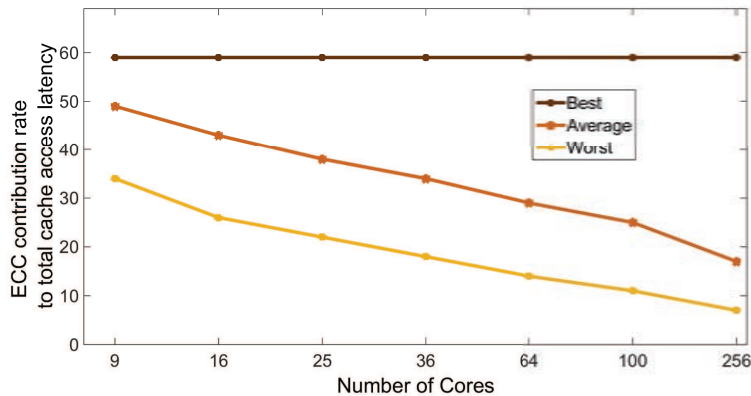
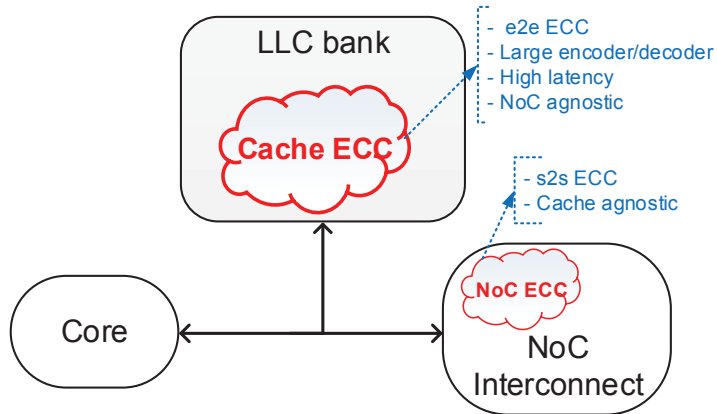
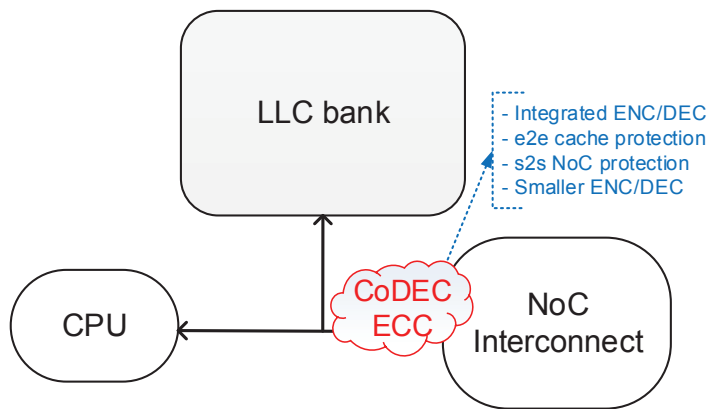


Figure 4.14: The ratio of ECC logic latency to total global LLC access Latency.



(a) Conventional approach



(b) CoDEC approach

Figure 4.15: Motivational example of LLC and NoC error coding integration

accesses to a 16-MB shared LLC with static NUCA when multithreaded workloads from the PARSEC suite are run on a 4x4 tiled CMPv (refer to Section 4.4.3 for the detailed experimental setup). As demonstrated in this figure, remote accesses to LLC banks through the interconnect are significantly more than the local ones. To see the effect of ECC latency on LLC access latency, Figure 4.14 depicts the latency contributions of the ECC logic in the LLC global access latency over different platforms. In this figure, the *Best* case considers the total LLC access time as the least global distance latency between the requester and the accessed LLC bank which is always one hop. For the *Average* case, the average hop count for each network size is considered as the distance between the requester and the accessed bank. For the *Worst* case, the network diameter of a mesh network with dimension order

routing (DOR) algorithm is considered. We can observe that a tangible amount of global LLC access latency is imposed by LLC cache ECC logic; especially if the requesting core is closer to the target LLC cache. Although the ECC latency contribution becomes smaller in larger platform, it is still non-trivial, e.g., almost 10% for the largest platform with 256 cores. The observation from both Figures 4.13 and 4.14 demonstrates that there is a great opportunity to improve the performance of the system by reducing the latency of the ECC logic which is in the critical path of each LLC access.

Figure 4.15 depicts a cartoon demonstrating a conventional versus a codesign approach to error coding in both LLC banks and NoC interconnect. Since the ECC encoder/decoder blocks of LLC bank and NoC have redundancy overlaps, they can be integrated together with smaller overheads in the codesign approach as shown in Figure 4.15(b). The common error coding scheme in current multicore platforms is the independent integration of error protection in both cache banks and NoC fabric as shown in Figure 4.15(a). We consider this scheme as our conventional approach and will explain it in more detail in next Section. This approach is not applicable to large emerging manycore platforms because of the high power/performance overhead due to unnecessary ECC logic redundancy. Since the ECC encoder/decoder blocks for conventional fault-tolerant LLCs are in the critical path of each LLC bank access [82], a significant number of global LLC accesses in manycores will impose high latency and energy overhead to the system (refer to Section 4.4.4 for more details). Moreover, using stronger ECC to protect against multi-bit errors magnifies such overheads. On the other hand, the choice of using different ECC schemes in both cache and NoC has a large design space with different power consumption, area, and performance trade-offs. To address these limitations, we propose a new system-level integrated error coding approach that co-designs error coding jointly for the LLC and NoC as depicted in Figure 4.15(b).



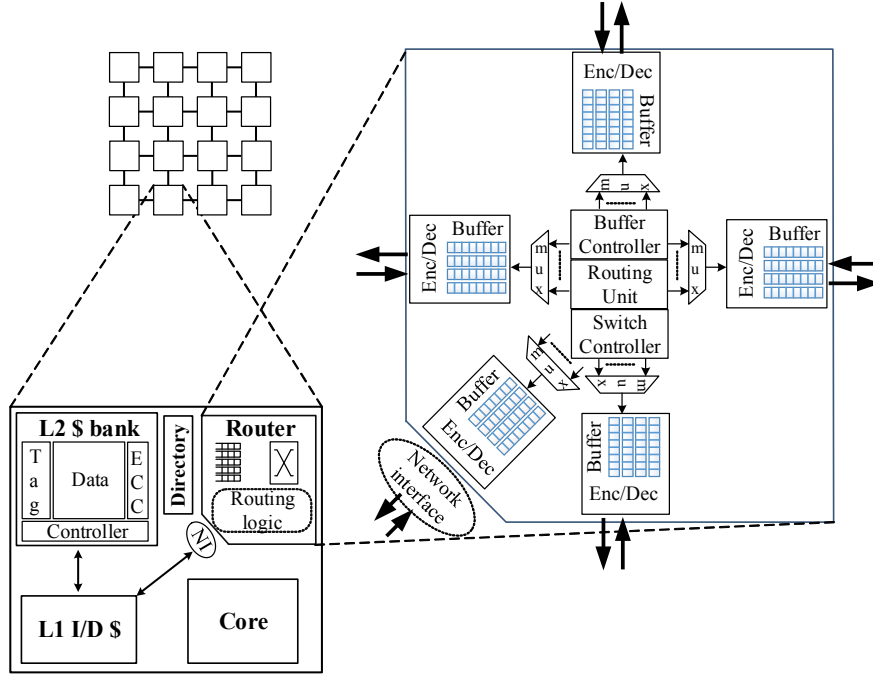


Figure 4.16: A multi/many-core architecture with a mesh NoC interconnect.

#### 4.4.2 CoDEC Overview

In this section, we introduce the target platform and the conventional error coding approach that we consider in our experimental results. In the conventional approach, we apply separate ECCs across LLC banks and NoC interconnect to protect global LLC accesses, passing through the NoC, against unexpected errors. On the other hand, our CoDEC approach combines the error coding of cache side and NoC side together. In contrast with the conventional approach, we replace the cache line-level ECC extension with segmented flit-level ECC. In this case, there is no need to have encoder/decoder blocks on the path between LLC bank and NI. Finally, we present design space exploration of the proposed approach parameters and possible design trade-offs.

## Target Platform

To illustrate our approach, we experiment on an exemplar tiled NoC-based multi/many-core architecture, where each tile comprises a processor core, private L1 data and instruction caches, a shared L2 cache (LLC) bank, network interface (NI), and router. Figure 4.16 shows our target 16-core architecture with a 4x4 mesh NoC interconnect. Based on our cache organization, each L2 bank is a portion of the larger distributed shared LLC. A MESI protocol is implemented in order to maintain cache coherence. The target design assumes a NUCA LLC [81]. In general, with multiple banks within the NUCA LLC, we have the choice of either always putting a block into a designated bank (static mapping) or allowing a block to reside in one of multiple banks (dynamic mapping). In static mapping, a fixed hash function uses the lower bits of a block address to select the target bank. In our platform we consider static mapping called static NUCA (SNUCA) [30]. SNUCA statically partitions the address space across cache banks connected via the NoC. In this architecture, the LLC access latency is proportional to the distance from the issuing L1 cache to the target LLC bank.

## Conventional Approach

Figure 4.17(a) shows the architecture of a conventional approach with integrated NoC-LLC error coding that leverages multi-bit ECC for LLC protection and s2s ECC for interconnect. On the cache side, each cache block is extended with an ECC, and every read or write requires error detection/correction coding, respectively. On the NoC side, each flit is extended with s2s ECC to protect data packets in transport layer against errors over the interconnect fabric including links and network routers. As shown in Figure 4.16, an encoder/decoder is applied on each port of all routers in the NoC. For LLC banks we consider a block-level multi-bit ECC which is a 2-bit error correction scheme (DECTED), that is common for fault-tolerant cache

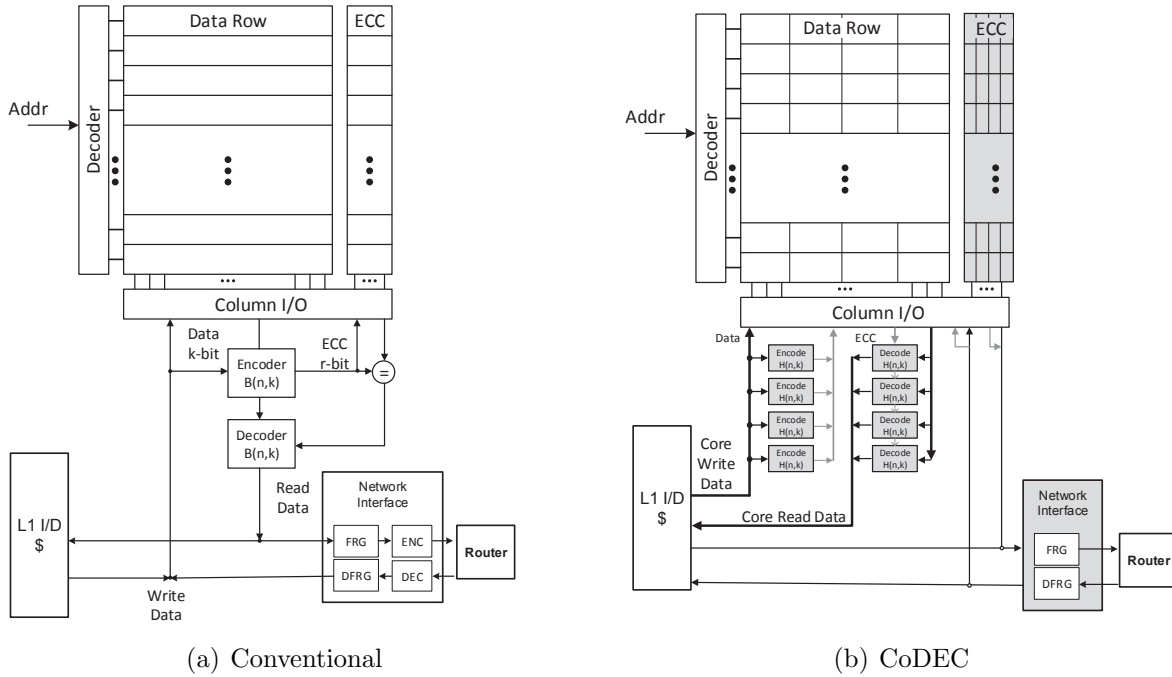


Figure 4.17: The architecture of an ECC-protected LLC bank in the Conventional and CoDEC approaches.

designs [132]. As depicted in Figure 4.17(a) each bank has its own encoder/decoder (shown as ENC/DEC) that are in the critical path of both core (local) and network (global) accesses. Consequently, for each LLC access, no matter whether it is local or over the network, the accessed cache block goes through encoder/decoder blocks. Since in shared NUCA LLCs most of the accesses to LLC banks are from non-local cores [75](as depicted in Figure 4.13), the cache blocks need to be protected over the network too. Hence, we leverage s2s ECC over the whole NoC and modify the NI and routers to include encoder/decoder blocks in each port, as shown in Figure 4.17(a). Figure 4.18(a) shows an example to demonstrate how a cache block is protected from the cache through the network in conventional approach. In this example, a 64-byte cache block extended with 40 ECC bits, is fragmented into four flits via the NI. Each flit is also protected with 8 ECC bits. On each LLC access, both the cache block and its corresponding ECC bits (40 bits in our example) are read and then the cache block is transferred through the cache decoder block for possible error detection/correction and then passed to the NI to be fragmented to multiple flits. Each flit will be encoded by

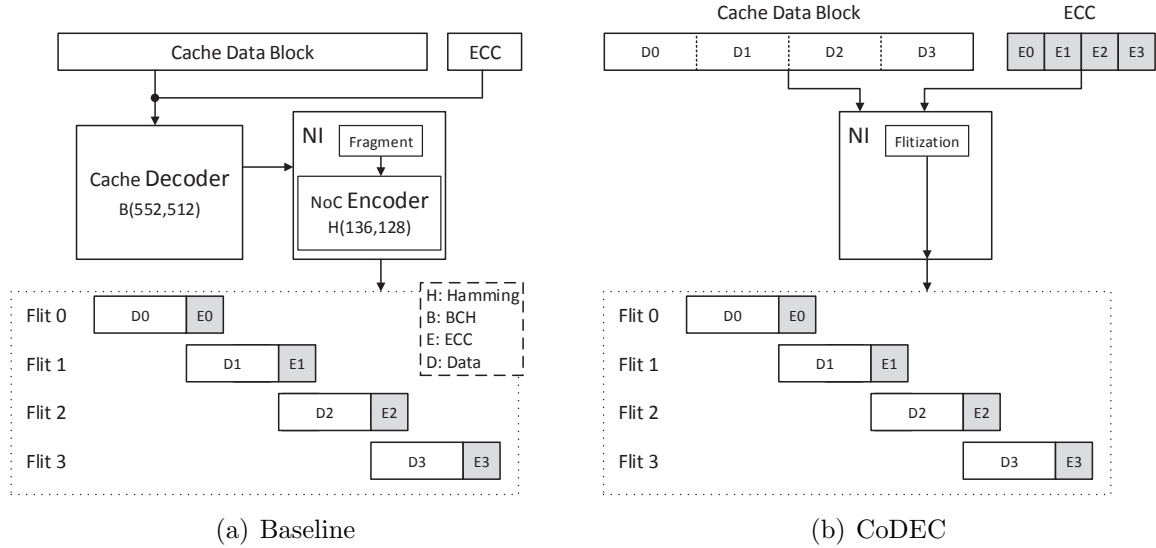


Figure 4.18: Data flow of LLC block through the network in Conventional vs. CoDEC approach.

the encoder inside NI and then transmitted to the router.

### Proposed CoDEC Approach

In *CoDEC*, we propose a Codesign of cache and interconnect error coding using segmented multi-bit ECC in LLC banks and s2s ECC in NoC. Using this approach, we reduce the latency and power of shared LLC protection while meeting the same level of reliability compared to conventional approaches. *CoDEC* replaces the block-level strong ECC (i.e., DECTED) in LLC banks with a segmented low-cost ECC (i.e., SECDDED) and sends the ECC bit chunks along the block through the network as represented in Figure 4.17(b). Using this approach, *CoDEC* removes the large encoder/decoder of cache banks from the critical path of LLC access via the network. Compared to the conventional approach, *CoDEC* incurs a small ECC storage overhead.

As depicted in Figure 4.17(b), LLC controller and NI blocks inside each tile need some modification to implement the *CoDEC*. Note that the modified components are highlighted

in this figure. In LLC banks, the data array needs to store all flit-level ECC bits alongside each cache line. The large cache-level encoder/decoder blocks are replaced by multiple parallel, but smaller ones and only on the path between LLC and core. Furthermore, there is no large cache-level encoder/decoder blocks in the critical path of network access as shown in Figure 4.17(b). The encoder/decoder blocks inside NI are also removed and its internal logic is modified to pass the ECC bits of each cache segment during flit generation process. Figure 4.18(b) represents the fragmentation process of each cache block and construction of data flits in the proposed architecture.

Figure 4.19 illustrates a complete data path of CoDEC design in interconnected many-core architecture with distributed shared LLC banks from network-level point of view. This figure provides an example to show how CoDEC guarantees end-to-end protection of LLC data blocks from source to destination over the network using s2s error protection capability over the interconnect. In summary, each cache data block is virtually divided into smaller segments as shown in Figure 4.18(b). Then, each of these partitions plus the generated ECC for each segment are passed through the NoC as a flit as shown in Figure 4.19. The figure also shows how a cache block becomes faulty through the network and how it can be fixed via different routers in the path.

### 4.4.3 Evaluation

In this section, after introducing the simulation setup, we discuss the effects of various design parameters and possible trade-offs in the design of CoDEC approach.

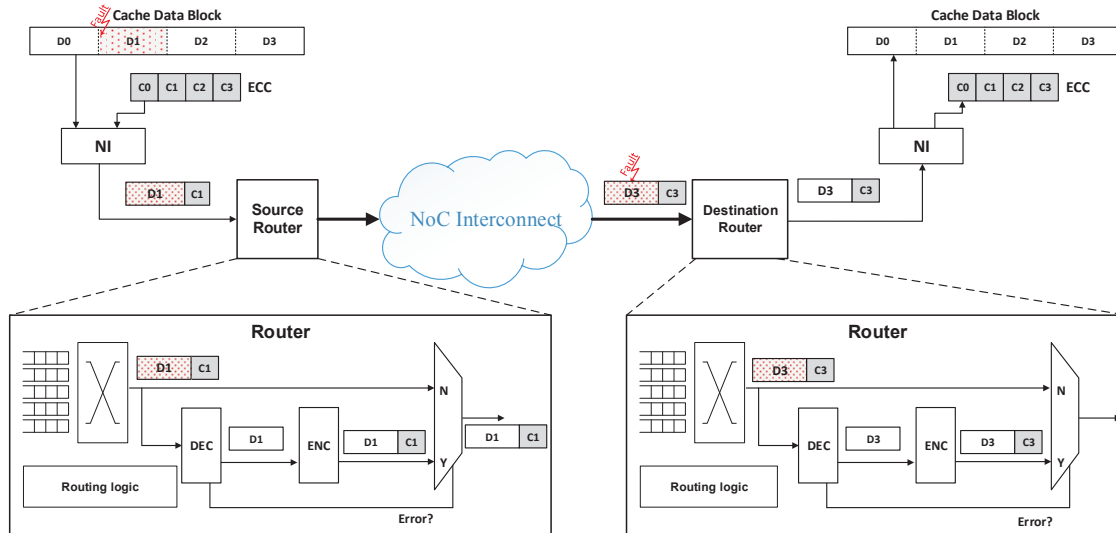


Figure 4.19: CoDEC LLC data protection over the network from source to destination.

## Experimental Setup

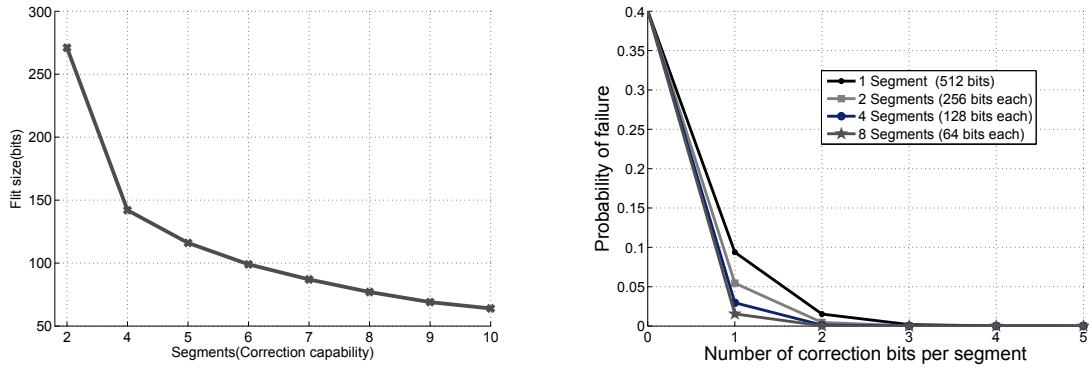
We leverage Sniper [40], a parallel, high-speed, and cycle-level x86 multicore simulator to evaluate the performance impact of our approach. Nine benchmarks from the PARSEC suite are selected and run on the modified simulator. The details of the simulation setup are listed in Table 4.5. Also, the RTL model of both conventional and CoDEC routers are implemented in Verilog to evaluate the implementation overheads. The routers contain eight-flit buffers for each ports. The accuracy of the NoC model is verified by tracing the packet routes through the network with the help of SystemVerilog language for Splash traffic patterns. Synopsys Design Compiler with TSMC 32nm standard cell library is used to synthesize and report accurate timing and area overhead of routers including encoder/decoder (ENC/DEC) blocks of both conventional and CoDEC designs. Power consumption is also measured by performing post-synthesis gate-level simulation with Synopsys PrimeTime. Each router is designed to operate at its maximum frequency with the assumption that processor tiles operate on their own frequency domain asynchronously with respect to the interconnection network, which is popular in industrial many-core chips [151]. CACTI 6.5 [116] is used to evaluate area, power and delay of LLC banks and their related components.

Table 4.5: Simulation Configuration

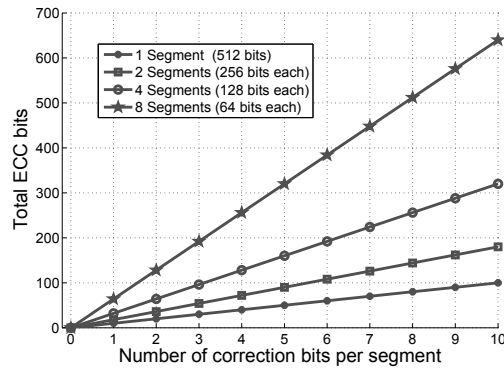
|                    |  |
|--------------------|--|
| Core               | 2.66 GHz, 4-way issue, 128-entry ROB   |
| L1 I/D Cache       | 32kB split I/D, 4-way assoc, 64B lines, 4 cycle, LRU   |
| L2 Cache           | shared 16MB, 16 banks, 16-way assoc, 64B lines, 8 cycle, LRU                                       |
| Coherence protocol | MESI   |
| Interconnect       | NoC of 2D mesh (4×4 for 16 banks and 16 cores) , 1-cycle link latency, pipeline router, XY routing |
| Technology         | 32nm   |

### Design Space Exploration

Each cache block entry is supposed to be divided into a fixed number of segments. The size of each NoC flit is chosen equal to the segment size in CoDEC design. We assume a fixed number of flits are dedicated for each buffer component of a NoC design; larger flits need bigger buffers. Flit size is a tuning parameter in choosing the segment size (number of segments) for the CoDEC approach, which is the major configuration parameter in CoDEC design. Different flit sizes are compared in Figure 4.20(a) in terms of various segment numbers. A 512-bit cache block is considered for this experiment. As expected, smaller flits are needed by dividing each cache block entry into smaller segments. On the other hand, as the number of segments increases higher ECC capability is provided by imposing more redundancy bits inside the cache. Note that more complex encoder/decoder blocks are needed in CoDEC for larger buffer components, resulting in higher power consumption, area overhead, and latency. Since the flit size of 64-128 bits is accepted by the NoC community [124], a cache block with 4 to 9 segments is acceptable for CoDEC design based on Figure 4.20(a). However, as the number of segments increases (flit size decreases), the whole packet is transmitted with higher delay. So, the minimum number of segments in the accepted range of NoC bandwidth is chosen for this experiment to be 4.



(a) Impact of cache block segmenting on NoC flit size (b) Failure probability of conventional and proposed methods in terms of bit error correction capability



(c) ECC bits overhead of conventional and proposed methods in terms of bit error correction capability

Figure 4.20: Impact of segment size on conventional versus CoDEC design.

## Reliability analysis

This section discusses the trade-off between overhead and reliability coverage in the proposed CoDEC method. The probability of a single bit error occurrence in a memory device with  $n$  bits is represented by Equation 4.1.

$$Q = \frac{FIT}{n} \quad (4.1)$$



where FIT refers to failure in time error/billion device hours. Typical FIT rates for latches and SRAM cells at sea level vary between 0.01 and 0.001 FIT/bit [148]. Recent development in cache physical fabrication has resolved the correlated multiple-bit error issue. Interleaving approach has been used in SRAM/DRAM cells in [146] to generate logical check bits from physically dispersed areas of the memory array. With this solution, the multi-cell upset will lead to multiple single-bit errors, rather than a multi-bit error. Consequently, in this work we assume that the probability of a single bit error is independent of any other bits in a cache block; the probability of  $E$  errors present in a cache block is calculated by Equation 4.2, the error rate uniform throughout the cache:

$$P(E) = \binom{N}{E} q^E (1 - q)^{N-E} \quad (4.2)$$

where  $N$  represents the number of bits in cache block and  $q$  is the probability of single soft error occurrence in a single bit ( $Q$  for  $n=1$ ). Therefore the probability of exactly a single bit error in a cache block is extracted from Equation 4.1 and 4.2 is shown as Equation 4.3.

$$P(E = 1) = N(FIT)(1 - FIT)^{N-1} \quad (4.3)$$

The robustness probability of a fault-tolerant cache block with at least  $E_{cov}$  bit error correction capability is shown in Equation 4.4.

$$\sum_{E=0}^{E_{cov}} P(E) \quad (4.4)$$

However, the failure probability of a fault-tolerant cache block with at least  $E_{cov}$  bit error correction capability in segmented ECC method depends on the number of segments as

shown in Equation 4.5.

$$1 - \left( \sum_{E=0}^{E_{cov}} P(E) \right)^S \quad (4.5)$$

where  $S$  represents the segment numbers. The partitioning process is tuned by the flit size definition as discussed earlier. The failure probability and ECC overhead bits of conventional and proposed methods are reported in terms of error correction capability per segment in Figures 4.20(b) and 4.20(c). Choosing a reasonable number of segments depends on the flit size limitation with similar failure probability to the conventional method. Larger flit size imposes higher data bandwidth of NoC designs while smaller flit sizes increases the latency of the packet transmission through the network. Furthermore, according to the accepted range of NoC bandwidth higher ECC bit correction in the proposed method needs more ECC bits as shown in Figure 4.20(c). The failure probability of conventional method with  $E_{cov} = 2$  has the most equivalent failure probability of proposed method with  $E_{cov} = 1$  and 4 segments for smaller number of ECC bits as shown in Figure 4.20(b). So, each cache block entry is divided into 4 segments in our proposed method with  $E_{cov} = 1$ .

#### 4.4.4 Experimental Results

In this section, at first, the impact of CoDEC on system performance is presented. Then, the power consumption and area overhead of different design configurations are compared separately for both cache and NoC. The LLC global and local access time is also reported to show the improvement of the CoDEC method over baseline. We compare the results of proposed CoDEC with two common single-error correction double-error detection (SECDED) and double-error correction triple-error detection (DECTED) approaches [43]. We also consider a Non-Fault-Tolerant (Non-FT) design without any error protection as a reference to compare the overheads of different approaches.

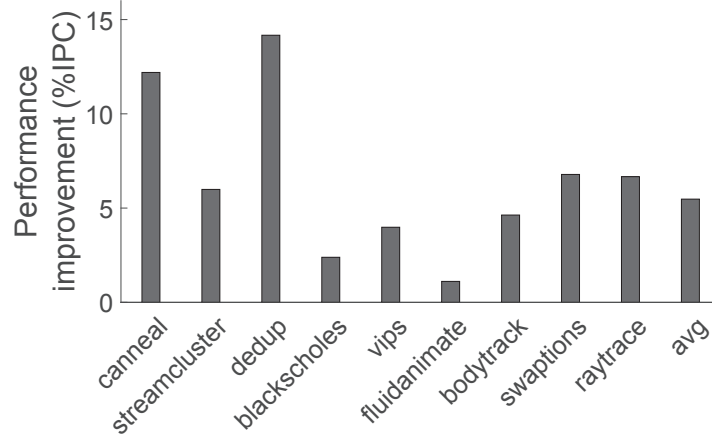


Figure 4.21: Performance (IPC) improvement of a system with CoDEC LLC, normalized to the baseline.

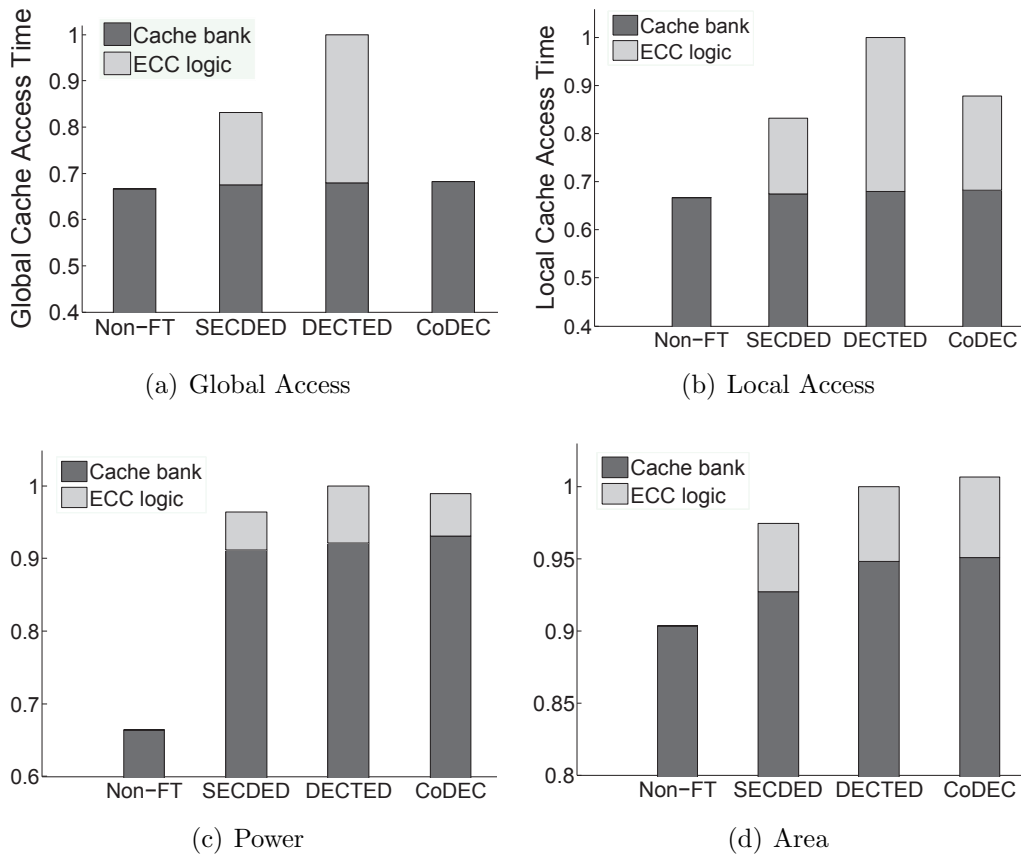


Figure 4.22: Cache experimental results normalized to DECTED results.

## Performance Results

We evaluate the performance impact by measuring the aggregate user instructions committed per cycle (IPC). Figure 4.21 shows the IPC improvement of CoDEC normalized to the IPC

of the conventional approach (refer to Section 4.4.2) running nine PARSEC benchmarks. The system with CoDEC experiences up to 14% and on average 5.5% improvement in the performance compared to the system with conventional LLC protection approach.

## Cache Results

Figure 4.22 depicts the latency, power, and area breakdown of different LLC protection approaches. Note that the reported values are normalized to DECTED which is considered as the conventional approach. In this figure, Global access time of each cache bank (dark bar) is the access time from the router to the LLC data array. Since CoDEC removes the ENC/DEC blocks from the critical path of LLC bank Global access, it has almost the same latency as NonFT as depicted in Figure 4.17. The latency of ECC logic in SECDED and DECTED (shown in light gray color) are added to the latency of cache data array, resulting in slower cache accesses. Although the local cache access latency of CoDEC method is higher than both of Non-FT and SECDED methods, it is still less than the conventional (DECTED) approach as shown in Figure 4.22(b). This is because larger error control block are replaced by multiple smaller ones on local cache access path. However, as discussed earlier in Section 4.4.1, the percentage of global cache access are much more than local cache access justifying the negligible latency overhead of CoDEC method comparing to the other approaches. The power consumption of the cache bank for all of the fault-tolerant approaches are higher than Non-FT method. But the power consumption of CoDEC techniques is less than DECTED while it supports more protections as shown in Figure 4.22(c). This is because smaller encoders have fewer inputs with simpler gate-level implementations. Note that the area overhead of CoDEC method is negligibly ( $< 1\%$ ) higher than the conventional approach since it needs to store more number of EEC bits to support a better protection, as depicted in Figure 4.22(d).

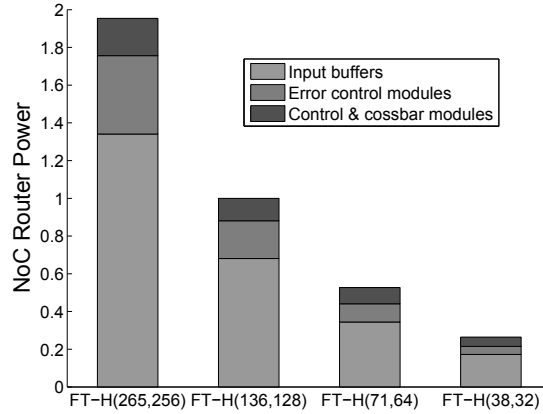


Figure 4.23: Power consumption of NoC router’s major blocks for various ECC sizes

## NoC Results

Figures 4.23 and 4.24 demonstrate the power and area of fault-tolerant routers in terms of different flit sizes while each flit is protected by the help of Hamming code. The number of EEC bits varies according to the flit sizes. Data flit sizes in range of 32 -256 bits are examined in this experiment which are shown through horizontal axis of Figures 4.23 and 4.24. All the reported values are normalized to the absolute value of CoDEC method with 4 segments. For a 512-bit cache block and 4 segments in CoDEC method, each flit is composed of 128 bits with 8 extra bits as shown by FT-H(136,128) term in Figures 4.23 and 4.24. Flit size and consequently the bandwidth of each port of routers (assuming equal flit and phit sizes), are determined by adding the extra redundant EEC bit to the actual data bits. For example the NoC router bandwidth for FT-H(136,128) method is 136. Power consumption and area overhead of NoC routers follow a descending trend by choosing smaller flit sizes. This is because smaller flit sizes need lower bandwidth and smaller buffer sizes.

Finally, Table 4.6 summarizes the comparison of SECDED, DECTED, and CoDEC methods. In this table, the overhead and improvement values are normalized to the Non-FT values. Note that the results in last column represent the reliability improvement with respect to the Non-FT which has not any error protection. Comparing CoDEC to DECTED, the local

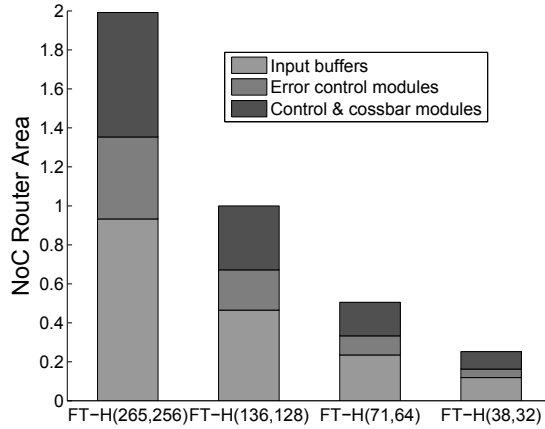


Figure 4.24: Area of NoC router's major blocks for various ECC sizes

Table 4.6: Overhead and improvements comparison.

| Methods             | Overheads (%) |               |             |               |             |             | Reliability Improvement (%) |
|---------------------|---------------|---------------|-------------|---------------|-------------|-------------|-----------------------------|
|                     | Local access  | Global access | Area        | Dynamic Power | Total Power | ECC Storage |                             |
| Non-FT              | 0             | 0             | 0           | 0             | 0           | 0           | 0                           |
| SECODED             | 24.8          | 24.8          | 7.9         | 45.2          | 16.5        | 2.1         | 77                          |
| DECTED <sup>2</sup> | 50            | 50            | 10.7        | 50.6          | 19.9        | 3.9         | 96                          |
| <b>CoDEC</b>        | <b>31.6</b>   | <b>2.4</b>    | <b>11.4</b> | <b>49.0</b>   | <b>19.7</b> | <b>6.3</b>  | <b>93</b>                   |

and global cache access time latencies are reduced from 50% to 31.6% and from 50% to 2.4%, respectively. While the CoDEC design imposes a minimal overhead (less than 1% area and 2.5% ECC storage overhead) over DECTED, it has almost similar power consumption and protection capability. Note that except Non-FT approach, for other approaches we consider a s2s single bit ECC to protect interconnect NoC.

## 4.5 Summary

In this chapter, we proposed two interconnect-aware approaches to error-resilient design of LLC in NoC-based multi/many-core platforms which are orthogonal to all previous efforts in fault-tolerant cache design as well as reliable NoC design.

First, we proposed a novel approach for efficient implementation of fault-tolerant design of NUCA LLCs in CMP architectures. Our solution is based on modular extension of NoC routers to support fault-tolerant schemes that also scale with increasing CMP sizes. We proposed four different policies for implementing a remapping-based fault-tolerant scheme leveraging different configurations of NoC. Our experimental results on an 8-core CMP with 16 shared LLC banks show that leveraging the on-chip interconnect fabric in design and implementation of the fault-tolerant scheme is an efficient and scalable solution that comes with a small overhead (less than 3% area and less than 7% power overhead). We studied the effects of four different fault-tolerant policies on the NoC, and showed that the parallelism of communication directly affects the results. In order to explore the impact of different parameters of a NoC we also presented results showing that some of these policies are more positively affected by some architectural changes like the number of virtual channels and adaptive routing. More virtual channels improve some of the policies that had previously presented worst because of a particularly bad use of the communication parallelism. However, results also show that this improvement reaches a saturation point very quickly. As for the experiments with an adaptive routing, results show that policies that created more contention now perform particularly better. Also, it is very important to consider the effect of a fault-tolerant method configuration on NoC parameters. In the cases presented here, we perform exploration on the distance between the host and target LLC banks in the NoC. Our experiments show that enabling the target blocks to be further away from the host blocks does not decrease much the number of disabled blocks. Also, we showed that there is a sweet spot where the overall IPC can be improved.

As a second contribution, we proposed CoDEC, a novel codesign error coding of cache and interconnect for many-core architectures to improve the performance by reducing cache access latency with minimal area and robustness impact. CoDEC guarantees the e2e protection of LLC data blocks by integrating strong multi-bit ECC in cache banks to s2s error coding of interconnect, providing a unified system-wide ECC policy. Our experimental results showed that with respect to the conventional DECTED approach, CoDEC improves LLC access time for global and local accesses as 50% and 20% respectively, that leads up to 14% performance improvement, with a negligible overhead of only 1% area and 2.5% more ECC storage. Potential directions for future work include adaptive interconnect error coding, improvements to the proposed architecture, a broader design space exploration involving different system configurations and coding algorithms.



# Chapter 5

## Relaxing Error Resiliency in On-chip Memories

### 5.1 Introduction

As the semiconductor industry continues to push the limits of sub-micron technology, the ITRS expects hardware variations to continue increasing over the next decade [157]. The power consumption of the memory subsystem, which is a major contributor to the overall power consumption of the system, is very vulnerable to the effects of variations. Device manufacturers have masked the presence of variability by expensive over-design and “guard-banding” in order to guarantee data integrity. However, 100% correctness may not be required for all applications. This chapter proposes the idea of relaxing or reducing the typical memory guard-banding for systems running applications that can tolerate some level of error in their computations. We refer to such class of memories with relaxed guard-banding as Partially-Forgetful Memories (PFMs). Potential advantages of deploying PFMs include: (i) Better energy efficiency for memory accesses, (ii) Increased lifetime of the memory subsys-

tem, and (iii) Reduced memory access latency. In the following, we overview some of the overheads of guard-banding in memories and briefly mention the application domains that can benefit from PFMs.

### 5.1.1 Cost of Guard-banding

Guard-banding leads to over-design with suboptimal power and/or lifetime for different memory technologies. SRAM studies [153] show that masking all variabilities in 90nm SRAM requires an increase in data retention voltage from nominal value of 0.35V to 0.7V, resulting in higher power consumption. Similarly, DRAM studies [99] show that  $10^3$  leaky cells in a 32GB DRAM module force the use of 64ms refresh interval while other cells can work with a refresh interval that is four times longer, increasing refresh power overhead. Likewise, studies on PCM [167] report a 96% increase in the programming power and 50X endurance degradation due to process variation.

### 5.1.2 Approximate Computing

Recently, researchers [134, 92, 50, 58, 136] have shown that a system can save energy by exposing faults to a variety of applications that are resilient to hardware errors during their execution [51]. In particular, embedded, multimedia and Recognition, Mining and Synthesis (RMS) applications could still process information usefully with error-prone elements [45]. This motivates the opportunity to deploy PFMs in systems executing approximate computing applications.

### 5.1.3 Challenges for Effective Use of PFMs:

The following challenges need to be addressed to effectively exploit PFMs for approximate computing:

1. *Data Partitioning* Data structures can be defined as critical (i.e., any corruption leads to catastrophic failure or significant quality degradation) or non-critical (i.e., quality degradation resulting from corruption may be acceptable). This concept has been used before in [100]. A simple partitioning of data – into critical and non-critical – has been shown to improve DRAM energy efficiency [100]. Further categorization of the non-critical data can lead to even more energy efficiency. For example: (i) integer data can be partitioned based on the magnitude of tolerable error, and (ii) floating point data can be partitioned based on a limit on precision. These categorizations can be reflected in the program by annotating different data structures. One approach [134][39] defines type qualifiers and dedicated assembly-level store and load instructions for this purpose. We present another approach for PFMs that uses dynamic declarations which are enforced by a run-time system, as shown by an example in Section 5.3.
2. *Data Mapping:* The data partitioning annotations in the previous step should guide mapping of each data category to an appropriate part of memory based on its reliability characteristics. Depending on the type of memory, this mapping can be done by the hardware, application/compiler, or operating system. For caches, the cache controller decides where to map a new incoming cache block. In case of software-controlled memories (e.g., scratchpads), the compiler aggregates data with the same reliability requirement in tagged groups, and then a run-time system (having knowledge about underlying hardware) does the actual mapping based on this tagging. For main memories, the operating system should do the mapping during logical-to-physical mapping.

3. *Controlling Exposed Error Rate to Program:* Most of the previous attempts at utilizing relaxed-reliability memories [100][135] lack the ability to dynamically adjust the exposed error-rate to the program. A recent work [150], has shown how the DRAM timing margin can be dynamically changed to make a precision-performance trade-off. We believe this is necessary because different applications have different levels of tolerance to errors. Even within an application, one execution phase may have more tolerance to errors (i.e., less criticality) than another phase. Therefore, it is important to enable adjustment of the guard-banding knobs based on the characteristics of application. Examples of these knobs are: DRAM Refresh Rate, SRAM Voltage, STT-RAM Retention Time,  $I_{RESET}$  Current in PCM, etc.
  
4. *Adaptation to Phasic Behavior of Applications:* There are three main reasons to change memory guard-banding knobs during run-time: (i) Depending on the (set of) applications that are executed, we might tolerate different levels of error, (ii) We may need to change the ratio of reliable to unreliable memory capacity to prevent performance degradations for specific situations (e.g., where most of the working set objects should be mapped to reliable parts of memory), and (iii) The programmer can save power by disabling idle parts of the memory for computation-bound application phases (this works even for applications that do not tolerate any errors).

### 5.1.4 Contributions

In this chapter, we explore how Partially-Forgetful Memories can be used by exploiting the intrinsic tolerance of a vast class of applications to some level of error for relaxing this guard-banding in memories [145]. We briefly discussed the challenges to be addressed above. While the notation of PFMs is applicable across the entire memory hierarchy, in the following we first introduce *Relaxed Cache* as an exemplar to address these challenges for partially-forgetful SRAM caches. Preliminary results show how adapting guard-bands to application

characteristics can help the system save significant amount of cache leakage energy (up to 74%) while still generating acceptable quality results.

## 5.2 Related Work

This work represents a convergence of two main bodies of research: variability-aware memory management and approximate computing.

### 5.2.1 Exploiting Memory Variability

There are several efforts on exploiting variations in both off-chip and on-chip memories for power saving. Some of these efforts elevate exploiting memory variability through the software stack and let the programmer to manage variability but none of them considers phasic behavior of an application. The work in [26] virtualizes on-chip and off-chip memory space to exploit the variation in the memory subsystem. They exploit this variability through a Variability-aware Memory Virtualization (VaMV) layer that allows programmers to partition their application’s address space into regions and create mapping policies for each region. Bathen et al. proposed ViPZone [28], an OS-level solution to exploit DRAM power variation through physical address zoning for Energy Savings. ViPZone is composed of a variability-aware software stack that allows developers to indicate to the OS the expected dominant usage patterns as well as level of utilization through high-level APIs. While Relaxed Cache is similar to such approaches in the sense of exploiting the memory variability, our proposal differs in two aspects. First, Relaxed Cache considers also phase behavior of application and criticality of data in its mapping policy. Second, Relaxed Cache intentionally relaxes the guard-bands under programmer control to gain more energy saving through matching different phases of application to the underlying hardware.

## 5.2.2 Approximate Computing

A significant amount of work has proposed different hardware and software techniques that trade-off application fidelity or correctness to gain benefits in energy, performance, lifetime, or yield. Many studies have shown that a variety of applications have a high tolerance to errors [51, 45, 92, 109, 95]. ERSA proposes collaboration between discrete reliable and unreliable cores for executing error resilient applications [92]. Esmailzadeh et al. proposed Truffle [59], a dual-voltage microarchitecture to support mapping of disciplined approximate programming onto hardware. Some software-based approaches use language extensions to give the ability to programmer to perform relaxations [134, 39]. EnerJ [134] uses type qualifiers to mark approximate variables. Using this type system, EnerJ automatically maps approximate variables to low power storage and uses low power operations to save energy. In contrast with EnerJ, Relaxed Cache lets the programmer change the criticality type of variables through the application to match its phasic behavior. Rely [39] is a language that allows specification of computations that execute on unreliable hardware and the analysis that ensures that the computation that executes on unreliable hardware satisfies its reliability specification. Rely considers an abstract model of the underlying hardware with a simple hardware reliability specification. It specifies the reliability of arithmetic/logical and memory operations by some probabilistic values. So it is limited to only soft errors.

Relax [50] is a compiler/architectural framework for exposing hardware errors to the software in specified regions of code for saving computational power. Relax allows programmers to mark certain regions of the code relaxed, and decrease the processor's voltage and frequency below the critical threshold when executing such regions. While Relax focuses on error recovery and hardware design simplicity, our Relaxed Cache approach emphasizes energy-efficiency over error detectability and supports a wider range of power-saving approximations. Moreover, Relax explores a code-centric approach, in which blocks of code are marked for failure and recovery while Relaxed Cache employs data-centric type annotations. Green

[15] trades Quality-of-Service (QoS) for energy efficiency in server and high-performance applications respectively. Green allows programmers to specify regions of code in which the application can tolerate reduced precision. Based on this information, the Green system attempts to compute a principled approximation of the code-region (loop or function body) to reduce processor power.

While most of the previous efforts on approximate computing have focused on the computational part, there exist only two works that have addressed unreliable off-chip main memories. Liu et al. proposed Flicker [100], a software/hardware technique which utilizes the low-refresh rate part of DRAM for storing non-critical data. A programmer using Flicker doesn't have control on the refresh rate of the unreliable part of the DRAM. So s(he) won't be able to adjust this rate to the application's tolerable level of error. Sampson et al [135] proposed two unreliable writes to solid-state main memories, based on Multilevel-cell (MLC) phase change memory (PCM) technology, for improving performance. Unlike prior work on memory, with Relaxed Cache, the programmer can adjust the dynamic knobs to the required capacity/reliability for each phase of program to achieve energy saving.

## 5.3 Relaxed Cache

This section presents *Relaxed Cache* as an exemplar to address the challenges outlined in the previous section for employing partially-forgetful SRAM caches. Relaxed Cache lets the application programmer adjust the cache guard-banding knobs and therefore adapt its reliability and capacity to the application’s demands. Accordingly, the application programmer needs to identify critical memory objects as well as major criticality and computational phases of the application. Although in the current prototype, most of this identification is done manually by the programmer, but some of this burden can be removed by developing automated analysis methods to give hints to the programmer for identifying: (i) critical memory objects [108], and (ii) sensitivity of application’s computational phases to cache capacity [89].

Figure 5.1 shows the high-level overview of the Relaxed Cache scheme. Our scheme requires some small modifications to both the traditional hardware and software components of a system (shaded component of Figure 5.1). In the following, we briefly describe the changes required for hardware and software support; more details are in our technical report [144].

### 5.3.1 Hardware Support

Here, we outline the changes needed for traditional cache hardware (shaded hardware blocks of Figure 5.1):

#### Tuning Relaxed Ways Based on Architectural Knobs

Consider a cache that has  $N$  ways. A fixed small number (e.g.,  $\log N$ ) of these ways are protected, by using a combination of high voltage and error correction codes that assure



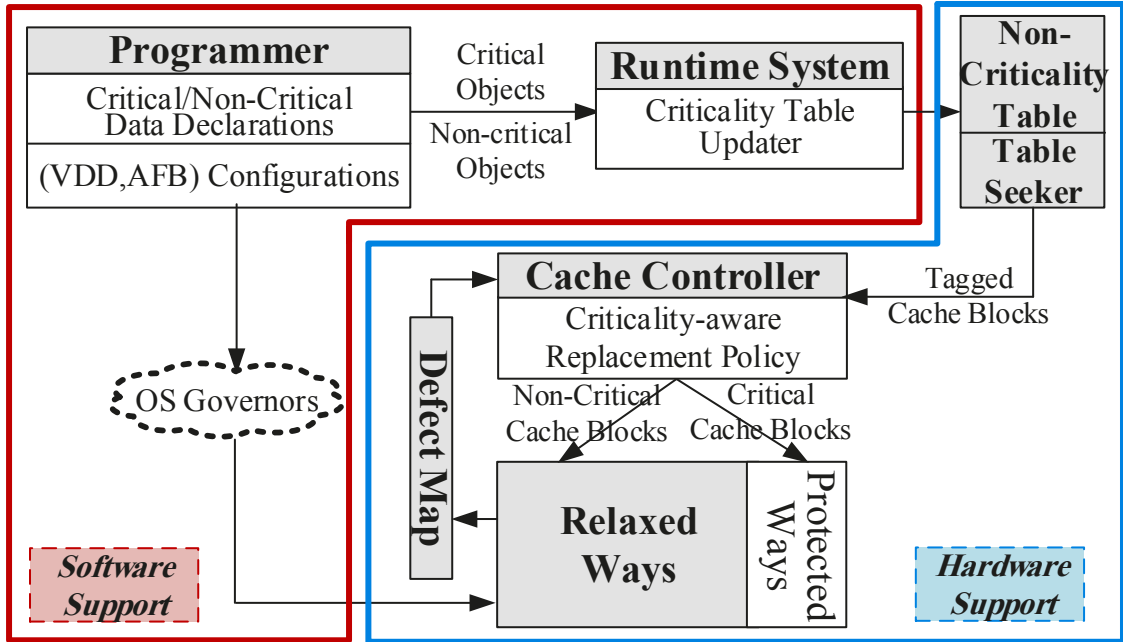


Figure 5.1: High-level diagram showing HW/SW components of Relaxed Cache and their interactions.

their fault-free storage(Protected Ways). The remaining cache ways in a cache set work with a lower dynamically adjustable voltage. The operation of the SRAM array for relaxed ways is controlled by two architectural knobs: (i) Supply voltage (VDD), and (ii) Number of acceptable faulty bits (AFB) per each cache block. Every block with more than AFB faults is disabled. The definition of AFB allows us to relax the guard-bands for a majority of cache ways while controlling the level of error that is exposed to the program. The combination of AFB and VDD also determines the active portion of the cache, hence can be tuned for required performance. According to these definitions, we can have four types of cache blocks for each (VDD, AFB) setting:

- Protected Block (PB): All blocks in the protected ways.
- Clean Block (CB): All fault-free blocks in the relaxed ways.
- Relaxed Block (RB): All blocks in the relaxed ways that have at least one but no more than AFB number of faults.

- Disabled Block (DB): All blocks in the relaxed ways that have more than AFB number of faults.

Figure 5.2 demonstrates these terminologies using a sample 4-way cache when AFB=4.

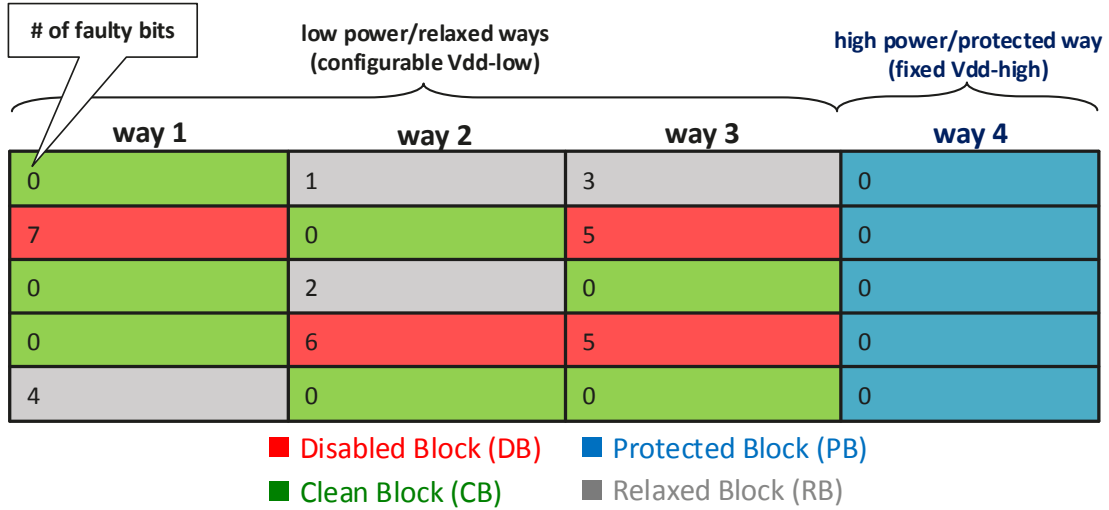


Figure 5.2: A Sample 4-way Cache with VDD=580mV and AFB=4.

## Defect Map Generation and Storage

Disabling those blocks that have more than AFB number of faults requires a defect map for bookkeeping as in previous fault-tolerant voltage-scalable caches (e.g., [22]). To populate the cache defect maps, we can use any standard Built-In Self-Test (BIST) routine that can detect memory faults (e.g., March Tests). The defect map bookkeeping requires a few additional bits in the tag array of each cache block. However, if the number of AFB and VDD levels are limited, this overhead is negligible. For instance, assume that we have four levels of AFB and three levels of VDD. We can show that in this case the defect map overhead is about 1.5% for a cache with block size of 64-byte. Assume that our VDD levels are  $V_0$ ,  $V_1$ ,  $V_2$ . Similarly we have  $A_0$ ,  $A_1$ ,  $A_2$ ,  $A_3$  for AFB levels. Figure 5.3a shows the result of running a march test on four blocks of a set in a 4-way cache for three different VDD levels. The digits inside blocks represent number of faulty bits for each block at that VDD. Note that

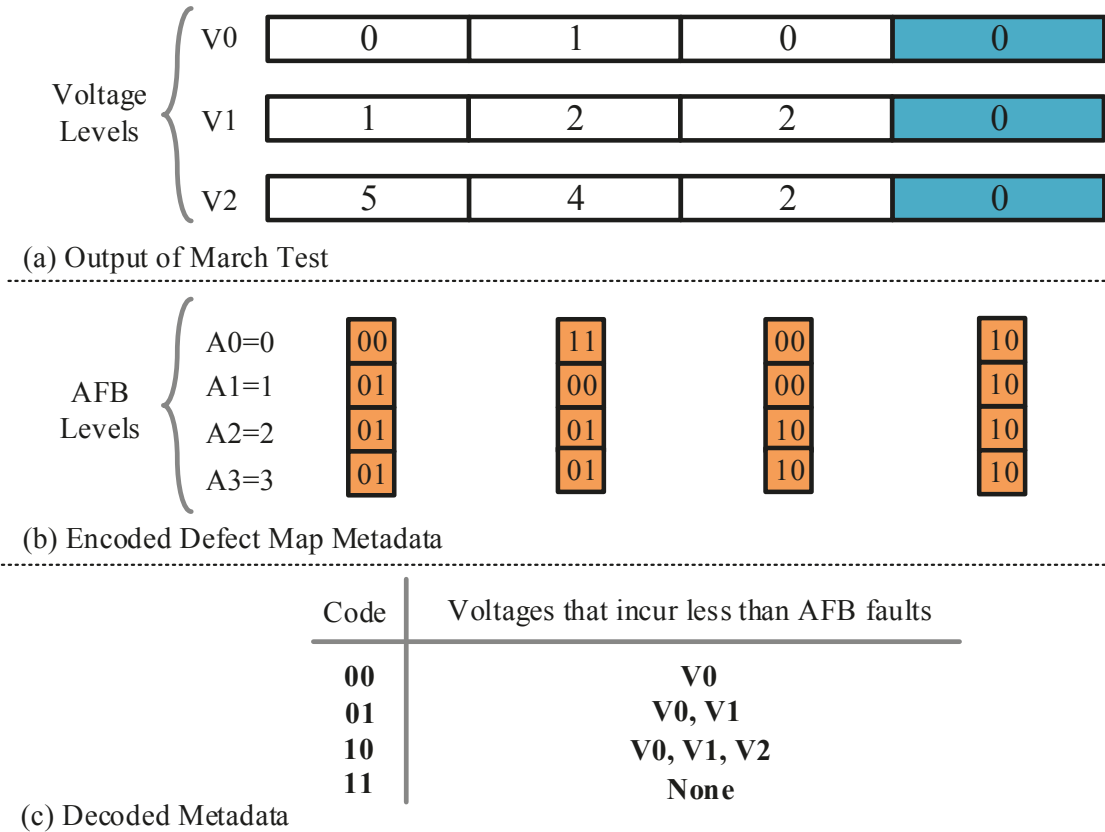


Figure 5.3: Encoding and decoding defect map info in Relaxed Cache

blue blocks belong to a protected way, therefore they are fault-free for all VDDs. The fault inclusion property [63] states that the bits that fail at some supply voltage level will also fail at all lower voltages. Therefore by decreasing the voltage, number of faulty bits increases monotonically. We use this property and for each  $AFB=A_i$  encode the lowest voltage level that results in  $A_i$  number of faults or less (Figure 5.3b). In this manner, for each 64-byte block we use 8 bits to capture defect status of that block in all available (VDD, AFB) pairs, resulting in a 1.5% area overhead for a traditional cache.

### Non-Criticality Table

A dedicated piece of on-chip memory is used to store the application’s virtual addresses that can be approximated (a sample is shown in Table 5.1). Before replacing a block with

an evicted block, this table is searched by hardware comparators to check if the block’s address is contained in any of the address ranges in this table. Accordingly the block is tagged. The number of entries in the table is a design choice, and our experiments show that approximately 30 entries are sufficient. Thus the storage overhead is  $30 * (32+32+1)$  bits  $\approx 4$  cache blocks, which is less than 1% of cache capacity.

Table 5.1: A sample criticality table for Relaxed Cache

| <b>Valid</b> | <b>Start</b> | <b>End</b> |
|--------------|--------------|------------|
| 1            | 0x100ef310   | 0x100ef79c |
| 1            | 0x100ec940   | 0x100efdcc |
| 0            | ...          | ...        |
| 0            | ...          | ...        |
| 0            | ...          | ...        |

Hardware comparators check if the block’s address is contained in one of the table entries. While the data is being fetched from L2 or main memory, these comparators sequentially search for the address inside the table. One comparator checks if the block’s start address is greater than Start and the other one checks if the block’s end address is less than End. The output of both comparators are ANDed to determine if the block is found in the table. The number of comparators is also a design choice. To keep the performance overhead of tagging low, more than 1 pair can be used. Note that since the access latency of L2 cache is in the order of 10 cycles, for a 30-entry table, the delay of the table searching can be masked using 3 pairs of comparators that would effectively make the performance overhead zero. These comparators will be activated only during cache misses ( $\approx 10\%$  of accesses). Our power analysis using Synopsys Design Compiler shows that total (dynamic and leakage) power consumption of this comparators will be about 1% of the total leakage power in baseline mode. This means any leakage saving of more than 1% would effectively compensate this power overhead.

## Making Cache Controller Aware of Block's Tag

Relaxed Cache's replacement policy needs to discriminate between critical and non-critical blocks. Whenever a miss occurs and the missed data block is tagged as a non-critical block the replacement policy should select a victim block from the relaxed ways of the corresponding cache set. However, a critical block is always allocated in a protected way. Because block replacement logic is not in the critical path for hit read/write accesses, this minor modification does not affect cache access latency.

### 5.3.2 Software Support

We now describe changes to software components in Figure 5.1:

#### Programmer-Driven Application Modifications

Data Criticality Declaration: In order to utilize relaxed ways, the software programmer should identify non-critical data structures in the program to be mapped to those relaxed ways. We believe a software programmer can easily make this distinction since (s)he is aware of the functionality and semantics of different parts of the application and related data structures. The constructs `DECLARE_APPROXIMATE(Start-VA, End-VA)` and `UNDECLARE_APPROXIMATE(Start-VA, End-VA)` are used to declare and undeclare data non-criticality dynamically within code. `Start-VA` and `End-VA` are the virtual address boundaries of target region. The additional capability of undeclaring a data region becomes important in two cases: (i) When an error bounding procedure should be performed on non-critical data object before passing it to the next stage of program in order to reduce the propagated error magnitude; here, it is usually desirable to stop introducing any further errors in data

after error bounding. (ii) When a critical procedure should be done on a piece of non-critical data (e.g., computing checksum of pixels at the end of image compression).

Cache Configuration: To utilize the capabilities offered by Relaxed Cache, the software programmer can annotate the code regions with cache configuration hints. These hints are then used by the system to dynamically control the VDD and AFB knobs to adjust the guard-banding to the current phase of execution. The programmer, with in-depth knowledge of the software is in the best position to identify application phases with different criticality and/or memory-insensitivity and based on that guide the adaptive guard-banding in hardware.

To accommodate programmers with different levels of hardware familiarity, we can abstract the Relaxed Cache knobs as shown in Figure 5.4. An embedded system programmer, with fairly good understanding of hardware details, can explicitly set Relaxed Cache’s knobs to fully exploit its capabilities. On the other hand, for a traditional software developer (with little knowledge of hardware), we can abstract the hardware knobs settings into discretized “levels” allowing the programmer to qualitatively set the knobs based on their effects (e.g., low power setting, or high-fidelity setting), as shown in Figure 5.4.

Language extensions then let the programmer use `CONFIG_CACHE(VDD-LEVEL, AFB-LEVEL)` for configuring the cache knobs. Since this reconfiguration incurs a penalty of several cycles, it should be done only in certain phase transitions of an application.

Figure 5.5 shows an image resizer code fragment that uses declarations and cache configuration to save energy. In this example, two large regions of data that can tolerate some level of errors are the original image’s data structure (pointed to by `src`), and the data structure for storing up-scaled image (pointed to by `dest`); the programmer has declared both regions as candidates for relaxed caching. After returning from the `scale()` function, and before moving to the next program phase, (s)he has undeclared both regions.

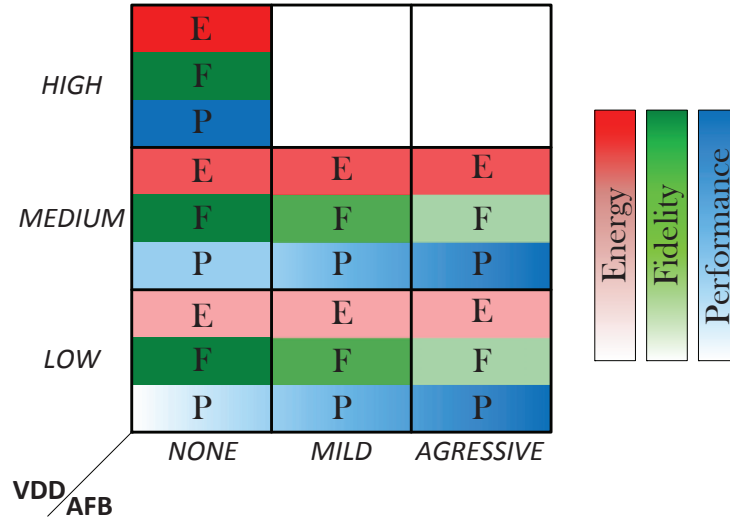


Figure 5.4: Abstracting Relaxed Cache knobs up to metrics familiar to a software programmer (i.e., performance, fidelity, and energy consumption). Note that (VDD = High, AFB = Mild) and (VDD = High, AFB = Aggressive) combinations are sub-optimal, hence not applicable.

## Run-time System

We assume a run-time system uses the programmer’s declarations to keep and update a table of virtual addresses that contain non-critical data objects for each application. During a miss, while the data is fetched from the lower level of memory (L2 data cache or main memory), the comparators in the table seeker sequentially search for the block’s address inside the table; the table seeker then tags the block as critical/non-critical. This tagging is used by cache controller’s replacement mechanism to find an appropriate cache way for replacement. Note that, in order for a block to be tagged as non-critical, the entire data block should hold non-critical data. A data block with mixed critical/non-critical data would still be tagged as critical. Further optimization is possible through criticality-aware data placement during compilation to avoid having mixed criticality blocks, which is out of the scope of this chapter.

```

#include "Approximations.h" // Enables approximation declarations
#define INT_SIZE sizeof(int);
int main(int argc, char**argv){
// Capturing Raw Image on Camera
// Compress Raw Image
/*----- Image Scaling -----*/
CONFIG_CACHE(MEDIUM VDD, MILD AFB);
int *src, *dest;
dest = (int*) malloc(*num_elmnts*INT_SIZE);
#ifdef SRC
    DECLARE_APPROXIMATE((uint64_t)&src[0],
                       (uint64_t)(&src[*num_elmnts]+INT_SIZE-1));
#endif
#ifdef DEST
    DECLARE_APPROXIMATE((uint64_t)&dest[0],
                       (uint64_t)(&dest[*num_elmnts]+INT_SIZE-1));
#endif
src = read_image(in_filename, &sw, &sh, &src_dim);
dest = allocate_transform_image(scale_factor, sw, sh, &dw, &dh, &dest_dim);
scale(scale_factor, src, sw, sh, dest, dw, dh);
#ifdef SRC
    UNDECLARE_APPROXIMATE((uint64_t)&src[0],
                         (uint64_t)(&src[src_dim]+INT_SIZE-1));
#endif
#ifdef DEST
    UNDECLARE_APPROXIMATE((uint64_t)&dest[0],
                         (uint64_t)(&dest[dest_dim]+INT_SIZE-1));
#endif
free((void *) src);
/*-----*/
// Other Image Transformations/Editing
}

```

Figure 5.5: A sample code showing programmer’s data criticality declarations and cache configurations to be used by Relaxed Cache.

## 5.4 Experimental Evaluation<sup>1</sup>

### 5.4.1 Experimental Setup

We modified the cache architecture in the gem5 framework [34] to implement our scheme in detail and used gem5’s pseudo-instruction capability for implementing the required language extensions. The common gem5 parameters used in all our simulations are summarized in Table 5.2. We used the *Random* block replacement policy that is commonly used in embedded processors [11, 12] due to its minimal hardware and power cost.

We injected errors into different cache blocks randomly, with uniform distribution over dif-

<sup>1</sup>More details can be found in our Technical Report [144].



Table 5.2: gem5 Common Parameter Settings

| Parameter     | Value          | Parameter          | Value        |
|---------------|----------------|--------------------|--------------|
| ISA           | Alpha          | Replacement Policy | Random       |
| CPU Model     | Detailed (OoO) | Cache Block Size   | 64B          |
| No. Cores     | 1              | L1\$ Size, Assoc.  | 32KB, 4-way  |
| Cache Config. | L1 (Split), L2 | L2\$ Size, Assoc.  | 256KB, 8-way |

ferent bit positions. We used bit error rate (BER) and leakage power data from [63] which are for a commercial 45nm technology. Using these BERs, we found the distribution for the number of faults for each data array voltage, thus allowing us to compute the expected cache capacity and energy saving for each AFB.

### 5.4.2 Benchmarks

We selected a number of multimedia benchmarks to examine the energy savings enabled by Relaxed Cache. **1) Scale:** Scale is an image resizer application. We use Peak-Signal-to-Noise Ratio (PSNR) as the fidelity metric for this application. A value larger than 28dB is considered acceptable. **2) Susan:** Susan is an image recognition package from MiBench [65]. We used **Image-Smoothing** and **Edge-Detection** kernels from this package. PSNR is used as the fidelity metric for Image-Smoothing. For Edge-Detection, accuracy metric is defined as the fraction of detections that are true positives rather than false positives. A value of 0.8 or more is usually considered acceptable. **3) x264:** This application is an H.264 video encoder [33]. PSNR of 32dB can provide satisfactory video quality for many types of videos.

### 5.4.3 Results

#### Leakage Energy Savings

Figure 5.6 summarizes the achievable leakage energy savings for different Relaxed Cache knob settings (VDD, AFB). The baseline cache is assumed to use 700mV for supply voltage. We assume power-gating for disabled blocks. Consequently, if we keep voltage constant and increase AFB (effectively reclaiming faulty memory blocks), the leakage energy saving decreases. Also note that for voltages  $\geq 540\text{mV}$  increasing AFB does not reduce energy savings. This is due to the low number of faulty blocks at those voltages. This analysis shows that we can decrease leakage energy up to 74% by adjusting the Relaxed Cache knob settings.

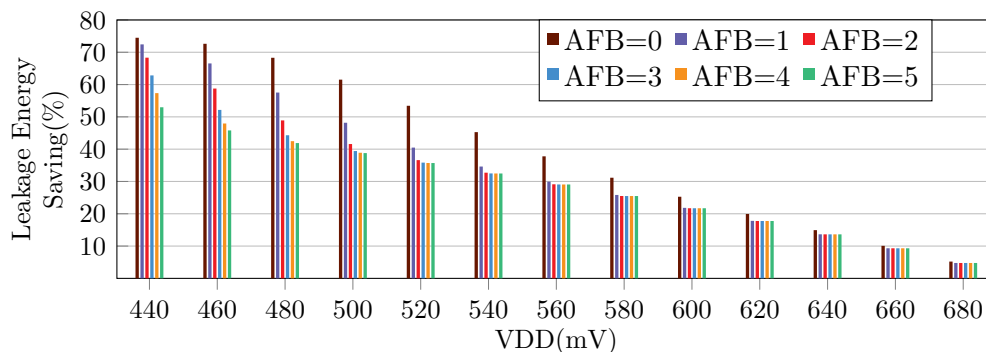


Figure 5.6: Leakage energy savings for a 4-Way L1 cache with 3 relaxed ways. (energy savings are normalized to a baseline cache that uses 700mV.)

#### Fidelity Analysis

Figure 5.7(a) shows the PSNR difference between the images up-scaled using Scale benchmark running on a system with energy-efficient Relaxed Cache versus a system that uses a guard-banded baseline cache.

The PSNR values for Image-Smoothing kernel of Susan package are reported in Figure 5.7(b).

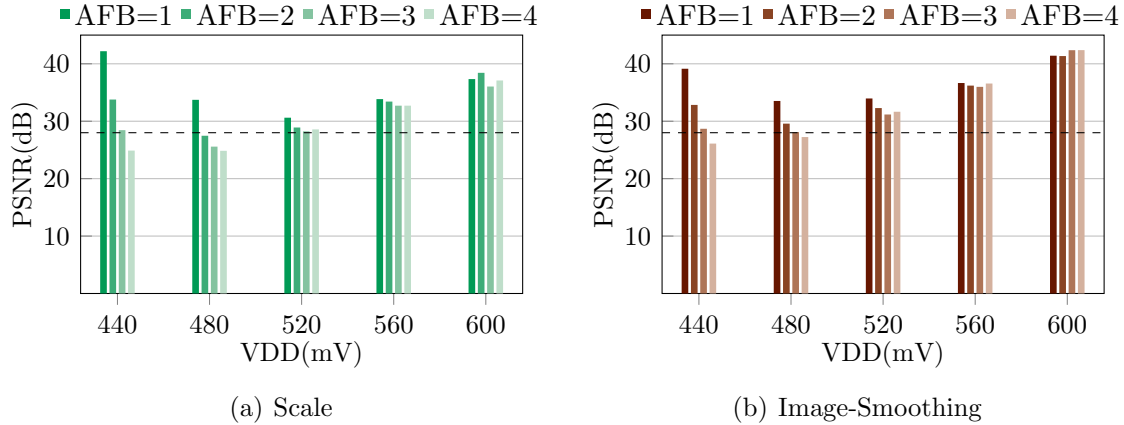


Figure 5.7: Fidelity results for (a) Scale and (b) Image-Smoothing benchmarks.

Same as Scale benchmark, many of the (VDD, AFB) settings result in an acceptable output. However, it is interesting to note that on average the PSNR values are larger than the ones reported for Scale. This shows that the Image-Smoothing is more error-tolerant and therefore the programmer can use a more aggressive policy for that kernel. This observation confirms that different applications have different levels of error-tolerance which can be exploited for energy saving.

Susan/Edge-Detection is the most error-tolerant application that can produce good results even at high error rates. Figure 5.8 shows the result of Susan/Edge-Detection on the Lena test image with different AFBs while VDD is set to 480mV. We can see that even AFB=6 (which reclaims 99% of the Relaxed Cache faulty blocks at that voltage) can result in an acceptable output.

## Performance Analysis

Operating at low voltages effectively disables cache regions affected by errors. Reduced cache capacity has minor impact on the performance of programs with small working set size. The degradations in total execution times are bounded by 3% for Scale and Edge-Detection benchmarks and 2% for Image-Smoothing benchmark over all (VDD, AFB) settings in our

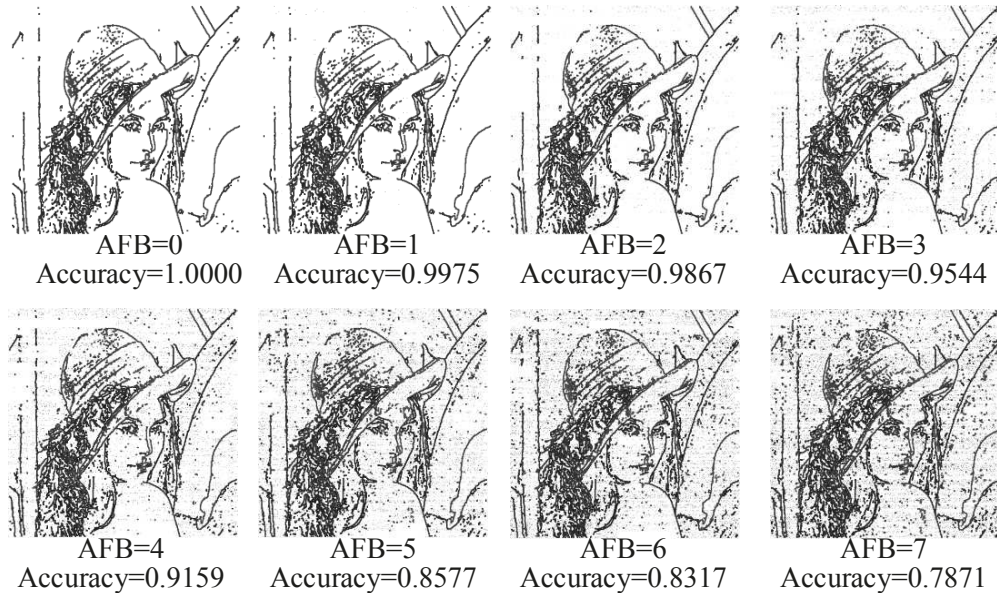


Figure 5.8: Fidelity results for Edge-Detection benchmark (VDD=480mV).

experiments. However, for applications with big working set size, the increased cache miss rate due to reduced cache capacity considerably degrades the performance. But, Relaxed Cache’s AFB knob allows us to reclaim parts of the lost cache capacity, thereby increasing performance of the application. This is particularly useful for application instances where a partial/less-accurate result generated within a deadline is more valuable than a late-but-perfect result. Relaxed Cache provides the means for a programmer to explore this trade-off space, by deciding when higher performance is desirable and when higher output fidelity is preferable. For instance, Figure 5.9 shows the adverse effect of reducing the voltage for the H.264 encoder: x’s correspond to PSNR loss and circles mark FPS loss. Due to the reduced cache capacity, a normal cache (Baseline in blue) would see 38% decrease in FPS, while maintaining the PSNR quality. On the other hand, Relaxed Cache (in red) trades off the PSNR quality for less degradation in FPS (up to 9.5%). However, in all the experiments the PSNR of degraded-quality video is still above 32dB threshold, yielding acceptable quality for the user.

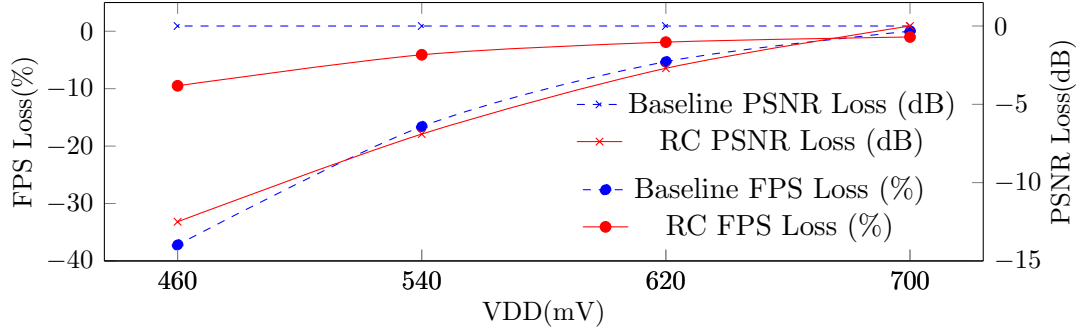


Figure 5.9: FPS-PSNR trade-offs w/ and w/o Relaxed Cache scheme (AFB=4)

## 5.5 Summary

In this chapter, we outlined challenges for utilizing partially-forgetful memories for approximate computing applications. We illustrated how to address this challenges for SRAM caches using the Relaxed Cache exemplar, and demonstrated significant cache energy savings (up to 74% in leakage) with acceptable degradation in the output quality.

As a future work, we consider the challenging issues of write operation –high write energy and limited write wearout– in emerging Non-Volatile Memories (NVMs). The idea is to propose an application-driven mechanism that reduces the number of writes to NVMs by discarding unimportant data stores to trade QoS for energy saving and lifetime improvement.

# Chapter 6

## System-wide Memory Resiliency Design Space Exploration

### 6.1 Introduction

The ongoing trend towards single-chip many-core systems is to deploy simple processing elements (e.g., in-order or RISC-like processors) and exploit task-level and application-level parallelism to maximize system performance by deploying hundreds or thousands of smaller cores, rather than dozens of bigger cores. Additionally, the data-centric nature of several emerging applications creates a demand for larger on-chip memories such as shared Last-Level Caches (LLCs). Hence, the memory subsystem is the fundamental performance and energy bottleneck and is likely to dominate reliability as well as energy concerns for future many-core computing systems [37]. Indeed, increasing failure rates (due to advanced technology nodes, higher integration, voltage scaling, and parametric variations) in many-core platforms, make reliability a major design challenge for both memory and computing components [48]. The design of memory subsystems becomes even more challenging in the face

of user requirements and system constraints like Quality of Service (QoS) and yield, beside possible trade-offs among usually conflicting design metrics (objectives) like reliability, power, and performance.

There is a large body of work on reliable and fault-tolerant memory and cache design in various systems from single-core processors to complex many-core platforms. These approaches mainly allocate redundant resources such as spare rows or error correcting codes (ECC) uniformly over the chip at design time. To maximize the fault coverage, the number of redundant resources may grow proportionally with the increasing number of faults. However, in practice only a limited amount of redundancy can be deployed due to additional overhead incurred in power and area. Even in this case, non-uniform distribution of failures over the chip due to manufacturing defects and variations causes inefficient use of uniformly distributed redundancies. Moreover, non-uniform access latency in shared LLCs as well as varying memory access patterns over different LLC banks, create a complicated resource management problem. To the best of our knowledge, there have been no previous system-level efforts that model and analyze redundancy management and fault-tolerance of distributed on-chip memories in large NoC-based platforms. Therefore, it is necessary to find an approach to efficiently manage distributed redundancies and then search the design space for the best solutions that optimize multiple design objectives in a bounded search time.

A number of Design Space Exploration (DSE) approaches have been proposed at the system-level, for effective design of embedded systems [62] where the search space is huge and systems are optimized for one or more disparate objectives. However, none of the previous approaches for reliable and fault-tolerant design of memories, considers redundancy analysis and optimization. These observations motivate us to first propose a model for effective redundancy management and then leverage the design space exploration approaches for optimizing the problem of effective redundancy management for reliable on-chip memory

design in many-core platforms.

In this chapter, we propose a novel system-level approach to improve the efficiency of reliable on-chip memory design in many-core platforms by sharing the redundancy resources to optimize the system costs.

In particular, this chapter makes the following contributions:

- We propose a system-level design methodology for scalable fault-tolerance of distributed on-chip memory blocks in many-core platforms.
- We introduce a novel *Reliability Clustering* model for efficient fault-tolerance analysis and shared redundancy management of on-chip memory blocks. Each cluster represents a group of cores that have access to shared redundancy resources for protection of their memory blocks.
- We develop analytical models for analysis of reliability, latency, and area overhead leveraging reliability clustering, and use these for design space exploration of fault-tolerant distributed memories in many-cores.
- We perform extensive design space exploration applying the proposed reliability clustering on a block-redundancy fault-tolerant scheme to evaluate the tradeoffs between reliability, performance, and overheads.

*We believe that our proposed design methodology will engender emerging many-core architectures capable of efficiently responding to the multiple challenges of increasing fault rates, variation in fault behaviors, local interconnects, non-uniform memory access pattern and latency, limited shared redundancy, and susceptibility to transient variations. To the best of our knowledge, ours is the first work to comprehensively analyze and explore the design space for redundancy-aware resiliency of distributed on-chip memory banks in many-core architectures.*



## 6.2 Background

### 6.2.1 Reliable Memory Design

Both off-chip and on-chip memories are vulnerable to different sources of faults and errors that degrade their yield and reliability. In general, different kinds of redundancy from physical-level (e.g., spare elements) up to application-level (e.g., information redundancy) are used to improve the reliability of memory components. These redundant resources are usually distributed uniformly over the chip and allocated mainly at design time. Consequently, the non-uniform distribution of permanent faults as a result of manufacturing defects and variability are neglected [24]. Therefore, some memory units may not use their entire allocated redundant resources while other memory units may require more redundancy due to higher failure rates. Thus the traditional approach of over-provisioning redundant resources to guarantee high reliability and high performance may result in excessive costs for many-core platforms with large distributed memories. Hence, it is necessary to minimize the amount of redundant resources used for error protection. One approach is to share them among all the memory units (LLC banks), so that cache banks with higher failure rates can borrow the unused redundant resources from other banks [20].

This problem can be cast as allocating the shared redundant memory resources over the chip and binding them to the faulty cache blocks to gain required reliability while minimizing other design objectives like power and cache access delay. The search space of allocation and binding problem is vastly increased due to non-uniform distribution of variability-induced permanent faults and also non-uniform memory access patterns to different cache banks due to varying behavior of different applications running on the system. Therefore, it is necessary to find an efficient approach to model this problem and search for the best solutions that optimize the design objectives in a bounded search time.

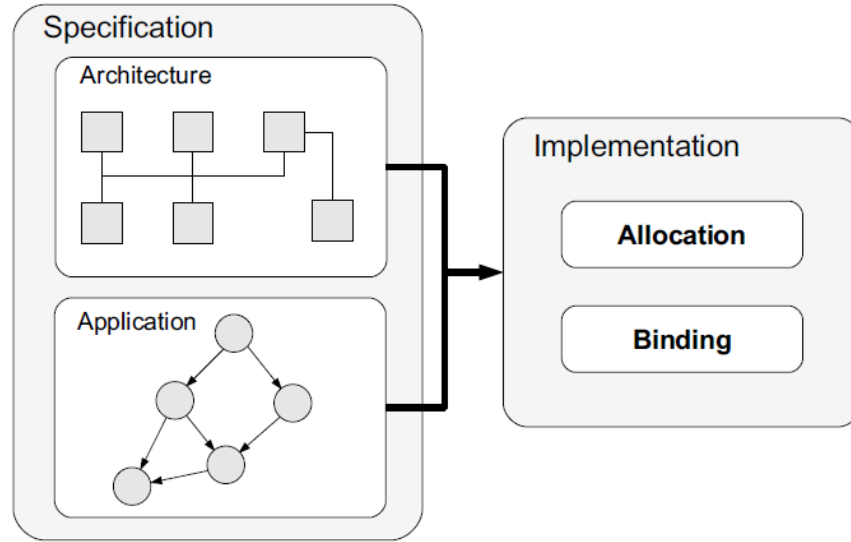


Figure 6.1: A general system-level design space exploration problem [101].

## 6.2.2 Design Space Exploration

In system-level design, Design Space Exploration (DSE) refers to a (semi)automatic system-level approach that explores the search space to identify implementations which satisfy the requirements of the designer. Then, the task of design space exploration problem is to find the set of optimal feasible implementations for a given specification. Figure 6.1 depicts the system-level design flow. The specification consists of the *architecture*, the *application*, and the relation between these two parts [101]. An implementation or solution of the problem which is deduced from the specification consists of two main parts including *allocation* of resources and *binding* of tasks to allocated resources. For multi-objective problems, where usually one objective cannot be improved in one dimension without worsening another, DSE does not seek a single optimal solution but rather a set of equally good solutions called Pareto-optimal (or near Pareto-optimal, often also called quality) set. We deploy this DSE approach for redundancy management of memory reliability in many-core platforms.

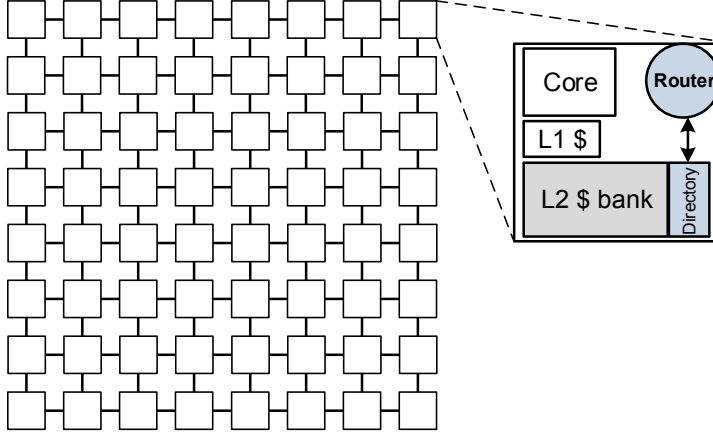


Figure 6.2: Exemplar Many-core Architecture with an 8x8 mesh NoC.

### 6.2.3 Exemplar Many-core Architecture

We illustrate our DSE approach using an exemplar tiled interconnected many-core, where each tile comprises a processor core, private L1 data and instruction caches, a shared L2 cache bank, and network router/switch. Tiles are interconnected as a 2-D mesh via a network-on-chip (NoC) infrastructure. Figure 6.2 shows our baseline 64-core architecture with an 8x8 mesh NoC. Based on our cache organization, the L2 bank is a portion of the larger distributed shared last-level cache (LLC). The baseline design assumes a Non-Uniform Cache Architecture (NUCA) [81] for LLC. A directory-based protocol is implemented in order to maintain cache coherence. In this platform, redundancy resources in the form of spare cache blocks that uniformly distributed over all LLC banks.

### 6.2.4 Motivation

As an example, Figure 6.3 shows a  $2 \times 2$  NoC with four LLC banks, a fixed amount of redundancy per bank (highlighted) and a random set of faulty blocks (cross dots). Figure 6.3(a) shows a sample mapping of four tasks (T1-T4) to the cores. As indicated in Figure 6.3(b), the distribution of faults is non-uniform and some banks (e.g., B1) have many more faulty

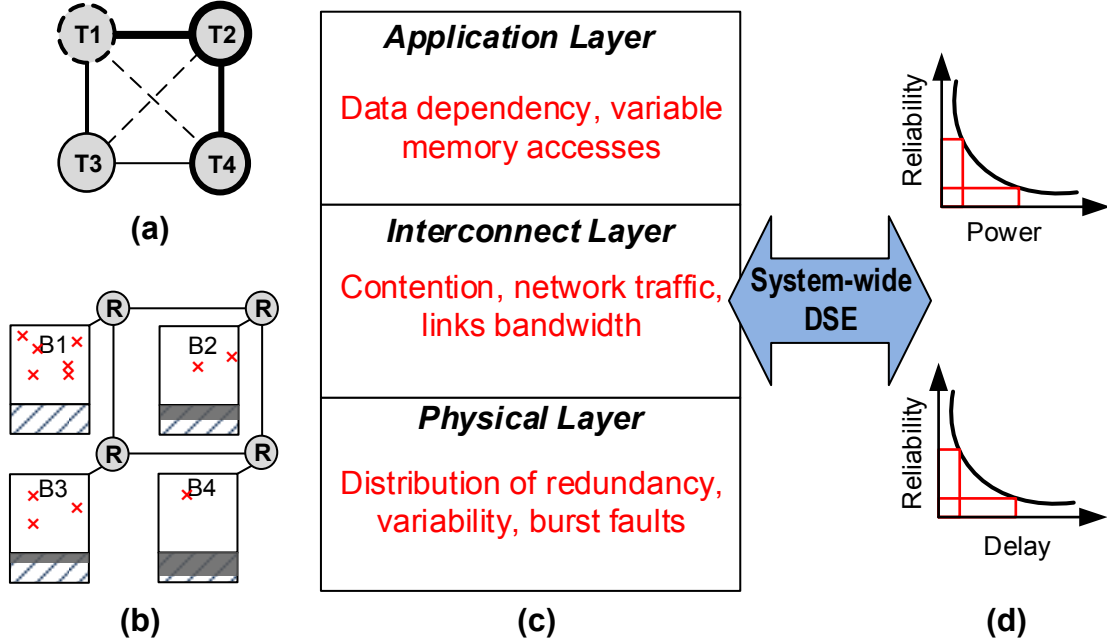


Figure 6.3: A sample case of a  $2 \times 2$  NoC and four tasks mapped on its cores; a) a sample task mapping; b) uniform distribution of faults vs non-uniform distribution of faults (shaded areas represent redundancy resources and cross signs represent faulty blocks); c) cross-layer design methodology; d) DSE and Pareto-optimal solution.

blocks than others (e.g., B4). On the other hand, due to non-uniformity of application memory demand or tasks mapped on different cores, some nodes become the hotspot of the system (e.g., T2). We assume that redundant resources are shared (like LLC banks) and can be used for protection of local faulty blocks as well as global blocks (in other banks). However, access to redundant blocks over the network incurs higher access latency and more traffic over the network which impacts the system power. Also, we assume it is possible to use block-level power gating [63] to turn-off some unused redundant blocks to save power.

Even for this exemplar architecture, the design space for reliable memory organization becomes very complex in the face of multiple conflicting objectives (e.g., delay, power, fault coverage, etc.) and can be cast as a complex multi-objective optimization problem. On the other hand, none of the previous approaches to reliable and fault-tolerant memory design have performed any redundancy analysis and optimization to minimize the cost of protection. These observations motivate us to integrate memory protection techniques with a

system-level DSE optimization framework to optimize the problem of effective redundancy management (sharing, allocation, and binding) in emerging many-core platforms. Note that our approach is complementary to, and can further leverage all conventional reliable and fault-tolerant memory design techniques to minimize the cost of memory protection.

## 6.3 System-Level Fault-tolerance Modeling

A methodology for fault-tolerance of distributed memory systems for scalable NoC platforms faces several inter-related challenges, requirements, and features: communication must be localized to reduce interconnect overheads; fault-tolerance schemes need to be topology-aware, and must organize and distribute redundancy based on criticality of communication paths; redundancy needs to be managed as a shared resource across the platform; and fault-tolerance should be addressed at higher design abstraction levels to gain more leverage over its effects on performance, cost, power, etc. To meet these needs, we develop analytical models and a system-level methodology for fault-tolerant design of distributed on-chip cache memories for a tiled NoC-based CMP. This chapter focuses on shared L2 banks that need to be protected. However, our approach can be applied to any form of on-chip memory architecture in NoCs that need reliability. To estimate design parameters we develop analytical models include a Reliability model, Access latency model, and Area overhead model. Using these models, we can analyze the trade-offs between yield, performance, and energy/area overheads of a fault-tolerant design of on-chip memory subsystem. We describe details of these models in our technical report [17]. Next, we present a novel reliability clustering concept for scalable, modular, and efficient design and implementation of fault-tolerant schemes applied to protect the memory blocks.

### 6.3.1 Reliability Clustering

To model fault-tolerance and organize redundancy we divide the whole NoC into clusters comprised of multiple groups of tiles. Each cluster is a subsection of the base NoC with the same topology but in a smaller group of tiles. The clustering is used to partition redundancy sharing among the tiles inside the cluster. In each cluster, some specific tiles (e.g., center tiles) contain the redundancy used for fault-tolerance of all tiles inside the cluster; these are

labeled as redundancy nodes. These redundancy nodes can be used flexibly to accommodate different fault-tolerance schemes and varying forms of redundancy, such as redundant rows/columns/blocks; exploiting sections of available clear/faulty blocks as redundancy; and ECC codes. Furthermore, we leverage the available interconnect backbone to support implementation of a variety of modular, scalable, and efficient fault-tolerance schemes. For instance, in our exemplar tiled NoC architecture, we modify the direct router connected to the redundancy nodes to support our selected fault-tolerant scheme. Because such clustering organization is independent of a particular topological structure, various physical topologies can serve as fixed-silicon but dynamically reprogrammable reliable multicore clusters on top of NoC platforms. Meanwhile, the inherent reconfigurability allows customizing clusters according to not only the clustered and varying fault rates but also to application communication patterns and resilience needs. Therefore, clusters can be defined statically or dynamically during runtime. For the sake of simplicity, here we consider static clustering. Our mesh-based NoC is composed of  $N$  nodes,  $C$  clusters, with each cluster containing a  $d \times d$  mesh of tiles ( $d =$  cluster dimension size). The size and number of clusters can affect the yield, overhead, and network latency. The cluster dimension size ( $d$ ) would determine the upper bound on latency of fault-tolerant LLC accesses. Depending on the shape and size of each cluster, number of clusters, amount of redundancy, and distribution of redundancy among clusters, we can explore different design strategies which meet various design constraints.

## 6.4 Evaluation

To illustrate the flexibility and utility of our exploration methodology, we outline a sample design space exploration study using the exemplar many-core platform.

### 6.4.1 Experimental Setup

We use SoCIN, a cycle-accurate SystemC model of a Mesh-based NoC [163] that uses Worm-hole switching with a 4-phit buffer size and an XY routing to avoid deadlocks. SoCIN router contains input buffers, the arbiter, the crossbar, and the links as depicted in Fig. 1. Each router is connected to a 1MB L2 cache bank. All L2 cache banks share the same address space of 64 MB LLC. Additionally, for each router, there is a module generating data and instruction requests based on memory traces obtained from a Simics [102] simulation using 64 sparcV8 as cores. A workload of parallel programs with very distinct behaviors is created using benchmarks from SPLASH2 [159], PARSEC [33], and a parallel version of MiBench [65] suites. With these memory traces as input to our framework, we are able to extract performance and energy results. The performance results are the total number of cycles to execute all 64 parallel tasks (for each application) and the energy results are obtained from Cacti 6.5 [116] for the memory subsystem and from Orion 2.0 [79] for the network-on-chip. Table 6.1 summarizes the experimental setup of our architecture.

For the sake of this study, faulty LLC banks are modeled randomly since SRAM cell faults occur as random events due to the major contribution of the random dopant fluctuation to the process variation [2]. Based on the results of a recent work [118], the predicted probability of failure for SRAM cells can be up to  $2.6^{-4}$ . Here, since our case study is a block-redundancy fault-tolerant scheme and the analysis is at system level, we model failures at the block level. Based on that, the probability of block failure would be up to  $7^{-2}$ . To enable a fair analysis during our evaluation, we consider 8 fault rates ranging from  $10^{-3}$  to  $7^{-2}$ . To determine the amount of redundancy based on our reliability model, to keep the reliability more than 99%, the redundancy (R) should be more than 2%. On the other hand, to keep effective yield more than 95%, we should keep the redundancy less than 5%. Thus we consider redundancy for any value ranging from 2% to 5%.



Table 6.1: Simulation Configuration

|                    |  |
|--------------------|--|
| Processor Core     | 64 SPARC V8  |
| L1 I/D Cache       | 2 Banks, 32KB each, 4-Way, 32B Block Size, 2 cycle, LRU  |
| L2 Cache (LLC)     | shared 64MB, 64 banks, 16-way assoc, 64B block, 10 cycle, LRU  |
| Coherence protocol | Directoy-based   |
| Memory Latency     | 250 cycles   |
| Interconnect       | NoC of 2D mesh ( $8 \times 8$ for 64 LLC banks and 64 cores) , 32-byte links (2 flits per memory access), 1-cycle link latency, 2-cycle router, XY routing, Wormhole Switching, 4 phit buffer size |
| Frequency          | 1.0 GHz  |
| Technology         | 45nm   |

### 6.4.2 Design Space Exploration Studies

We evaluate the impact of system-level reliability clustering on performance, energy, and area overhead of the system using simulation runs of the experimental platform described earlier. For different system configurations we change one design parameter such as fault rate or amount of redundancy and study its effect on different design metrics of the memory subsystem. We consider the design space at two levels: cluster-level and system-level.

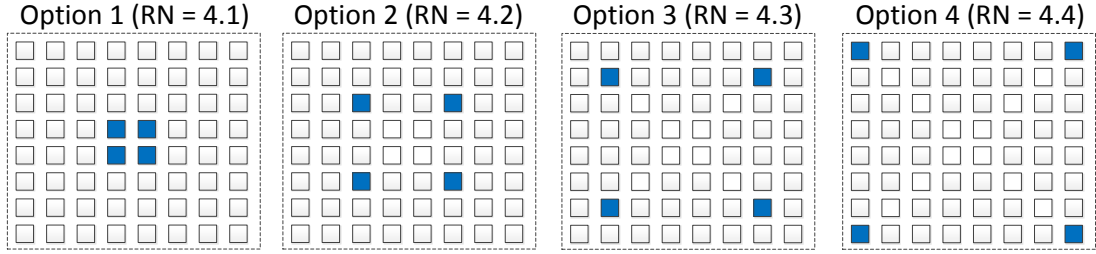
1. **Cluster-level (Intra-cluster):** Here, we study the effect of redundancy distribution (number and location of redundancy nodes) on system design metrics. In this set of results, we consider the whole NoC as one cluster, fix either the amount of redundancy or fault rate, and change the redundancy distribution. In these experiments we explore the redundancy organization by changing either the number of nodes that contain redundancy (redundancy nodes) or their location. Redundancy node distribution can be studied in two directions, ranging from central nodes to all outward nodes or ranging from corner nodes to all inward nodes. Redundant elements are spread equally among all nodes which contain redundancy. Proposed distributions are selected based on a regular and scalable pattern which is independent of the size of the cluster. Figure 6.4(a) presents some

possible distributions for our base architecture with four redundancy nodes. Here, for each redundancy distribution we have other variations of the distribution by spreading the nodes from the center (Option 1) towards the corners (Option 4). Note that the label  $N.X$  means that the configuration has  $N$  nodes with redundancy and  $X$  representing the distribution option. Higher values of  $X$  represent redundancy nodes that are closer to the corners.

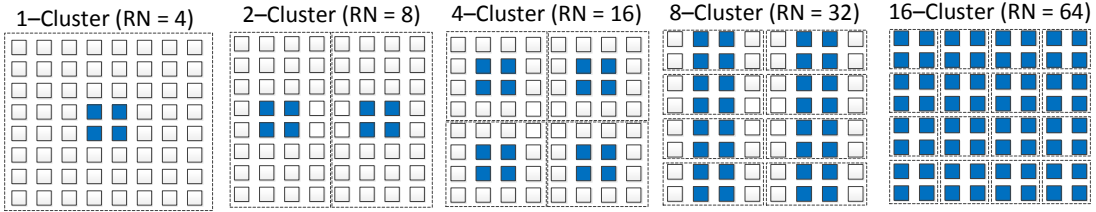
2. **System level (Inter-cluster):** Here, we investigate the effect of node clustering and shared redundancy management among clusters on system design metrics. In this set of results, we change the number and size of clusters while fixing the total redundancy in the system. We select the size and number of clusters based on a regular and scalable pattern. Redundancy is spread equally among all clusters and all redundancy nodes inside each cluster. Here, we put the redundancy nodes in the center of each cluster. The intuition behind this approach is to guarantee that the redundancy nodes are always surrounded by a certain number of cores. This has the goal of not only minimizing the average distance between the cores and the redundancy nodes but also minimizing the variance in the average distance for each case. Figure 6.4(b) presents some possible clustering and distribution of redundancy for our base architecture with four central redundancy nodes per cluster. Here we illustrate sample distributions for 1, 2, 4, 8, and 16 clusters with 4, 8, 16, 32, and redundancy nodes, respectively.

### 6.4.3 Sample Exploration Results

In this section, we summarize the normalized performance/energy results of the block-redundancy scheme using the proposed clustering methodology, with respect to a baseline system without fault-tolerance support (i.e., where access to a faulty memory address results in an on off-chip memory access). Detailed experimental results are in our technical



(a) Cluster-level



(b) System-level

Figure 6.4: Possible redundancy configuration patterns in (a) cluster-level and (b) system-level models with four redundancy nodes per cluster.

report [17].

## Cluster-level Results

Figure 6.5(a) shows the gains of performance – the normalized execution time to the baseline for each configuration shown in Figure 6.4(a) with 3% of memory redundancy and also 3% of block fault rate in the system. Note that 3% block fault rate means that 3% of all memory blocks in the system are faulty. This figure shows the susceptibility of performance gains varying from one application to another. While the performance improvements vary across most applications, as expected, configurations with more redundancy nodes (e.g., configuration 16.2) present better results.

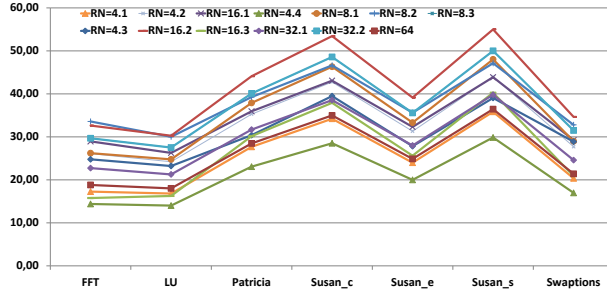
Figure 6.5(b) presents the average performance results over all benchmarks when using 3% of memory redundancy and changing the block fault rate in the system. We note that configurations using too many redundancy nodes do not present good results. This may suggest that the best distribution must not have a small amount of redundant memory per

node because this will spread the redundant area too much, creating a higher average NoC distance between the nodes.

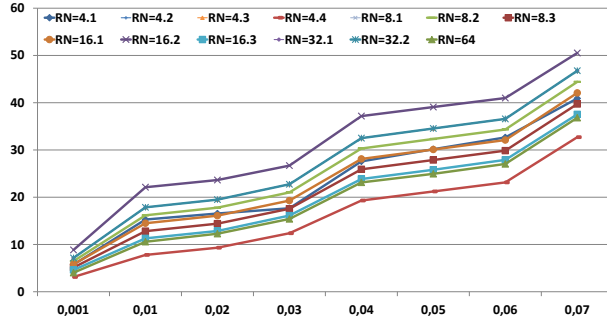
In another experiment we evaluate the effect of different amounts of redundancy for cluster level while setting the block fault rate to 3% shown in Figure 6.5(c). Note that it is not sufficient to have redundancy greater than the fault rate, since the faults may not be evenly distributed, and other factors (e.g., routing and contention) may affect performance. Indeed, Figure 6.5(c) shows an inversion from 4% of redundancy memory: configurations that were inferior in the previous experiments start to present better performance and those that were previously superior now appear to become worse. This suggests that at some point when the total of redundancy memory is higher than the amount of faulty blocks in the system, it is better to have the redundancy nodes in the corners. This could be due to the fact that as the *XY* routing tends to concentrate the load in the center of the NoC [52], these packets for redundancy memory requests (which now occur less frequently) may generate less contention if avoiding the middle of the NoC.

## System-level Results

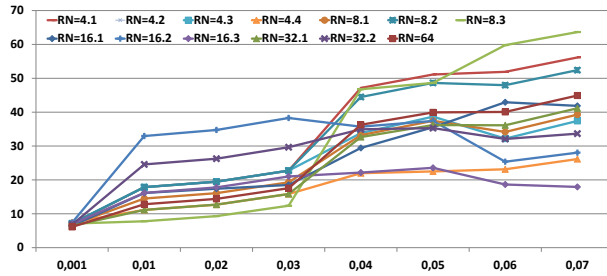
For the inter-cluster case we put the redundancy nodes in the center of cluster and change the size and number of clusters. Similar to Figure 6.5(a), Figure 6.6(a) presents the results at the system-level in terms of performance gains when fixing the redundancy memory at 3% and the block fault rate also at 3%. In this case, for all configurations except 16-cluster, there is not much variation from one application to another. This is a configuration that spreads the redundancy data equally throughout all the nodes and therefore is more susceptible to different memory access rates. For lower memory accesses it works well because there are few redundancy memories per node across all nodes. For high memory accesses it may not work well since it has a higher chance where the redundancy nodes may be too far away.



(a) Benchmarks



(b) Fault rate



(c) Redundancy

Figure 6.5: Normalized performance improvement of different cluster-level configurations across different a) benchmarks, b) fault rates, c) amount of redundancy.

Figure 6.6(b) presents performance results for these configurations with a total of 3% of redundancy memory while changes the block fault rate, ranging from 0.1% to 7%. The figure shows that the 8-cluster configuration has the better results. This is due to the fact that this configuration presents a low average distance between cores and redundancy area and also because this average distance does not vary too much considering all cores.

Figure 6.6(c) presents results regarding the same exploration presented in Figure 6.5(c) but this time with this set of system level configurations. Similarly, we observe that a few

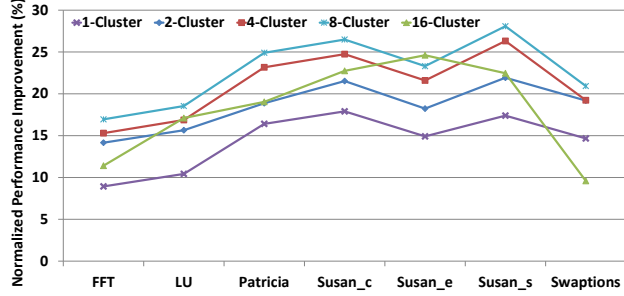
changes occur after the point of 3% of redundancy memory but differently from the case of cluster level configurations, there is no clear change of scenario, i.e. the better result remain unchanged. This could be due to the fact that in this set of configurations there is no configuration that places the redundancy nodes in the edges of the NoC.

Overall, by looking at the results on both sets of experiments, we observe that good decisions on redundancy organization or selecting proper cluster-level and system-level configurations can lead up to 20% improvement in normalized performance gains over the baseline. This is a relevant differential since it incurs no increase in overhead and only requires changing the redundancy organization.

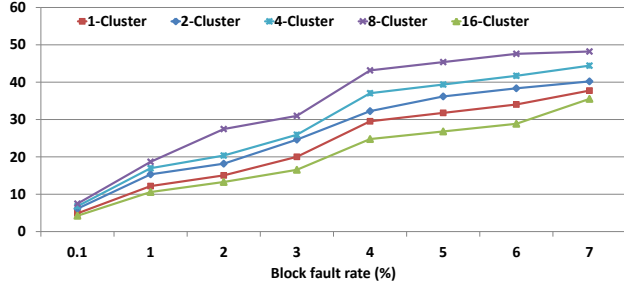
## Energy Results

We also studied the energy saving (normalized to baseline) for both sets of experiments by fixing the fault-rate to 3% and varying the redundancy from 0.1% to 7%, as represented in Figure 6.7. The trends here are quite similar to the one presented in Figures 6.5(c) and 6.6(c). The difference is that since off-chip memory accesses consume more energy than regular on-chip memory accesses, the penalty for not having a redundancy memory hits the baseline the hardest. The results presented here show the energy savings of the redundancy cluster approaches compared to the baseline.

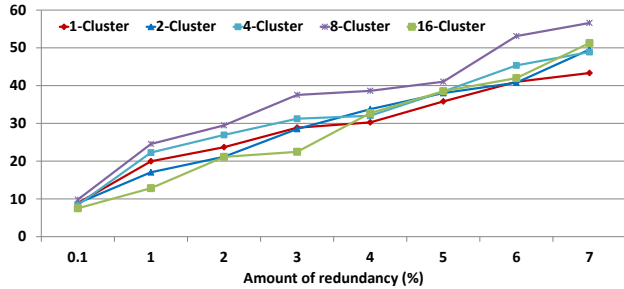
In Figure 6.7(a) we present the results for the intra-cluster level approach. The energy consumption in this experiment for the most part follows the same trend as the performance results. Some situations can present better results depending on the average distance between the cores and the redundancy nodes (e.g.,  $RN = 16.2$ ). If better distributed, the energy consumption to reach the redundancy nodes will also be smaller. After the point where the amount of redundancy surpasses the block fault rate, some configurations that place the redundancy blocks far away from the center of the NoC present better results (e.g.,



(a) Benchmarks



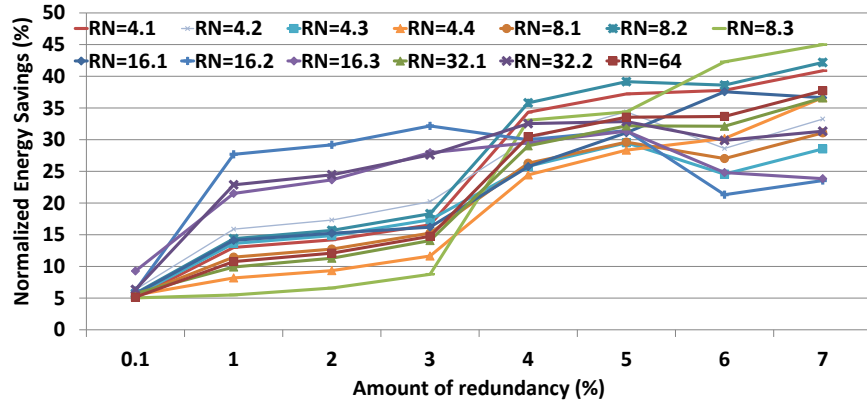
(b) Fault rate



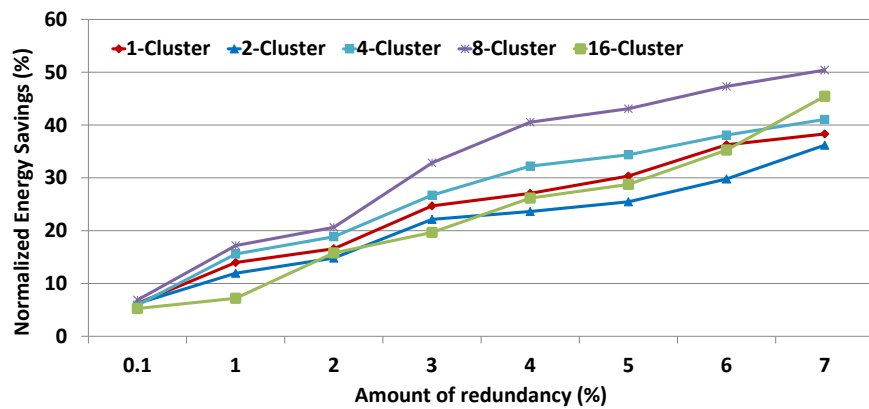
(c) Redundancy

Figure 6.6: Normalized performance improvement of different system-level configurations across different a) benchmarks, b) fault rates, c) amount of redundancy.

$RN = 8.3$ ). This could be because of the tendency of XY routing to create more contention in the center of the NoC. In Figure 6.7(b) the inter-cluster (system-level) approach energy savings are presented and although the trends are similar to the one presented in Figure 6.6(c), the differences between the configurations with different number of clusters seem to be increased and therefore, the 8-cluster configuration presents the best results due to an even distribution of redundant blocks in the NoC without reducing the amount of redundancy per node too much. The 16-cluster results present a better result at 7% redundancy rate. This is due to the fact that in this system-level approach if the redundancy rate is much higher



(a) Cluster Level



(b) System Level

Figure 6.7: Energy results across different amount of redundancy for (a) Intra-cluster, (b) Inter-cluster (system-level) configurations.

than the block fault rate, the amount of redundancy per node reaches a point where fewer and closer redundancy nodes are enough to cover the faults. With lower distance between the nodes, lower NoC energy is consumed.

Overall, depending on different configurations the normalized energy savings can be improved up to 47% in the cluster-level approach and 51% in the system-level approach for different amounts of redundancy.



## 6.5 Summary

To construct reliable memory architectures in emerging multi/many-core NoC platforms, designers must consider the interconnect, distribute and utilize redundancy efficiently to alleviate the high cost of fault-tolerance schemes, employ a hierarchical set of fault-tolerance strategies, and create novel design paradigms that consider system-level issues. In this chapter, we first proposed a system-level design methodology for scalable fault-tolerance of distributed on-chip memories in many-cores. We introduced the novel concept of *Reliability Clustering* for efficient redundancy management and fault-tolerant design of memory blocks. Experimental results on an exemplar 64-core CMP with an 8x8 mesh NoC show that distinct design strategies or reliability clustering configurations of a block-redundancy scheme may yield up to 20% improvement in normalized performance gain and up to 51% in normalized energy gain, uncovering many interesting design configurations. We also Future work will explore application-aware dynamic clustering of cores and shared memory redundancy for fault-tolerance.

As a future work, we want to define and formulate the problem of shared redundancy allocation and binding as a DSE problem. Then, propose a system-level optimization approach to solve it. We want to exploit a cross-layer approach that combines both physical layer variability issues as well as application-level memory access variations in design for memory reliability. Finally, we need to integrate the proposed optimization approach in a comprehensive Design Space Exploration (DSE) framework considering multiple system and architecture-level design objectives such as reliability, power, and performance.

# Chapter 7

## Conclusion and Future Work

### 7.1 Conclusion

Computing systems become more vulnerable to different sources of failures from soft errors to ever-increasing hard errors in the nano era. Reliability becomes a major design challenge, especially for on-chip memories. The complexity of systems is increasing and there is need to new leverage new scalable and cost-efficient approaches. We think error-resiliency is more important than reliability. The premise of this thesis is to provide multiple solutions in different layers of system hierarchy targeting a verity of architectures from embedded single-core microprocessors to emerging large many-core platforms to address cost-efficient error-resiliency of on-chip memory in Nano era.

First, two approaches were proposed for cost-efficient fault-tolerance of cache memories in different processors. We first presented FFT-Cache, a flexible cache architecture to protect regular SRAM-based cache memories against high degree of process variation-induced permanent faults. Then, we proposed EEC, a low-cost technique to leverage embedded erasure coding to tackle soft errors as well as hard errors in data caches of a high-performance as

well as an embedded processor.

Second, to efficiently tolerate the large number of permanent faults in the large NUCA LLCs in CMP architectures, when operating in the near-threshold region, two highly reconfigurable fault-tolerant cache design, REMEDIATE and RESCUE, were presented.

Third, two interconnect-aware designs were introduced to protect LLC in NoC-based architectures. In first design, we leverage the NoC infrastructure for efficient protection of on-chip memories against permanent faults. Then, we propose a co-design error coding approach for LLC and interconnect to minimize the overall overhead of error protection in NoC architectures.

Fourth, the concept of Partially-Forgetful Memories (PFMs) introduced to demonstrate how to exploit the intrinsic tolerance of a vast class of applications to some level of error for relaxing the guard-banding in memories. We outlined challenges for utilizing partially forgetful memories for approximate computing applications. Then, we proposed *Relaxed Cache* as an exemplar to address the challenges for partially-forgetful SRAM caches.

Finally, this thesis provided novel system-level design space exploration and optimization approaches to minimize the cost of resilient on-chip memory design in many-core architectures. First, the concept of reliability clustering for distributed fault-tolerance of on-chip memories in NoCs was proposed. Then, Proposing a DSE optimization approach to minimize the cost of redundancy assignment toward LLC protection in many-core platforms.

## 7.2 Future Work

This thesis is the first to introduce novel concepts such as reliability clustering and distributed shared redundancy and propose new approaches such as NoC-based fault-tolerance

and Codesign error coding to resilient on-chip memory design. As a result, there are multiple ways in which this work can be extended in the future:

- Extension of scalable cache fault-tolerance: To leverage the benefits of both remapping-based as well as ECC-based approaches, a hybrid scheme of *Remapping + ECC* would be an interesting approach. Such approach alleviates the cost of fault-tolerance in case of low bit error rates by using ECC instead of remapping and also protects caches against transient soft errors.
- Extension of NoC-based LLC fault-tolerance: Since this is a first of its kind piece of work, there are multiple opportunities for further optimization of this work such as adaptive fault-aware routing, dynamic redundancy assignment for fault tolerance, and non-unified design for fault tolerance. Moreover, design for soft error protection is also another promising direction of work.
- Extension of CoDEC approach for adaptive/hybrid coding: Since there are many trade-offs in design of error coding schemes for cache memories as well as NoCs, it would be interesting explore how to modify the CoDEC approach to make it adaptive by leveraging hybrid coding techniques. For example, depending on the NoC routing scheme and hop count of accessed block, CoDEC can leverages different coding techniques to optimize the coding overheads.
- Extension of Partially-Forgetful Memories:
  - Partially-Forgetful Non-Volatile Memories: Emerging non-volatile memories such as Phase Change Memory (PCM) and Spin-Torque Transfer RAM (STT-RAM) suffer from high write energy and limited write wearout and there are many research efforts to make their write operation more energy efficient and limit their wearout issue [47, 168]. Considering the approximate computing applications features, a future direction is to propose an application-driven mechanism that

reduces the number of writes to partially-forgetful NVMs by discarding unimportant data stores to trade QoS for energy saving and lifetime improvement.

- Partially-Forgetful Software-Controlled Memories: Any system that employs software-controlled memories (a.k.a. scratchpad memories – SPM and tightly-coupled memories – TCM) already contains a mechanism to statically or dynamically map application data onto these memories. This can significantly negate the overhead induced in Relaxed Cache to manage bookkeeping of data partitioning declarations (i.e., criticality table) and perform data mapping accordingly (i.e., the modifications in the cache controller). The idea of partially-forgetful memories therefore naturally lends itself to software-controlled memories.
- Partially-Forgetful Distributed Memories for Many-core Systems: The data-centric nature of several emerging applications creates a demand for denser on-chip memories in emerging many-core systems which makes the memory subsystem one of the fundamental performance and energy bottlenecks in such systems. Partially forgetful memories provide an opportunity for co-optimization of reliability and energy consumption. A major challenge would be sharing on-chip partially-forgetful memory resources between a set of applications running on the system while adjusting memory reliability knobs according to the requirements of the applications.
- Extension of Reliability Clustering: Since the concept of Reliability Clustering is a first of its kind piece of work, there are many opportunities for further optimization. Future work will explore dynamic clustering of resources during run-time. It can leverage application-aware or cross-layer policies to create heterogeneous clusters by considering application-level as well as physical-layer variabilities during run-time.
- Extension of System-level Memory Resiliency: Since this is the first piece of work exploring shared redundancy in distributed cache memories, we believe that there

are many directions for future work in the redundancy-aware reliability management domain. As a direct extension, we want to define and formulate the problem of shared redundancy allocation and binding as a DSE problem. Then, propose a system-level optimization approach to solve it. We want to exploit a cross-layer approach that combines both physical layer variability issues as well as application-level memory access variations in design for memory reliability. Finally, we need to integrate the proposed optimization approach in a comprehensive Design Space Exploration (DSE) framework considering multiple system and architecture-level design objectives such as reliability, power, and performance.

- Extension of Resiliency DSE approach with dynamic redundancy allocation: By dynamically allocating some parts of shared memory (LLC) as redundancy, the problem of shared redundancy management becomes more complicated and challenging which would be an interesting direction. Another direction is redundancy virtualization to define redundancy blocks as virtual resources that can be managed in software level.
- In general, the application of Cross-layer memory resiliency in a real system would bring interesting challenges and opportunities to the software stack.

# Bibliography

- [1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González. Low vccmin fault-tolerant cache with highly predictable performance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 111–121. ACM, 2009.
- [2] A. Agarwal, B. C. Paul, H. Mahmoodi-Meimand, A. Datta, and K. Roy. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Trans. VLSI Syst.*, 13(1):27–38, 2005.
- [3] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *ISCA*, 2007.
- [4] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu. Energy-efficient cache design using variable-strength error-correcting codes. In *Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11*, pages 461–472, 2011.
- [5] P. Ampadu, M. Zhang, and V. Stojanovic. Breaking the energy barrier in fault-tolerant caches for multicore systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 731–736. IEEE, 2013.
- [6] B. Amrutur and M. Horowitz. Speed and power scaling of sram's. *IEEE Journal of Solid-State Circuits*, 35(2):175–185, Feb 2000.
- [7] F. Angiolini, D. Atienza, S. Murali, L. Benini, and G. D. Micheli. Reliability support for on-chip memories using networks-on-chip. In *ICCD*, 2006.
- [8] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *ISLPED*, 2009.
- [9] A. Ansari, S. Feng, S. Gupta, and S. A. Mahlke. Archipelago: A polymorphic cache design for enabling robust near-threshold operation. In *HPCA*, 2011.
- [10] A. Ansari, S. Gupta, S. Feng, and S. A. Mahlke. Zerehcache: armoring cache architectures in high defect density technologies. In *MICRO*, 2009.
- [11] ARM. *Cortex-A5 processor manual*. <http://www.arm.com>.

- [12] ARM. *Cortex-R4 processor manual*. <http://www.arm.com>.
- [13] ASU. *Predictive Technology Model (PTM)*. <http://ptm.asu.edu>.
- [14] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.
- [15] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.
- [16] A. Banaiyanmofrad. Reliable on-chip memory design for cmps. In *Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems*, pages 487–488. IEEE Computer Society, 2012.
- [17] A. BanaiyanMofrad, N. Dutt, and G. Girão. Analyzing and exploring fault-tolerant distributed memories for nocs. Technical Report CECS-TR-12-15, Center for Embedded Computer Systems, University of California, Irvine, December 2012.
- [18] A. Banaiyanmofrad, M. Ebrahimi, F. Oboril, M. B. Tahoori, and N. Dutt. Protecting caches against multiple bit upsets using embedded erasure coding. In *European Test Symposium (ETS)*. IEEE, 2015.
- [19] A. BanaiyanMofrad, G. Girao, and N. Dutt. A novel noc-based design for fault-tolerance of last-level caches in cmps. In *CODES+ISSS*, 2012.
- [20] A. BanaiyanMofrad, G. Girao, and N. Dutt. Modeling and analysis of fault-tolerant distributed memories for networks-on-chip. In *DATE*, 2013.
- [21] A. Banaiyanmofrad, G. Girão, and N. Dutt. Noc-based fault-tolerant cache design in chip multiprocessors. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):115, 2014.
- [22] A. BanaiyanMofrad, H. Homayoun, and N. Dutt. Fft-cache: a flexible fault-tolerant cache architecture for ultra low voltage operation. In *CASES*, 2011.
- [23] A. Banaiyanmofrad, H. Homayoun, and N. Dutt. Using a flexible fault-tolerant cache to improve reliability for ultra low voltage operation. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(2):32, 2015.
- [24] A. BanaiyanMofrad, H. Homayoun, V. Kontorinis, D. Tullsen, and N. Dutt. Remediate: A scalable fault-tolerant architecture for low-power nuca cache in tiled cmps. In *IGCC*, 2013.
- [25] L. A. D. Bathen and N. D. Dutt. E-roc: Embedded raids-on-chip for low power distributed dynamically managed reliable memories. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.
- [26] L. A. D. Bathen, N. D. Dutt, A. Nicolau, and P. Gupta. VaMV: Variability-aware memory virtualization. In *DATE*, 2012.



- [27] L. A. D. Bathen, N. D. Dutt, D. Shin, and S.-S. Lim. Spmvisor: dynamic scratchpad memory virtualization for secure, low power, and high performance distributed on-chip memories. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 79–88. ACM, 2011.
- [28] L. A. D. Bathen, M. Gottscho, N. Dutt, A. Nicolau, and P. Gupta. ViPZonE: Os-level memory variability-driven physical address zoning for energy savings. In *CODES+ISSS*, 2012.
- [29] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.
- [30] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *MICRO*, pages 319–330. IEEE Computer Society, 2004.
- [31] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, et al. Tile64-processor: A 64-core soc with mesh interconnect. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 88–598. IEEE, 2008.
- [32] D. Bertozzi, L. Benini, and G. D. Micheli. Error control schemes for on-chip communication links: The energy-reliability tradeoff. *IEEE Trans. CAD*, 24(6):818–831, 2000.
- [33] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81, 2008.
- [34] N. Binkert and et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.
- [35] N. L. Binkert and et al. The m5 simulator: Modeling networked systems. *IEEE Micro*, 2006.
- [36] P. Bogdan, T. Dumitras, and R. Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. In *VLSI Design*, 2007.
- [37] S. Y. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.
- [38] B. H. Calhoun and A. Chandrakasan. A 256kb sub-threshold sram in 65nm cmos. In *ISSCC*, 2006.
- [39] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA*, 2013.
- [40] T. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov 2011.

- [41] A. Chakraborty and et al. E < mc2: less energy through multi-copy cache. In *CASES*, 2010.
- [42] 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. In *VLSI Technology, 2005. Digest of Technical Papers. 2005 Symposium on*, pages 128–129, June 2005.
- [43] C. Chen and M. Hsiao. Error-correcting codes for semiconductor memory applications: A state of the art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.
- [44] G. K. Chen, D. Sylvester, D. Blaauw, and T. N. Mudge. Yield-driven near-threshold sram design. *IEEE Trans. VLSI Syst.*, 2010.
- [45] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, 2013.
- [46] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. In *MICRO*, 2009.
- [47] S. Cho and H. Lee. Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 347–357. IEEE, 2009.
- [48] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, July 2003.
- [49] R. Das, A. K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. S. Yousif, and C. R. Das. Performance and power optimization through data compression in network-on-chip architectures. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 215–225. IEEE, 2008.
- [50] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA*, 2010.
- [51] M. de Kruijf and K. Sankaralingam. Exploring the synergy of emerging workloads and silicon reliability trends. In *SELSE*, 2009.
- [52] M. Dehyadgari, M. Nickray, A. Afzali-Kusha, and Z. Navabi. Evaluation of pseudo adaptive xy routing using an object oriented model for noc. In *Microelectronics, 2005. ICM 2005. The 17th International Conference on*, pages 5–pp. IEEE, 2005.
- [53] A. Dixit and A. Wood. The impact of new technology on soft error rates. In *Proc. of the Inter. Reliability Physics Symposium*, 2011.
- [54] N. Dutt, P. Gupta, A. Nicolau, A. BanaiyanMofrad, M. Gottscho, and M. Shoushtari. Multi-layer memory resiliency. In *Proceedings of the 51st Annual Design Automation Conference, DAC '14*, pages 48:1–48:6, 2014.

- [55] M. Ebrahimi, A. Evans, M. Tahoori, E. Costenaro, D. Alexandrescu, V. Chandra, and R. Seyyedi. Comprehensive analysis of sequential and combinational soft errors in an embedded processor. *IEEE TCAD*, 2015.
- [56] M. Ebrahimi, A. Evans, M. Tahoori, R. Seyyedi, E. Costenaro, and D. Alexandrescu. Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales. In *Design, Automation and Test in Europe*, 2014.
- [57] A. Eghbal, P. M. Yaghini, H. Pedram, and H. Zarandi. Designing fault-tolerant network-on-chip router architecture. *International Journal of Electronics*, 97(10):1181–1192, 2010.
- [58] H. Esmailzadeh, E. R. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, pages 365–376, 2011.
- [59] H. Esmailzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ASPLOS*, 2012.
- [60] J. Gaisler. Leon 3 synthesizable processor.
- [61] G. Girão, D. Barcelos, and F. R. Wagner. Performance and energy evaluation of memory organizations in noc-based mpsoCs under latency and task migration. In *VLSI-SoC: Technologies for Systems Integration*, pages 56–80. Springer, 2011.
- [62] M. Glass, M. Lukasiewicz, T. Streichert, C. Haubelt, and J. Teich. Reliability-aware system synthesis. In *DATE*, pages 1–6, April 2007.
- [63] M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, and P. Gupta. Power/capacity scaling: Energy savings with simple fault-tolerant caches. In *Design Automation Conference (DAC)*, 2014.
- [64] M. Gries, U. Hoffmann, M. Konow, and M. Riepen. Scc: A flexible architecture for many-core platform research. *Computing in Science and Engineering*, 13(6):79–83, 2011.
- [65] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *WWC*, 2001.
- [66] J. L. Hafner. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *Proc. of the 4th USENIX Conference on File and Storage Technologies, FAST*, 2005.
- [67] R. W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.
- [68] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 2011.

- [69] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 99. ACM, 2013.
- [70] H. P. Hofstee. Power efficient processor architecture and the cell processor. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 258–262. IEEE, 2005.
- [71] M. Hosseinabady, A. Banaiyan, M. N. Bojnordi, and Z. Navabi. A concurrent testing method for noc switches. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 1171–1176. European Design and Automation Association, 2006.
- [72] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, et al. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173–183, 2011.
- [73] M. Y. Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM J. Reserach and Development*, 25(14):295–301, 1970.
- [74] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *Proc. of the USENIX Conference*, 2012.
- [75] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler. A nuca substrate for flexible cmp cache sharing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(8):1028–1040, Aug 2007.
- [76] E. Ibe, H. Taniguchi, Y. Yahagi, K.-i. Shimbo, and T. Toba. Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule. *IEEE Trans. on Electron Devices*, 57(7), 2010.
- [77] ITRS. *International technology roadmap for semiconductors*, 2011.
- [78] J. Jeffers and J. Reinders. *Intel Xeon Phi coprocessor high-performance programming*. Newnes, 2013.
- [79] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428. European Design and Automation Association, 2009.
- [80] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son. A novel migration-based nuca design for chip multiprocessors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 28. IEEE Press, 2008.

- [81] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. *SIGARCH Comput. Archit. News*, 30(5):211–222, 2002.
- [82] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *MICRO*, pages 197–209, Dec 2007.
- [83] J. Kim, C. Nicopoulos, and D. Park. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *ISCA*, 2006.
- [84] J. Kim, H. Yang, M. McCartney, M. Bhargava, K. Mai, and B. Falsafi. Building fast, dense, low-power caches using erasure-based inline multi-bit ecc. In *Pacific Rim International Symposium on Dependable Computing*, pages 98–107, Dec 2013.
- [85] W. Klotz. Graph coloring algorithms. *Mathematics Report*, 5(2002):1–9, 2002.
- [86] C.-K. Koh, W.-F. Wong, Y. Chen, and H. Li. The salvage cache: A fault-tolerant cache architecture for next-generation memory technologies. In *ICCD*, 2009.
- [87] C.-K. Koh, W.-F. Wong, Y. Chen, and H. Li. Tolerating process variations in large, set-associative caches: The buddy cache. *ACM Trans. Archit. Code Optim.*, 6(2):8:1–8:34, July 2009.
- [88] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *Micro, IEEE*, 25(2):21–29, March 2005.
- [89] V. Kozhikkottu, A. Pan, V. Pai, S. Dey, and A. Raghunathan. Variation aware cache partitioning for multithreaded programs. In *DAC*, 2014.
- [90] J. P. Kulkarni, K. Kim, and K. Roy. A 160 mv, fully differential, robust schmitt trigger based sub-threshold sram. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design, ISLPED '07*, pages 171–176, 2007.
- [91] L. Kunz, G. Girão, and F. R. Wagner. Improving the efficiency of a hardware transactional memory on an noc-based mp soc. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–4. IEEE, 2011.
- [92] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *DATE*, 2010.
- [93] K. M. Lepak and M. H. Lipasti. Silent stores for free. In *MICRO*, pages 22–31, 2000.
- [94] X. Li, S. Adve, P. Bose, and J. Rivers. Softarch: an architecture-level tool for modeling and analyzing soft errors. In *DSN*, pages 496–505, June 2005.
- [95] X. Li and D. Yeung. Application-level correctness and its impact on fault tolerance. In *HPCA*, 2007.
- [96] S. Lin and D. J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

- [97] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. Nvidia tesla: A unified graphics and computing architecture. *Ieee Micro*, 28(2):39–55, 2008.
- [98] M. Lis and et al. Scalable, accurate multicore simulation in the 1000-core era. In *ISPASS*, 2011.
- [99] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent DRAM refresh. In *ISCA*, 2012.
- [100] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flicker: Saving DRAM refresh-power through critical data partitioning. In *ASPLOS*, 2011.
- [101] M. Lukasiewicz, M. Glass, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pages 691–696, March 2008.
- [102] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.
- [103] N. Mahatme, B. Bhuvva, Y.-P. Fang, and A. Oates. Analysis of multiple cell upsets due to neutrons in srams for a deep-n-well process. In *International Reliability Physics Symposium*, 2011.
- [104] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of multi-bit soft error events in advanced srams. In *IEEE International Electron Devices Meeting*, Dec 2003.
- [105] S. Manolache, P. Eles, and Z. Peng. Fault and energy-aware communication mapping with guaranteed latency for applications implemented on noc. In *DAC*, 2005.
- [106] M. Manoochehri, M. Annavaram, and M. Dubois. Cppc: Correctable parity protected cache. In *38th Annual International Symposium on Computer Architecture*, pages 223–234, 2011.
- [107] R. Marculescu, m. Y. Ogras, L.-S. Peh, N. D. E. Jerger, and Y. V. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(1):3–21, 2009.
- [108] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. Rinard. Reliability-aware optimization of approximate computational kernels with Rely. Technical Report MIT-CSAIL-TR-2014-001, MIT, January 2014.
- [109] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. C. Rinard. Quality of service profiling. In *ICSE*, 2010.
- [110] A. M. H. Monazzah, H. Farbeh, S. G. Miremadi, M. Fazeli, and H. Asadi. Ftspm: a fault-tolerant scratchpad memory. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–10. IEEE, 2013.

- [111] F. Moradi, D. Wisland, S. Aunet, H. Mahmoodi, and T. V. Cao. 65nm sub-threshold 1t1r1w sram for ultra low voltage applications. In *SOC Conference, 2008 IEEE International*, pages 113–118, Sept 2008.
- [112] R. H. Morelos-Zaragoza. *The Art of Error Correcting Coding*. John Wiley & Sons, 2006.
- [113] Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto. An area-conscious low-voltage-oriented 8t1r1w sram design under dvs environment. In *VLSI Circuits, 2007 IEEE Symposium on*, pages 256–257, June 2007.
- [114] S. Murali, D. Atienza, L. Benini, and G. De Michel. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of the 43rd annual Design Automation Conference*, pages 845–848. ACM, 2006.
- [115] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *Design Test of Computers, IEEE*, 22(5):434–442, 2005.
- [116] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Cacti 6.5. In *Technical Report, HP Laboratories*, 2009.
- [117] R. Naseer and J. Draper. Parallel double error correcting code design to mitigate multi-bit upsets in srams. In *34th European Solid-State Circuits Conference*, pages 222–225, Sept 2008.
- [118] S. R. Nassif, N. Mehta, and Y. Cao. A resilience roadmap: (invited paper). In *DATE*, pages 1011–1016, 2010.
- [119] P. Ndai, A. Goel, and K. Roy. A scalable circuit-architecture co-design to improve memory yield for high-performance processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(8):1209–1219, 2010.
- [120] H. A. Omari and K. E. Sabri. New graph coloring algorithms. *Journal of Mathematics and Statistics*, 2(4):439–441, 2007.
- [121] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara. Sram immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect. *JSSC*, 39(5):827–833, May 2004.
- [122] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 15–25, 2006.
- [123] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das. Exploring fault-tolerant network-on-chip architectures. In *DSN*, pages 93–104, 2006.

- [124] S. Park, T. Krishna, C.-H. Chen, B. Daya, A. Chandrakasan, and L.-S. Peh. Approaching the theoretical limits of a mesh noc with a 16-node chip prototype in 45nm soi. In *Proceedings of the 49th Annual Design Automation Conference, DAC '12*, pages 398–405, 2012.
- [125] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proc. of the ACM Intrn. Conf. on Management of Data, SIGMOD*, pages 109–116, 1988.
- [126] S. Paul, F. Cai, X. Zhang, and S. Bhunia. Reliability-driven ecc allocation for multiple bit error resilience in processor cache. *Computers, IEEE Transactions on*, 60(1):20–34, 2011.
- [127] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *IEEE Symp. VLSI*, 2004.
- [128] J. S. Plank. T1: erasure codes for storage applications. In *Proc. of the 4th USENIX Conference on File and Storage Technologies*, pages 1–74, 2005.
- [129] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunit: A cheap and robust fault-tolerant packet routing mechanism. In *ISCA*, 2004.
- [130] P. Rao, M. Ebrahimi, R. Seyyedi, and M. B. Tahoori. Protecting sram-based fpgas against multiple bit upsets using erasure codes. In *DAC*, pages 1–6. IEEE, 2014.
- [131] D. Roberts, N. S. Kim, and T. N. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *Microprocessors and Microsystems - Embedded Hardware Design*, 2008.
- [132] D. Rossi, N. Timoncini, M. Spica, and C. Metra. Error correcting code analysis for cache memory high reliability and performance. In *Design, Automation Test in Europe (DATE)*, pages 1–6, March 2011.
- [133] K. R. S. Mukhopadhyay, H. Mahmoodi. Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscaled cmos. In *TCADICS*, 2005.
- [134] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI*, 2011.
- [135] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. In *MICRO*, 2013.
- [136] J. Sartori, J. Sloan, and R. Kumar. Stochastic computing: Embracing errors in architecture and design of processors and applications. In *CASES*, 2011.
- [137] A. Sasan, H. Homayoun, A. M. Eltawil, and F. Kurdahi. Inquisitive defect cache: A means of combating manufacturing induced process variation. *IEEE Trans. Very Large Scale Integr. Syst.*, 19(9):1597–1609, Sept. 2011.



- [138] A. Sasan, H. Homayoun, A. M. Eltawil, and F. J. Kurdahi. A fault tolerant cache architecture for sub 500mv operation: resizable data composer cache (rdc-cache). In *CASES*, 2009.
- [139] S. Schuster. Multiple word/bit line redundancy for semiconductor memories. *Solid-State Circuits, IEEE Journal of*, 13(5):698–703, Oct 1978.
- [140] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, et al. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics (TOG)*, 27(3):18, 2008.
- [141] S. Shamschiri, A.-A. Ghofrani, and K.-T. Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *IEEE International Test Conference (ITC)*, pages 1–10, Sept 2011.
- [142] J. L. Shin, D. Huang, B. Petrick, C. Hwang, A. S. Leon, and A. Strong. A 40nm 16-core 128-thread sparc® soc processor. In *Solid State Circuits Conference (A-SSCC), 2010 IEEE Asian*, pages 1–4. IEEE, 2010.
- [143] P. P. Shirvani and E. J. McCluskey. Padded cache: A new fault-tolerance technique for cache memories. In *VTS*, 1999.
- [144] M. Shoushtari, A. Banaiyan, and N. Dutt. Relaxing manufacturing guard-bands in memories for energy saving. Technical Report CECS-TR-14-04, Center for Embedded Computer Systems, University of California, Irvine, May 2014.
- [145] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt. Exploiting partially-forgetful memories for approximate computing. *Embedded Systems Letters, IEEE*, 7(1):19–22, March 2015.
- [146] C. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *Device and Materials Reliability, IEEE Transactions on*, 5(3):397–404, Sept 2005.
- [147] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. *ACM SIGPLAN Notices*, 35(11):234–244, 2000.
- [148] V. Sridharan, H. Asadi, M. B. Tahoori, and D. Kaeli. Reducing data cache susceptibility to soft errors. *Dependable and Secure Computing, IEEE Transactions on*, 3(4):353–364, 2006.
- [149] J. Stevens, P. Tschirhart, M.-T. Chang, I. Bhati, P. Enns, J. Greensky, Z. Chishti, S.-L. Lu, and B. Jacob. Memory resiliency. *Intel Technology Journal*, 17(1), May 2013.
- [150] M. Taassori, N. Chatterjee, A. Shafiee, and R. Balasubramonian. Exploring a brink-of-failure memory controller to design an approximate memory system. In *WACAS*, 2014.

- [151] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, et al. An 80-tile 1.28 tflops network-on-chip in 65nm cmos. In *IEEE International Solid-State Circuits Conference, San Fransisco, USA, 2007*, pages 98–99. IEEE, 2007.
- [152] N. Verma and A. P. Chandrakasan. A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy. *Solid-State Circuits, IEEE Journal of*, 43(1):141–149, 2008.
- [153] J. Wang and B. H. Calhoun. Standby supply voltage minimization for reliable nanoscale SRAMs. In *Solid State Circ. Tech.* InTech, 2004.
- [154] Y. Wang, L. Zhang, Y. Han, H. Li, and X. Li. Address remapping for static nuca in noc-based degradable chip-multiprocessors. In *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*, 2010.
- [155] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA*, pages 83–93, 2010.
- [156] 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. In *Proceedings of the 35th Annual International Symposium on Computer Architecture, ISCA '08*, pages 203–214, 2008.
- [157] L. Wilson. International technology roadmap for semiconductors (itrs). *Semiconductor Industry Association*, 2013.
- [158] W.-F. Wong, C.-K. Koh, Y. Chen, and H. Li. Vosch: Voltage scaled cache hierarchies. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*, pages 496–503, Oct 2007.
- [159] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. *ACM SIGARCH Computer Architecture News*, 23(2):24–36, 1995.
- [160] P. M. Yaghini, A. Eghbal, H. Pedram, and H. R. Zarandi. Investigation of transient fault effects in synchronous and asynchronous network on chip router. *Journal of Systems Architecture*, 57(1):61 – 68, 2011. Special Issue On-Chip Parallel And Network-Based Systems.
- [161] D. H. Yoon and M. Erez. Memory mapped ecc: low-cost error protection for last level caches. In *ISCA*, 2009.
- [162] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez. Free-p: Protecting non-volatile memory against both hard and soft errors. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 466–477. IEEE, 2011.

- [163] C. A. Zeferino and A. A. Susin. Socin: a parametric and scalable network-on-chip. In *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pages 169–174. IEEE, 2003.
- [164] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems. *ACM Trans. Embed. Comput. Syst.*, 4(2):363–387, May 2005.
- [165] W. Zhang. Replication cache: a small fully associative cache to improve data cache reliability. *Computers, IEEE Transactions on*, 54(12):1547–1555, 2005.
- [166] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. Icr: in-cache replication for enhancing data cache reliability. In *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, pages 291–300, June 2003.
- [167] W. Zhang and T. Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *MICRO*, 2009.
- [168] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. Energy reduction for stt-ram using early write termination. In *Computer-Aided Design-Digest of Technical Papers, 2009. IC-CAD 2009. IEEE/ACM International Conference on*, pages 264–268. IEEE, 2009.
- [169] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim. Minimizing total area of low-voltage sram arrays through joint optimization of cell size, redundancy, and ecc. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 112–117. IEEE, 2010.