Achieving Reliable and Sustainable Next-Generation Memories

by

Donald E. Kline, Jr

M.S. Electrical Engineering, University of Pittsburgh, 2017

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Donald E. Kline, Jr

It was defended on

July 9th, 2019

and approved by

Alex K. Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

Rami Melhem, Ph.D., Professor
Department of Computer Science

Jun Yang, Ph.D., Professor
Department of Electrical and Computer Engineering

Jingtong Hu, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Feng Xiong, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Dissertation Director: Alex K. Jones, Ph.D., Professor
Department of Electrical and Computer Engineering

Achieving Reliable and Sustainable
Next-Generation Memories

Donald E. Kline, Jr, PhD

University of Pittsburgh, 2019

Conventional memory technology scaling has introduced reliability challenges due to dysfunctional, improperly formed cells and crosstalk from increased cell proximity. Furthermore, as the manufacturing effort becomes increasingly complex due to these deeply scaled technologies, holistic sustainability is negatively impacted. The development of new memory technologies can help overcome the capacitor scaling limitations of DRAM. However, these technologies have their own reliability concerns, such as limited write endurance in the case of Phase Change Memories (PCM). Moreover, emerging system requirements, such as in-memory encryption to protect sensitive or private data and operation in harsh environments create additional challenges that must be addressed in the context of reliability and sustainability. This dissertation provides new multifactor and ultimately unified solutions to address many of these concerns in the same system.

In particular, my contributions toward mitigating these issues are as follows. I present GreenChip and GreenAsic, which together provide the first tools to holistically evaluate new computer architecture, chip, and memory design concepts for sustainability. These tools provide detailed estimates of manufacturing and operational-phase metrics for different computing workloads and deployment scenarios. Using GreenChip, I examined existing DRAM reliability techniques in the context of their holistic sustainability impact, including my own technique to mitigate bitline crosstalk. For PCM, I provided a new reliability technique with no additional storage overhead that substantially increases the lifetime of an encrypted memory system. To provide bit-level error correction, I developed compact linked-list and Bloom-filter-based bit-level fault map structures, that provide unprecedented levels of error tabulation, combined with my own novel error correction and lifetime extension approaches based on these maps for less area than traditional ECC. In particular, FaME, can correct N faults using N bits when utilizing a bit-level fault map. For operation in

harsh environments, I created a triple modular redundancy (TMR) pointer-based fault map, HOTH, which specifically protects cells shown to be weak to radiation. Finally, to combine the analyses of holistic sustainability and memory lifetime, I created the LARS technique, which adjusts the GreenChip indifference analysis to account for the additional sustainability benefit provided by increased reliability and lifetime.

Table of Contents

Preface . . . xvii

1.0 Introduction . . . 1

1.1 Contributions . . . 1

1.1.1 PART I: Holistic Sustainability Solutions . . . 1

1.1.1.1 GreenChip . . . 1

1.1.1.2 GreenASIC . . . 2

1.1.1.3 LARS . . . 2

1.1.2 PART II: Bit-level Fault Maps . . . 3

1.1.2.1 SFaultMap . . . 3

1.1.2.2 FLOWER . . . 4

1.1.2.3 HOTH . . . 4

1.1.3 PART III: Encoding Solutions for Mitigating Memory Faults . . . 5

1.1.3.1 PFE: Periodic Flip Encoding . . . 5

1.1.3.2 Counter Advance . . . 6

1.1.3.3 FaME . . . 7

2.0 Background . . . 8

2.1 Memory Reliability . . . 8

2.1.1 DRAM . . . 8

2.1.2 PCM . . . 9

2.1.3 Existing Fault Maps . . . 10

2.2 Holistic Sustainability . . . 10

2.2.1 LCA of Computing Systems . . . 12

2.2.2 Impacts from IC Fabrication . . . 12

2.2.3 Holistic Sustainability Related Work . . . 14

2.3 Miscellaneous . . . 15

2.3.1 Memory Encryption . . . 15

2.3.2 Bloom Filters . . . 15
3.0 GreenChip . . . 20
3.1 The GreenChip Sustainable Computing Prediction and Evaluation Tool . . . 20
3.2 Case Study I: Environmental Impacts of Recent Processor Trends . . . 22
3.2.1 Experimental Setup . . . 23
3.2.2 Results . . . 24
3.3 Case Study II: Sensitivity Analysis of the Impact of Cache Sizes on Sustainability . . . 25
3.4 Case Study III: Impact of Main Memory Density on Sustainability . . . 27
3.4.1 Single Benchmark Detailed Analysis . . . 27
3.4.2 Multiple Benchmark Analysis . . . 29
3.4.3 Improving Cloud Server Utilization . . . 30
3.5 Conclusion and Future Work . . . 32
4.0 GreenASIC . . . 48
4.1 Environmental Impact Model . . . 48
4.1.1 Parameterizing the Fabrication Process . . . 48
4.1.2 Scaling the Parameterized Model . . . 51
4.2 Results . . . 53
4.2.1 Experimental Setup . . . 53
4.2.2 Minimizing Purely Manufacturing Impacts . . . 53
4.2.3 Optimizing for Holistic Sustainability . . . 54
4.2.4 Additional Sustainability Reports . . . 60
4.3 Conclusions and Future Work . . . 60
5.0 LARS Indifference Analysis . . . 62
5.1 LARS Concept and Implementation . . . 62
5.2 LARS Case Study . . . 63
5.3 Conclusion . . . 64
6.0 SFaultMap . . . 66
6.1 SFaultMap Design . . . 66
6.1.1 Performance Improvement 1: Offset Segment Lookup . . . 68

6.1.2 Performance Improvement 2: Zero-fault Bit . . . 68
6.1.3 Fault Map Extensions and Discussion . . . 68
6.2 Experimental Setup . . . 69
6.3 Evaluation . . . 70
6.3.1 Area Overheads and Embodied Energy . . . 71
6.3.2 Runtime Overheads and Operational Energy . . . 72
6.3.3 Holistic Energy Analysis . . . 75
6.4 Conclusion . . . 77
7.0 FLOWER . . . 79
7.1 The FLOWER Fault Map . . . 79
7.1.1 MinCI: A Tuned Hash for FLOWER . . . 81
7.1.2 FLOWER Architecture . . . 83
7.1.2.1 In-Memory FLOWER Access for DRAM . . . 85
7.1.2.2 In-Memory FLOWER for PCM . . . 86
7.1.2.3 Updating the Fault Map . . . 87
7.1.3 Implementation Details . . . 88
7.1.3.1 Improving Performance . . . 88
7.1.3.2 Storing FLOWER Reliably . . . 88
7.2 FLOWER for Fault Tolerance . . . 89
7.2.1 DRAM Bitline Crosstalk . . . 90
7.2.2 PCM Stuck-at Faults . . . 91
7.3 Results . . . 91
7.3.1 FLOWER for Enhanced Fault Tolerance . . . 92
7.3.1.1 Bitline Crosstalk Correction (DRAM) . . . 92
7.3.1.2 Stuck-at Fault Correction (PCM) . . . 93
7.3.2 Performance Impact (DRAM) . . . 93
7.4 Conclusion . . . 95
8.0 HOTH and Neutron Radiation Experiments . . . 99
8.1 The HOTH Faultmap . . . 99
8.2 Procedure . . . 101

8.3 Results . . . 102
8.3.1 Experimental Results . . . 102
8.3.2 HOTH Fault Tolerance Results . . . 103
8.3.3 Correlation Non-Radiation Reliability Concerns . . . 105
8.3.4 Secondary Qualitative Results . . . 105
8.4 Conclusions . . . 106
9.0 PFE: Periodic Flip Encoding . . . 112
9.1 Design . . . 112
9.1.1 Fault Oblivious PFE (PFE_{FO}) . . . 112
9.1.2 Fault-Aware PFE (PFE_{FA}) . . . 113
9.2 Memory Controller Implementation . . . 114
9.3 Tolerance Capability . . . 115
9.4 Experimental Methodology . . . 118
9.5 Results . . . 120
9.5.1 Fault-Oblivious Effectiveness . . . 120
9.5.2 Fault-Aware Effectiveness . . . 122
9.5.3 Impact on Performance . . . 122
9.5.4 Comparison of different fault mitigation schemes . . . 123
9.5.5 Sensitivity to block size . . . 124
9.6 Conclusion . . . 125
10.0 Counter Advance . . . 128
10.1 Design . . . 128
10.2 Evaluation . . . 130
10.3 Alternative Implementations . . . 132
10.4 Related Work . . . 132
10.5 Conclusion . . . 133
11.0 FaME . . . 137
11.1 FaME Error correction . . . 137
11.2 FaME Design . . . 137
11.3 Accumulated Faults . . . 139

11.3.1 Memory Update Queue 140
11.3.2 PETALs: In-Memory FLOWER Correction 141
11.4 FaME Lifetime Improvement for PCM 142
11.5 Conclusion 144
12.0 Overall Conclusions 146
Bibliography 147

List of Tables

1 Multi-program Workloads and Memory Footprints for the Parsec and SPEC Benchmarks. Low (L), Medium (M), and High (H) represents those respective memory footprints. 35

2 Activity and Sleep Scenarios 36

3 Manufacturing costs for chips at different process nodes following product trends (pseudo ISO-area) [1, 2]. 36

4 Disk-Stressing Benchmarks 42

5 Parameterized model [3] with 130nm process LCA data [2] differentiating 10, 8, and 6-layer metal stacks for a 200mm wafer. 49

6 Process steps and scaled 300mm wafer energy for 90nm, 65nm, 45nm, and 32nm processes based on the IC model from Figure 22. 50

7 Fabrication savings when using the minimum possible area. 54

8 Results from an exploration of the effects of using eight metal layers versus six metal layers in the same process. Results are within 6% of max cell utilization. 55

9 Report on sustainability metrics, eight metal layers. 59

10 Report on sustainability metrics, six metal layers. 59

11 Architecture parameters. 70

12 Architecture Parameters 98

13 Devices under test 109

14 Results from neutron radiation experiment 109

15 PFE transformations of 3-bit sequences. 118

16 Simulator Parameters 119

17 Bit overheads for fault tolerance schemes. 120

18 The overhead of different schemes with latency optimization and power optimization. 127

19 UBER for in-memory combination protected with PETALs versus memory controller combination protected with ECC-1 for a 4D FLOWER fault map. 142

List of Figures

1 Impact of manufacturing/production from ICs in "use phase energy" optimized systems. 18

2 IC fabrication energy and global warming potential (GWP) trends. 18

3 Counter mode encryption in the memory controller. 19

4 GreenChip evaluation flow. 34

5 Joules Per 10billion instructions for the Parsec and SPEC multiprogram workloads with different process node. All are run with the same chip area, as part of the iso-area comparison. 37

6 Indifference points (t_I) for the pseudo ISO-area comparisons across adjacent technology nodes for GLMN highlighting four usage scenarios. 38

7 Break even time (t_B) to move to the next technology node in a pseudo ISO-area comparison for GLMN highlighting four usage scenarios. 39

8 Average break even times and indifference points across all benchmarks for pseudo iso-area comparison. 40

9 Manufacturing energy for four-core systems with varying LLC capacities across technology nodes. 40

10 Indifference time (t_I) between 45nm and 28nm for multiple LLC cache capacities. 40

11 Average break even times across all the benchmarks, iso-architecture comparison with 4 cores. Note: All 90nm vs. 65nm data points for desktop except one benchmark never broke even. Also, one benchmark for the server at 0.5MB never broke even, so the average is the remainder of the benchmarks 41

12 Global IPC and MPKI averaged for Parsec and SPEC four process multiprogram workloads across different technology nodes while varying LLC capacity. (min and max shown by error bars) 41

13 Joules Per 10 billion instructions averaged for Parsec and SPEC four process multi-program workloads across different technology nodes while varying LLC capacity. 42
14 Memory indifference points for the NAS Parallel Benchmark "sp." 43
15 Indifference times (years) for 4GB vs. 8GB comparisons at 55nm. Note that the scale is different from Figure 16. 44
16 Indifference times (years) for 4GB vs. 8GB comparisons at 65nm. Note that the scale is different from Figure 15. 44
17 Indifference times (years) for 65nm vs. 55nm comparisons at 4GB. 45
18 An example of the impact of the different activity ratio options and their corresponding impacts on indifference point for servers. The circle represents the original activity ratio and the arrow shows increased activity ratios to achieve ten, five, and three year indifference points. 45
19 Indifference times (years) for the full range of server activity ratios at the 55nm technology node for the comparison between 4GB and 8GB main memory sizes. Applu and bwaves always have an infinite indifference time. 46
20 Indifference times (years) for the full range of server activity ratios at the 65nm technology node for the comparison between 4GB and 8GB main memory sizes. Applu and bwaves always have an infinite indifference time. 46
21 Indifference times (years) for the full range of server activity ratios at the 4GB memory size for the comparison between 65nm and 55nm technology nodes. Cg's curve begins at 33 years, and ilbdc always has an infinite indifference time. 47
22 Total energy consumption across the different process steps for technologies between 130nm and 32nm as calculated by our process model. The total across all steps [2] (patterned bars of the same colors) use the secondary axis. . . . 50
23 Mobile scenario eight vs. six layer indifference time (years). 57
24 Cloud scenario eight vs. six layer indifference time (years). 57
25 Desktop scenario eight vs. six layer indifference time (years). 58
26 HPC scenario eight vs. six layer indifference time (years). 58

27 Indifference points (t_I) for the traditional GreenChip tool for ECP6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted. . 64
28 LARS indifference points (t_I) for the GreenChip tool for ECP6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted. 65
29 Encoding strategy for data-agnostic bit-level fault map. 67
30 Example of the data-agnostic fault map, with a row-segment size of 10 bits and a row size of 8 bits (3 bits per pointer). Even rows are orange. 67
31 IC (Area) Overhead for ECP, ArchShield, and the Fault Map at different initial weak cell rates. 71
32 Energy consumption for decoding ECP and the improved fault map at different sizes and initial fault rates. 10^{-2} cannot have a fault map size of 192 or lower because certain rows require at least that many bits alone. 73
33 IPC impact of 256 bit block fault map designs across SPEC benchmarks normalized to a fault-free IPC at different fault incidence rates. 74
34 Average IPC across SPEC benchmarks, relative to a system without a fault map. 75
35 ECP vs. SFaultMap indifference times for different initial fault ratios, usage scenarios, and row-segment sizes. 76
36 Indifference times SFaultMap and SFaultMap+. SFaultMap has the lower embodied energy. 77
37 Two-Dimensional, unified-array FLOWER example. Red represents faulty cells. M is the length of each row, N are the bits in the address space for the memory, and k are the bits which result from the hash function 80
38 4 masks for MinCi hash functions assuming $N = 9$ address bits, $d = 4$ dimensions, and $h = 4$ hash bits. 84
39 False positive rates for 4D fault maps with different storage overheads and initial fault rates. 84
40 DRAM Organization for FLOWER using in-memory logical operations. . . . 85
41 In-memory 4D FLOWER reading example. 86
42 Encoding “0x5D5” into a 12-bit (11...0) with weak cells as positions 7, 5, and 3, using PFE. 90

43 Iso-area PFE correction capability at 10^{-3} incidence weak-cell rate.Total overhead including fault map and encoding bits shown in "[]." 96

44 Iso-area analysis of UBER for ECP and ECP's extension, Yoda with FLOWER at 10^{-3} fault incidence rate. Colors indicate different fault map allocations: Orange: 6.25%, Green: 3.13%, Purple: 1.56%, Blue: 0%. 97

45 IPC over 9 SPEC Benchmarks at 6.25% area overhead at different error rates (10^{-6} to 10^{-2}) for FLOWER. Striped bars indicate IPC improvement from using in-memory operations. 97

46 Average IPC over 9 SPEC Benchmarks for different fault map area overheads (0.39% to 12.5%) and error rates (10^{-6} to 10^{-2}). Striped bars indicate IPC improvement from using in-memory operations. 98

47 Architecture for the HOTH fault map. A masked physical address is hashed to table rows, each bin has 6 "ways." Each way uses remaining physical address to test "tag" bits for valid "V" entries. The payload is the appropriate pointer and spare bit. Entries are protected with TMR as indicated. 107

48 Neutron beam test at Los Alamos Neutron Science Center (LANSCE). . . . 108

49 Detailed rig with DIMMs parallel to beam path highlighted in white. 108

50 Weak and stuck-at cell heatmap (T=10) over groups of rows for 16 adjacent 32 bit word accesses. Rows split evenly according to address space. 109

51 Heatmap of all bitflips excluding weak cells and stuck-at cells over groups of rows for 16 adjacent 32 bit word accesses. 110

52 UBER for rows with one weak cell based on frequency of failures in the neutron radiation beam assuming different weak cell thresholds. Error bars show range based on cell weakness. 110

53 UBER for rows with one weak cell based on frequency of failures after the neutron radiation beam assuming different weak cell thresholds. Error bars show range of cell weakness. 111

54 Encoding W = 0x638 using PFE. 113

55 Memory controller implementation of fault aware PFE. 115

56 Comparison of "moderate-overhead" (∼25%) fault-oblivious approaches of FFE with ECP_{FO}-12 and PFE_{FO}+ECC-1_{32}. 121
57 Comparison of PFE to other "low-overhead" (6.25%) fault-aware approaches. 126
58 Performance impact of run-time determination of weak cells at the memory controller (MemCtrl) or within the memory DIMM (MemDIMM) compared to a fault-oblivious baseline that does not query the weak cell map. 126
59 UBER for different fault mitigation schemes as weak cell incidence rate varies from 0.01% to 1%. 127
60 PFE for different block sizes, n. 127
61 Counter advance example. Green indicates a SA-R fault and orange a SA-W fault. Purple blocks are error free and red blocks contain an error. 129
62 Block level counter advance architecture. 134
63 Cell fault rate for different coefficients of variation. 135
64 UBER for various error rates. Counter advance explores one epoch (w=8), except where noted. Word CA+ECP is reported for CM with an error bar indicating PM. 135
65 Counters advanced per write at various error rates. Counter advance explores one epoch (w=8) except where noted. Word CA+ECP is reported for CM with an error bar indicating PM. Row Encryption is always unity. 136
66 Auxiliary bit example for a 512-bit PCM row for a) traditional ECP, b) Yoda, and c) FaME. SA-W and SA-R refer to stuck-at-wrong and stuck-at right cells. FaME and Yoda overheads do not include the fault map. 138
67 FaME update cache. 140
68 (1) Data/FLOWER access, (2) PETAL comparison, and (3) Error correction on petal detection. 141
69 Lifetime to 1MB memory for 12.5% area budget (iso-area). FLOWER overheads shown in "[]." 144

Preface

I dedicate this dissertation to my parents and family, whose constant and unending love and support made me who I am today. I also present this thesis in memory of my Aunt Betsy, who helped inspire in me a love of reading and the drive for the pursuit of knowledge which I hold very dear to me.

I would like to thank the teachers whose supreme dedication to their students and the pursuit of knowledge helped inspire in me a love of science and technology. In particular, I would like to thank Mrs. Nancy Snyder, formerly of St. Alphonsus grade school, and Mr. Mark C. Krotec at Pittsburgh Central Catholic High School.

I would also like to thank Dr. Alex Jones, for advising me in both academia and life through the process of graduate school. Additionally, I thank Dr. Rami Melhem for his counsel and advice on my research throughout my graduate study. Further, I thank Dr. Yang, Dr. Hu, and Dr. Xiong for serving on my committee and providing input and encouragement on my thesis work.

Finally, I want to thank all of my friends and lab mates for their constant support. In particular, I especially want to thank Ma Luo at Vanderbilt University for his friendship and comradery as we went through the graduate school process together, and Brendan Rodgers, Mike Trentadue, Ben Tyke, Matthew Kizior, Dr. Jiangwei Zhang, Dr. Seyed Mohammad Seyedzadeh, and Sebastien Ollivier.

1.0 Introduction

Technology scaling has introduced many challenges in reliability and sustainability. To solve these problems, I have developed holistic sustainability tools for deeply scaled technologies, compact bit-level fault maps, and efficient encoding solutions which either require no overhead or utilize the bit-level fault maps for unprecedented levels of error prevention.

1.1 Contributions

1.1.1 PART I: Holistic Sustainability Solutions

As process technologies continue to shrink, manufacturing energy has become a significant contributor to system energy consumption [1, 2]. However, most existing energy frameworks ignore this contribution of manufacturing, instead focusing on the use-phase costs. My solutions GreenChip and GreenASIC provide architectural and circuit-level design tools, respectively, to include this manufacturing impact in holistic energy analysis. LARS extends these analyses to also include replacement costs from memories and devices which wear-out over time, such as Phase Change Memories (PCM).

1.1.1.1 GreenChip *GreenChip* [4, 5] is the first predictive manufacturing and use phase environmental impact estimation flow based on targeted technology node and computer architecture design choices such as number of processor cores, cache and main memory sizes and architectures, and solid state disks. GreenChip can provide detailed analysis of these choices with an end goal of supplying consumers with more holistic environmental data akin to fuel efficiency reports for cars. We demonstrate this idea by using GreenChip to compare the impact of different computer architecture choices. We classify the workloads based on memory access requirements as one example of how the data can be aggregated. Our comparisons are made with indifference point [6] and break even analyses.

Indifference point analysis is a common economic metric to determine the point at which there is no difference in cost between two alternatives. For environmental analysis, we define the indifference point as the time when the energy to manufacture and operate two competing system architectures is equivalent. The indifference point can be compared to typical or projected product lifetimes to determine whether a change in manufacturing cost, either across technology generations or due to changes in system architecture, is justified. The break even time indicates the point when a new system will reach the same energy consumption of the system it will replace. This comparison assumes the manufacturing cost has already been invested for the original system. Thus, it identifies the *upgrade* time, when the energy for the new system will be less than leaving the original system in service.

1.1.1.2 GreenASIC GreenASIC [7, 8] is a circuit and CAD-level tool flow for holistic sustainability analysis. For GreenASIC, we developed a parameterized model for determining environmental impacts of process-level changes for technologies between 130nm and 32nm. As an example of using the parameterized model we demonstrate the environmental cost savings for changing the number of metal layers in the stack at these technology nodes. Then, using a variety of benchmark circuits, we demonstrated the change in area and use-phase energy for different metal stacks and provide a holistic environmental evaluation of their change in sustainability.

Our results indicate changing metal layers from eight to six can, with area tightly constrained, result in manufacturing energy savings of 9.5%, 13.8%, and 13.0%, on average, for 130nm, 90nm, and 65nm nodes. Moreover, while eight layer metal can have a lower operational energy than six layer metal, we show that this savings can often take years, or in some cases may never overcome the additional manufacturing impacts from the more complicated process.

1.1.1.3 LARS LARS is a new cradle-to-grave energy evaluation framework that considers system lifetime. As carbon emissions increase, it is increasingly important to have a holistic understanding of how computing systems consume energy. However, tools such as GreenChip do not consider lifetime due to reliability in their analysis. We propose the Life-

time Amortized Replacement for Servers (LARS) indifference analysis approach to conduct sustainability evaluation of systems. LARS considers the impact of reliability techniques via replacement cycles for servers, along with embodied and operational energy. Using LARS in concert with GreenChip, we conduct a holistic life-cycle-based energy analysis of the manufacture and operation of PCM with AES-XTS across its lifetime. Using LARS, we can provide a holistic analysis of reliability solutions such as MACE and WINDU, demonstrating how the 15% mean reduction in dynamic energy and lifetime improvements inform system design and memory selection.

1.1.2 PART II: Bit-level Fault Maps

The scaling of process technologies has also significantly impacted the frequency of faulty cells due to process variation. For many types of faults, process variation results in the existence of *weak cells*, cells which have an abnormally large probability to exhibit the failure mode in question. This ranges from vulnerability to crosstalk, to abnormally fast wear-out, to weakness to neutron radiation. In order to enable highly-effective bit-level correction schemes, I have developed compact bit-level fault maps which provide a dense representation of these weak cells in the memory system.

1.1.2.1 SFaultMap SFaultMap [9] is a sustainable bit-level fault map (SFaultMap) for use in memories with potential faults caused by process variation. A goal of SFaultMap is to reduce holistic energy, including embodied energy. As embodied energy is rising along with technology scaling, SFaultMap's design makes apparently *unconventional tradeoffs* to address holistic energy. We explore tradeoffs in adjusting SFaultMap's search strategy and size in order to make the most sustainable choice.

In DRAM, SFaultMap can be used to identify cells susceptible to crosstalk (i.e., "weak" cells), and cells with low retention time. It may also be used to identify permanently faulty cells in DRAM, such as manufacturing defects from improper cell orientations or cells with unconnected bit or wordlines. Moreover, it can be used to identify cells that have failed during active use such as endurance stuck-at faults from PCM. We compare SFaultMap to

error correction pointers (ECP) that essentially store a version of the fault map distributed across the rows of the memory [10], as well as ArchShield [11]. Further, we demonstrate through indifference analysis the points at which different error mitigation strategies are the most sustainable for holistic energy consumption. Finally, SFaultMap is a low-area bit-level fault map for a low embodied energy solution to reliability in emerging technology nodes, which demonstrates significant benefits in holistic sustainability when compared to existing solutions such as ECP and Archshield.

1.1.2.2 FLOWER FLOWER (Fine-grained, Low Overhead Weak- and Error-prone-cell Registry) is a space and energy efficient bit-level fault map to enable fault tolerance in the presence of high fault incidence rates (*i.e.*, $\geq 10^{-4}$). FLOWER leverages Bloom filters for its implementation to ensure that it is both space efficient and has a low access latency. While FLOWER can report a small number of "phantom faults" or false positives due to overlapping of the hashing functions, FLOWER is guaranteed to report all "true faults." We demonstrate that the number of false positives are easily minimized through effective multiple dimension hashing while maintaining the same storage overhead. Further, leveraging processing-in-memory techniques, we demonstrate how FLOWER can efficiently construct fault vectors in DRAM and emerging non-volatile memories such as PCM from the multi-dimensional fault map.

Additionally, FLOWER supports dynamic updates and scales efficiently at higher fault rates expected in current and future deeply scaled technology. FLOWER uses *minimum-cumulative intersection*, a tuned hashing technique we developed, which has lower collisions for moderately dense data than traditional (*e.g.,* disk-level hashing) techniques along with better performance. FLOWER can use processing-in-memory to construct fault vectors from multiple dimensions to minimize memory traffic, and uses novel *PETAL bits* for error correction of in-memory operations.

1.1.2.3 HOTH Before beginning HOTH, I examined the hypothesis that single event effects from neutron radiation are heavily biased to a small number of cells which, while not permanently failed, experience a higher incidence of soft faults. In particular, evidence from

experiments conducted at Los Alamos Neutron Science Center (LANSCE) demonstrates that a small number of these weak cells are responsible for a high proportion of SEEs in DRAM. Moreover, these SEEs experienced can be classified into three groupings, multi-bit latchup, single cell upset to a weak cell (predictable), and single cell upset to a random cell (unpredictable).

Based on this classification, we propose a fault correction approach based on HOTH, or Hashed Omniposition-correction-pointers with TMR for Harsh-environments. HOTH combines a fault map that stores pointers to fault locations with spare bits to mitigate predictable faults. When combined with Chipkill ECC, we demonstrate that fault tolerance can be dramatically improved to be competitive with dedicated radiation tolerant solutions while leveraging the density, performance, and energy of commercial, off the shelf (COTS) DRAM.

Our experiments show that COTS DRAM is highly prone to SEEs and these SEEs are highly predictable. In contrast, FPGA BlockRAMs (SRAMs) experience few unpredictable SEE faults, and Intel Optane experience no faults. Furthermore, we demonstrate that HOTH provides **seven or more orders of magnitude** additional fault tolerance in terms of uncorrectable bit error rate (UBER) to SEEs compared to traditional fault tolerance alone.

1.1.3 PART III: Encoding Solutions for Mitigating Memory Faults

With low-area bit-level fault maps made available by work in my previous section, extremely compact and thorough correction can be created. In these chapters of my dissertation, I discuss my low-overhead encoding methodologies for avoiding faults for different memory vulnerabilities. This includes PFE, primarily for bitline crosstalk, counter advance for worn-out cells in encrypted PCM, and FaME, for a variety of permanent failures or weak cells.

1.1.3.1 PFE: Periodic Flip Encoding Periodic Flip Encoding (PFE) [12] avoids bad patterns that lead to crosstalk. This technique reduces bitline crosstalk while not harming the protection against wordline crosstalk. PFE first partitions the data into groups and then

flips the same bit position of each group. For groups of three bits, this approach provides four different code words for a block, and only two auxiliary bits are needed to specify the code word used. In the absence of information about weak cells, PFE is fault-oblivious and selects a code word that decreases the number of bad patterns. However, when the locations of the weak cells are known, PFE may also be employed in a fault-aware fashion by selecting a code word that may have a bad pattern, *as long as the center of the bad pattern does not overlap with a weak cell.*

PFE is the first work that uses coding-based techniques to mitigate bitline crosstalk in DRAM cells by reducing the number of bit patterns vulnerable to crosstalk. PFE is available with fault-oblivious and fault-aware (which requires a bit-level fault map) periodic flip encodings that increase the encoding space in return for crosstalk fault mitigation. We provide a characterization of the fault mitigation of PFE and conduct an extensive study to illustrate the tradeoffs between reliability, cost, performance, and power.

1.1.3.2 Counter Advance *Counter advance* [13] is a technique which leverages the existing encryption hardware to improve reliability in PCM with dedicated counter-based encryption. Counter advance utilizes several observations: First, for stuck-at PCM cells, depending on the value to be written and the state of each faulty cell, the data can either be stuck-at the right value (stuck-at right, SA-R) or the wrong value (stuck-at wrong, SA-W). Second, a new encryption of the same data item can be generated by running the encryption with the next counter value. Third, for each new encrypted candidate for storage, there is a 50% chance for each faulty cell that the data will be SA-R. Thus, by exploring multiple counter values, a value that eliminates (or minimizes) SA-W values may be obtained. In this case, even for a row with some faulty cells due to endurance, the system may continue to operate, extending the useful lifetime of the memory. Assuming a 27- to 32-bit counter, 10^8 to 10^9 writes per row, which meets or exceeds the projected PCM cell endurance, can be achieved for a single key. If multiple encryptions are used only to tolerate faults, then the average counter advances per write will be $<< 2$.

Counter advance improves the reliability and lifetime of PCM storage subject to endurance faults. As part of this contribution, we demonstrate an architecture to apply counter

advance in the context of the leading method to reduce energy for counter encrypted PCM, and show the combined capability of counter-advance with ECC and ECP protection to improve reliability and minimize storage overheads.

1.1.3.3 FaME FaME (**Fa**ult **M**ap **E**nabled error correction) is a new fault tolerance technique enabled by our bit-level fault maps. FaME distributes spare storage cells as small numbers of auxiliary bits for each memory row. For each memory access, FLOWER reports the location of the faulty cells in that row and the FaME spare storage cells are used in place of those bit locations. This avoids the need to maintain pointers to the fault locations within the row, as is the case for ECP. FaME also naturally protects auxiliary bit failures by including the auxiliary bits in the fault map and sparing faulty auxiliary bits. In contrast, ECP requires two pointers to correct one failure in the auxiliary bits. For the space of an single ECP pointer and replacement cell, FaME can correct as many as 10 faults. FaME can dramatically reduce storage overheads while maintaining high fault tolerance for high fault rates.

2.0 Background

This chapter is divided into three sections. The first covers the first major topic of my dissertation, memory reliability. Similarly, the second section covers holistic sustainability, the second major topic of the dissertation. The third section covers background necessary for one or several of my contributions, but not all or most of my contributions.

2.1 Memory Reliability

2.1.1 DRAM

Certain DRAM cells, referred to as *victim cells*, experience wordline crosstalk faults when their stored charge is drained prematurely due to the repeated charging of adjacent wordlines without the victim cells being refreshed [14]. Studies have demonstrated that as few as 30,000 accesses to adjacent rows without refreshing can manifest in an incorrectly stored data bit [14]. "Row hammering" exploits this form of crosstalk through intentionally repeated opening and closing of a row with the goal to create changes in neighboring rows to gain access to protected memory locations or to cause the system to crash [15]. Proposed solutions include ECC, per-row and groups of counters [16], probabilistic refreshing [14], reducing the refresh interval, and runtime testing which identifies and avoids weak (prone to crosstalk) DRAM cells with use of a word-level fault map [11].

Similar to wordline crosstalk, bitline crosstalk occurs when certain susceptible cells are especially vulnerable to this form of crosstalk [17]. Bitline crosstalk is manifested by fluctuations in the read bit values originating from the voltage levels of bitlines connected to the same sense amplifiers [18]. Within an individual row, bitline crosstalk can be observed in cases where bits surrounding a weak cell, and the weak cell itself, share the same charge [17]. This can result in the weak cell flipping, resulting in an incorrect data bit [19]. Until recently, circuit-level techniques [20, 21] have all but eliminated bitline crosstalk. However, these ap-

proaches exacerbate wordline crosstalk. The combination of increased wordline crosstalk and restrictions in the design kit, such as 1-D routing, have resulted in DRAM manufacturers returning to the open bitline structures where bitline crosstalk is present [22, 23]. Bitline crosstalk is sufficiently present in scaled DRAM that it has been used to determine the locations of physically neighboring cells in DRAM products [17].

Weak cells can also manifest with reduced retention time that can lead to faults without crosstalk, if not suitably refreshed. This hampers power savings efforts by reducing refresh intervals [24, 25] or by requiring additional error correction [26, 27, 28]. While significant work has been completed with the intention of solving all three of these problems, many of the correction schemes developed for one type of fault either directly exacerbate another problem (bitline twisting, increased refresh time) or come at the cost of significant area overhead for error correction (e.g., multi-bit ECC). Further, it has been shown that cells with reduced retention capability are not related to the wordline crosstalk victim cells despite the functional similarities[14], increasing the difficulty of managing all three types of faults.

2.1.2 PCM

PCM has been proposed as a potential replacement of DRAM for main memory to gain storage density and save static power. It has recently gained commercial traction in 3D Xpoint (Optane) memory [29]. Other than the challenge of higher write energy, the primary concern with PCM utilization is the limited write endurance (circa 10^8 writes) before cells become permanently "stuck-at" a particular value. While these cells can be read, they can no longer transition from their current amorphous or crystalline phase [30].

Additionally, early failures of weak cells due to process variation will severely limit the useful lifetime of PCM. By extending the fault-tolerance capability from 10^{-4} to 10^{-2} using techniques I have developed throughout this dissertation, the lifetime can be increased by more than 100% for a coefficient of variation of 0.2. This improvement grows to 3.8× and 16.4× as CoV grows to 0.25 and 0.3, respectively [13].

PCM is also vulnerable to write disturbance due to crosstalk. Due to the high disparity of energy during writing from the fundamental asymmetry of the phase-change process, writing

certain patterns such as “00” in combination with a weak cell can be incorrectly stored as “10” or “01” [31]. In all of these cases, knowledge of the location of weak cells is critical to appropriate fault tolerance.

2.1.3 Existing Fault Maps

ECP [10] is essentially a fault-map combined with spare bits distributed through row-level auxiliary bits. For stuck-at faults at low error rates, ECP is more efficient than ECC but sacrifices protection of transient faults. ECP has also been adapted for DRAM [32, 28]. Each N-bit memory row contains pointers of $log(N)$ bits along with a bit to store the correct value at that pointer location. While ECP is effectively a bit-level fault map, it is expensive in terms of area, and scales linearly with the number of expected faults per row. ArchShield [11] is a word-level fault map combined with sparing proposed for DRAM. In its design, each row contains two bits to set whether the row has zero, one, or multiple faulty words in a row. A combination of error correction (when possible) and replication is used to maintain fault tolerance. Rows with multiple faults must access the redundant copy of the faulty words on every access, while rows with single faults only need to access the redundant copy if the ECC protection in the original stored location fails. ArchShield operates on incident fault rates between 10^{-6} and 10^{-4}, and within this range is space efficient and typically results in a performance degradation of less than 2%. As error rates scale beyond 10^{-4}, the required overhead scales super-linearly, and as a result ArchShield is not viable in that operating range.

2.2 Holistic Sustainability

The considerable attention and focus to use phase energy consumption in modern computing systems is a natural extension of research that aims to address thermal concerns caused by increases in power density associated with semiconductor technology scaling. These use phase energy reduction measures can help maximize battery life for mobile elec-

tronics and minimize operational energy costs of data centers. To achieve holistic sustainability requires considering the entire computing life-cycle, for which a science called Life Cycle Assessment (LCA) is commonly used. In this section we briefly introduce LCA and discuss previous work in LCAs on computing equipment and in particular their ICs.

LCA allows an engineer to quantitatively evaluate how processes and products use materials, water, and energy resources and the resulting environmental impacts throughout their lifetimes. Established guidelines for performing detailed LCAs are documented by the Environmental Protection Agency, Society for Environmental Toxicologists and Chemists, the International Organization of Standardization (ISO), and the American National Standards Institute [33, 34]. As defined by the ISO 14040 series, LCA is an iterative four-stage process including: 1) Scoping —- defines the extent of analysis and the system boundaries; 2) Inventory Analysis —- documents material and energy flows that occur within the system boundaries (life cycle inventory or LCI); 3) Impact Assessment —- characterizes and assesses the environmental impacts using data obtained from the LCI (life cycle impact assessment or LCIA); and 4) Interpretation and Improvement —- identifies opportunities to reduce the environmental burden throughout the product's life cycle.

There are three main LCA strategies: Process LCA, Economic Input/Output (EIO) LCA, and Hybrid LCA. Process LCA evaluates all steps involved in each stage of the LCA and directly evaluates their impacts as well as impacts from upstream elements such as the fundamental components used in the process. EIO LCA works from the principle that environmental impacts typically correlate with the financial cost of the process, and therefore uses the cost of a product to estimate its environmental impacts. Hybrid LCA uses a mixture of both Process and EIO LCA. Several life-cycle tools and databases have been developed such as the NREL LCI database [35] and the LCIA Tool for the Reduction and Analysis of Chemical and other environmental impacts (TRACI) for distinguishing carcinogenic impacts [36]. In the next section we present some relevant research on LCA as it applies to computing systems.

2.2.1 LCA of Computing Systems

In Figure 1, we present the carbon emissions from the life-cycles of various Apple products [1], demonstrating that the dominant phases are production (manufacturing) and use. Contrary to expectation, use phase impacts do not dominate; for tablets [Figure 1(a)], manufacturing can reach as much as 90% of the overall carbon footprint. Additionally, while the use phase impact decreases across product generations, the manufacturing impact has continued to rise. For instance, comparing the iPhone 5s and 6s, the use phase impacts remain constant, but the manufacturing impact increases by more than 25%. A similar situation can be observed with the iPad Mini and Mini 4, which have nearly identical carbon impacts, but the use phase savings are entirely offset by the manufacturing increase. Looking at computing systems from laptops to workstations [Figure 1(b)], we see a similar trend where manufacturing impacts are at least half, and in many cases far more than half, of the total carbon footprint.

Recent life-cycle studies [37, 38, 39, 40] have pinpointed ICs [1] and displays as having the dominant manufacturing environmental impacts of computing systems. As the use phase energy and resulting environmental impacts continue to decrease, there is mounting evidence that the environmental trends for IC manufacturing are becoming increasingly environmentally unfriendly. Considering the two desktop machines from Figure 1(b) without an integrated display, the Mac Pro and Mac Mini gain 67% and 90% of their impacts from manufacturing of non-display components, respectively. In these cases, the IC components become the dominating contributors due to solid state drives (SSDs) and large amounts of memory in addition to the processor and supporting circuitry. We explore IC manufacturing trends further in the next section.

2.2.2 Impacts from IC Fabrication

A hybrid LCA of IC manufacturing over a 15 year period ranging from 350nm to 45nm [2] reveals problematic environmental trends. Environmental impacts from fabrication per area

[1] ICs are grouped with printed circuit board manufacturing [39] but shown to be negligible compared to ICs [38].

(Figure 2) reached a minimum point at 130nm. Unfortunately, as technology descended below 90nm, the "environmental impacts per die area" increased with feature size. While these trends could be mitigated if IC die area per system decreased with the descent in feature size and resulting increase in device/transistor density, the opposite trend has been observed. Newer systems tend to have more IC area (for example, area increased from 750 to 1200 mm^2 between 2001 and 2010) due to trends to include more processor cores, embedded memory, accelerators such as graphics processing units, and solid state storage [39]. Moreover, trends such as "dark silicon," where many infrequently used hardware blocks and accelerators are included for use in improving energy efficiency and performance of relatively infrequent niche functions, also work against manufacturing sustainability.

Using the parameterized fabrication estimation method for ICs from Murphy et al. [3] combined with the Apple LCA data, it was possible to extend the chart in Figure 2 from 45nm to 28nm. We examined the manufacturing cost of several apple tablet products over different generations implemented with processors fabricated at 45nm and 28nm. Using the reported breakdown of IC contributions [40] and the overall manufacturing effort at each node, the trend indicates a dramatic increase in manufacturing effort, supported by a transition in CMOS manufacturing from planar bulk CMOS to silicon-on-insulator at 36nm [41] (between 45nm and 28nm). This resulted in a significant savings in use phase energy [42] but seems to dramatically increase manufacturing effort. There is additional indication that lithography effort, currently the dominant component of manufacturing costs [43], may have seen a sharp increase.

Standard immersion lithography (193nm ArF source with immersion) provides a pitch size limited to approximately 60nm. 2X CMOS nodes (28nm and lower[2]) require some form of double patterning [43], which, depending on technique, can increase the number of lithography steps and resulting environmental impacts dramatically. The resulting data indicates a 5X increase in energy and GWP [1], which is consistent with this trend. Moreover, this trend appears to be poised to accelerate aggressively as nodes at 10nm and lower appear to require multiple patterning lithography. This is due to extreme ultraviolet lithography's

[2]22nm is confirmed to require double patterning while 32nm only requires single patterning [43]. 28nm is assumed to require double patterning based on the increase in IC manufacturing impacts reported by Apple [1] which is a feasible changeover point for a 193+i lithography pitch limit of 60nm.

earliest availability being predicted for the 7nm node [43] and is consistent with economic cost improvements of Dennard scaling breaking down at these nodes [44]. With current technology utilizing power-optimized hardware, production often exceeds 75% and reaches 90% of the total life-cycle cost for a 4-year service time (see Figure 1) [1]. This fact, along with the aforementioned trends, points to a need to examine the holistic environmental cost.

2.2.3 Holistic Sustainability Related Work

There have been several LCA studies of ICs [40, 45, 38, 37, 2, 46] and full computing systems [39, 47, 48]. The studies examining IC manufacturing typically evaluate the fabrication process generally for different technology nodes [2, 3]. Other studies examine study particular computing products for their environmental impacts [1]. Other studies consider case studies of particular products to draw general conclusions about different classes of computing products [45, 49]. Finally, many existing LCA tools provide the ability to provide estimates for semiconductor environmental impacts such as GaBi and SemaPro. Unfortunately, these LCA tools are designed to examine the materials used in the final product to estimate impacts, which can often be misleading for semiconductors in which materials may be applied and entirely removed to create smaller feature sizes, a common practice in multiple patterning lithography.

The LCA results from many studies have identified use phase and manufacturing phase impacts as the dominant contributors to energy and carbon emissions for computing systems [2, 38]. This is demonstrated in Figure 1, where phases like transportation and recycling are very low compared to the manufacturing and use phases of the system life cycles [1]. Our tool, GreenChip, focuses on these two dominant phases to provide a representative view of the system.

2.3 Miscellaneous

2.3.1 Memory Encryption

Counter-mode encryption [50] to secure main memory, depicted in Figure 3, was originally proposed for DRAM [51] by adding a cipher into the memory controller and adding counter storage for each memory row. Using a private key, a unique counter value, and the row address, the cipher generates a one-time pad (OTP). The OTP is XORed with the data to create an encrypted ciphertext. Decryption functions in the reverse, reading the data ciphertext and the counter in plaintext from the memory row to recreate the OTP and reverse the encryption process.

Unfortunately, encryption has the side effect of disrupting data locality, as for each plaintext value written, a unique OTP is generated containing a random set of 0's and 1's. Because the OTP changes for each write and the OTP is XOR'd with the plaintext, small or large changes in the plaintext results in similarly random ciphertext. This unpredictable output for each OTP is what gives the encryption its strength. When applied to a memory technology like PCM, standard energy saving techniques such as encoding and differential write are defeated and the increase in bit changes can lead to early cell wear-out.

SECRET or Smartly EnCRypted Energy EfficienT non-volatile memories [52] addresses these challenges by allocating a dedicated "epoch sub-counter" for blocks within the row. Thus, row writes must only encrypt dirty blocks. The sub-counter maintains independent count values for each block within an epoch. When a sub-counter saturates, the epoch ends, the main counter is advanced, all sub-counters are reset, and the entire row is encrypted and written with the new count value. SECRET addresses reliability by allocating ECP pointers [10] per row to tolerate faults.

2.3.2 Bloom Filters

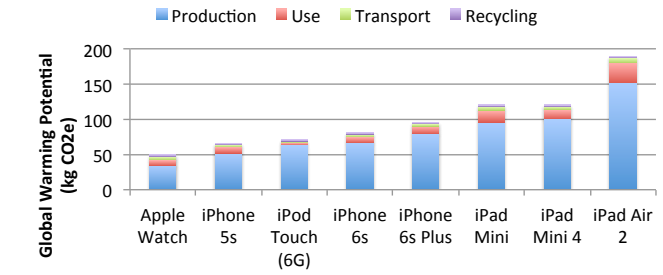
Bloom filters [53] improve space and time efficiency of checking set membership. Their application includes commercial search engines [54], DNA sequencing [55], and fast packet classification [56]. Bloom filters typically appear in the form of a bit array with a hash

function, where an item hashed to the corresponding bit in the array was set to '1.' Multi-dimensional Bloom filters use multiple hashes where each hash location in the array is set to '1.' On checking set membership, the bit corresponding to each hash function is accessed, and the item is only considered a member if each of those hashed values return '1.' Because Bloom filters reduce the amount of stored information, there is a chance to categorize a non-set member as a member of the desired set. While this situation, referred to as a *false positive*, may occur, the opposite case, where a member of the set is incorrectly classified as not being in the set (or a *false negative*), cannot occur. Another side effect is that while Bloom filter construction (adding set members) is simple, items cannot be removed from the set without outside knowledge.

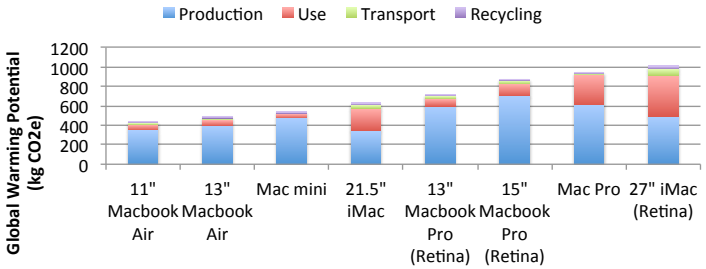
Many Bloom filter modifications have been developed to suit particular applications. A selection of these many variants include the blocked Bloom filter [57] (optimized to fit into cache lines) , the counting Bloom filter [58] (allows deletion by decrementing count value), the stable bloom filter [59] (optimized for streaming applications), the quotient bloom filter [60] (stores additional meta data for deletion), and the cuckoo bloom filter [61] (improved version of quotient filter). In addition to adjustments to the structure of the Bloom filter, many studies and enhancements have been performed on the hashing algorithms used in Bloom filters. The goal of a Bloom filter hash algorithm is to produce independent random variables when hashing the value in a multidimensional Bloom filter. This minimizes the risk to hash into the same bin and ultimately minimizes the occurrence of false positives. While traditional cryptographic hash functions such as SHA-2 work well for this purpose, faster non-cryptographic hash functions which approximate k-independence are often sufficient, such as Pearson's hash [62] and MurmurHash [63] , the latter of which is used in Hadoop [64].

Similar to our utilization of Bloom filters for memory error correction, CiDRA [32] utilized Bloom filters to create a cache to detect if an address contains any faulty cells. In this way, the additional error correction procedures or redundant information retrieval only has to occur for the select few addresses which have faults or are false positives. While both CiDRA and this work utilize Bloom filters, CiDRA uses them for caching memory elements (*e.g.,* cache lines or memory rows) which have errors. CiDRA runs into similar problems

to solutions like ArchShield for high fault rates as CiDRA will report nearly all memory elements are faulty. This is exacerbated by the false positives reported by the Bloom filter.



(a) Tablets



(b) Computers

Figure 1: Impact of manufacturing/production from ICs in "use phase energy" optimized systems.

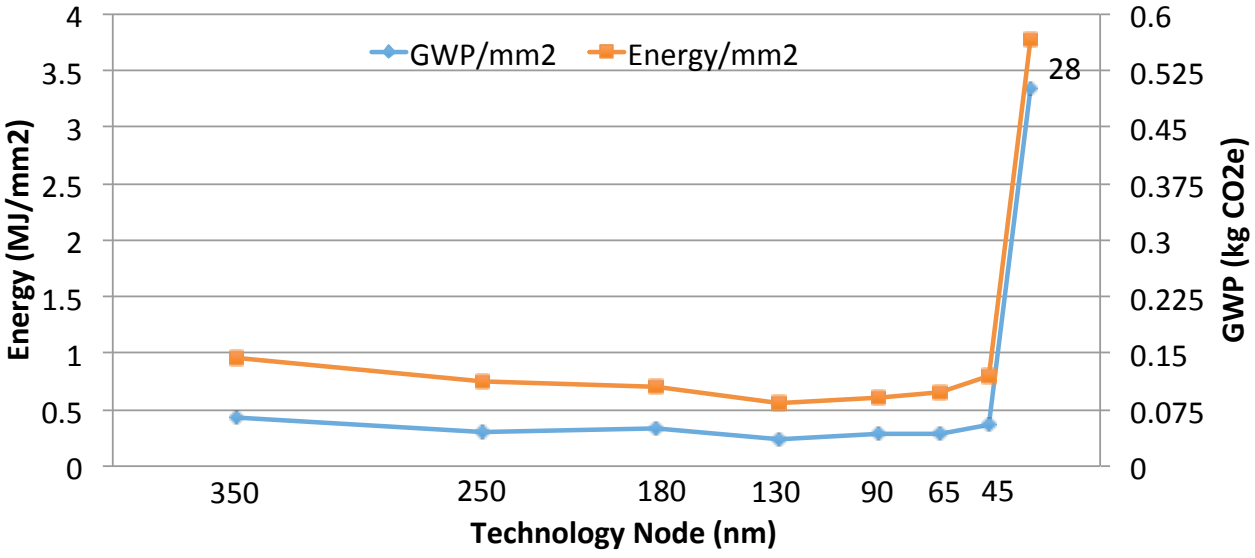


Figure 2: IC fabrication energy and global warming potential (GWP) trends.

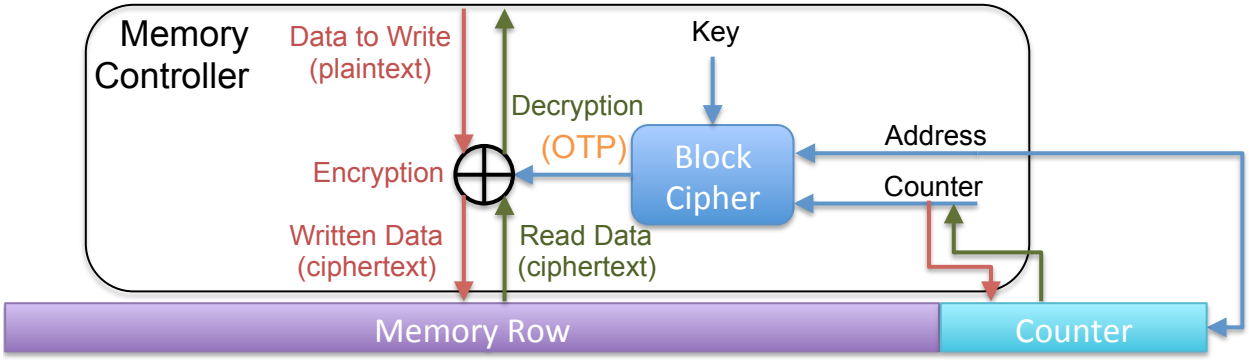


Figure 3: Counter mode encryption in the memory controller.

3.0 GreenChip

3.1 The GreenChip Sustainable Computing Prediction and Evaluation Tool

To evaluate and compare the manufacturing and use phases of new computer architectures and systems we have created the GreenChip flow shown in Figure 4. GreenChip can be used for the processor in isolation [Figure 4(a)], combined with the main memory system [Figure 4(b)], and even extended to consider secondary storage in the form of SSDs [Figure 4(c)]. GreenChip first simulates the behavior of a mixture of workloads on a proposed architecture to generate performance statistics. The simulator output, architecture specification, and technology node are then fed into a use phase power estimation flow. To accomplish this, existing tools are available and may be leveraged. GreenChip is currently built from the Sniper full system simulator [65] and the McPAT use phase power estimation flow [42]. A more detailed, cycle-level simulator such as gem5 [66] can easily be used with GreenChip. However, this level of detail may not be required for this type of analysis. Furthermore, the simulation time required for cycle level processor simulation makes it difficult to use large workloads that stress the main memory and disk in a reasonable amount of simulation time.

GreenChip extends this flow with a manufacturing environmental cost estimator[1] that uses a combination of the technology node impacts per area and the predicted area of the IC or ICs. For example, in the processor evaluation [Figure 4(a)], area estimates for the processor for a particular technology node can be obtained from McPAT [42] and CACTI [67] and combined with the manufacturing cost per area of CMOS logic (Figure 2) to determine overall manufacturing cost.

DRAM manufacturing cost is computed in a similar way to the processor cost using the DRAM technology trend [2]. DRAM tends to mirror the CMOS trends with an increasing cost per area starting after 70nm. Total cost is determined based on the die area, itself a function of the capacity per DRAM IC, combined with the cost per area. DRAM tends to

[1]When discussing manufacturing cost, effort or impact, we are referring to environmental impact/cost, not economic cost, unless specified.

trail CMOS by one technology node, so generally systems comparisons would consider the year a system was built rather than identical feature sizes (e.g., a 45nm processor would have a 57nm main memory system). Manufacturing cost can also be determined for Flash ICs used in SSDs, although the data is typically reported per capacity rather than die area [2], and is incorporated into GreenChip [Fig 4(c)] in the same fashion.

To determine the overall energy cost of a system during its lifetime, a usage scenario must be considered consisting of the time the system is awake versus asleep (*sleep ratio*), when awake, how much it is active versus idle (*activity ratio*), and the time period it will be in service. We determine this number from the average power of the usage scenario shown in Eq. 3.1

$$P = (1 - r_S)(r_A(P_D + P_S) + (1 - r_A)P_S) + P_L \tag{3.1}$$

where P_D, P_S, and P_L are dynamic, static, and sleep power, respectively, during workload execution, r_S is sleep ratio, and r_A is the activity ratio of time spent executing the specified workload. Thus, the time t_W for the processor to be doing useful work is $t_W = t(1 - r_S)r_A$ where t is the service time. The overall energy cost is computed by $E = P \cdot t + M$ where M is the manufacturing cost described previously.

In addition to reporting raw environmental impact outputs for a particular system design, GreenChip also provides direct comparisons of two design choices. Using indifference analysis, the choice of system can be evaluated based on the expected service time. The indifference formula t_I of comparing two architectures, $System_0$ vs. $System_1$, is shown in Eq. 6.1. t_I is the time at which the increase in manufacturing cost will be outweighed by the savings in use phase cost. If the proposed service time $t < t_I$ the architecture with the lower manufacturing cost minimizes environmental impact and for a proposed service time $t > t_I$ the architecture with the lower use phase cost minimizes impact. If one system is lower in both costs, t_I is either < 0 or $= \infty$, making it invalid and pointing toward the selection of the lower cost system regardless of service time.

$$t_I = \frac{M_1 - M_0}{P_0 - P_1} \qquad t_B = \frac{M_1}{P_0 - P_1} \tag{3.2}$$

The break even time t_B is also defined in Eq. 6.1. t_B represents, given an existing system ($system_0$), what service time for a new system ($system_1$) would be required to offset the upfront manufacturing cost to save overall energy. This is relevant to answer the "upgrade" question. For both of these comparisons, GreenChip automatically adjusts the selected usage scenario to account for the change in performance due to a different architecture configuration. Using the workload, the IPC of both proposed systems is determined. With $system_0$ as a baseline, the activity ratio of $system_1$ is adjusted by replacing r_A with $r'_A = r_A(\frac{IPC_0}{IPC_1})$ in Eq. 3.1[2].

Additionally, GreenChip is able to report various gas and byproduct emissions including carbon emissions and carcinogens from manufacturing and use phase energies for the U.S., China, and for a worldwide average using data from the literature [2] and electricity generation mix data [68]. For the remainder of this paper we focus on case study comparisons of energy from manufacturing and use phases of common architecture configurations at different technology nodes and for different workloads to highlight interesting trends and the importance of considering manufacturing impacts in developing next generation sustainable computers.

3.2 Case Study I: Environmental Impacts of Recent Processor Trends

As processors have descended below the 90nm node, clock frequencies have become relatively fixed to manage thermal concerns. Performance improvements have instead been achieved by using the additional density per die to increase the number of processor cores

[2]In our comparisons, the baseline system was typically set to the slower of the two systems. If $IPC_0 > IPC_1$, the activity ratio of $system_0$ would instead be adjusted by $r'_A = r_A(\frac{IPC_1}{IPC_0})$, with the activity ratio of $system_1$ unaltered.

and on-chip cache sizes. In this section we use GreenChip to demonstrate how these trends impact sustainability.

3.2.1 Experimental Setup

We consider pseudo ISO-area configurations across several different technology generations that mirror the processor products available in the corresponding years. In particular, a 90nm processor was configured with one core and 1MB of LLC, a 65nm processor with two cores and 2MB LLC, a 45nm processor with 4 cores and 4MB LLC, and a 28nm processor with eight cores and 8MB LLC. Each system employs a 4-way issue, out of order core model operating at 2.6GHz[3] with a bus-based interconnect to access the LLC and main memory. Cache and main memory latency and power consumption were taken from CACTI [67] and DRAMSim2 [69], respectively. Power consumption of the processor configurations (i.e., P_D, P_S, and P_L from Eq. 3.1) was determined using McPAT [42].

We used GreenChip to analyze the indifference points, IPC, energy, and MPKI of a mix of the Parsec [70] and SPEC-CPU2006 [71] multi-program workloads. The memory impact and specific benchmarks to construct the workloads is shown in Table 1. The Parsec workloads are both multi-threaded and multi-program, while the SPEC workloads are single threaded and multi-program. Unfortunately, due to limitations in the simulation environment, the Parsec multi-threaded workloads could only be run on the four and eight core configurations, limiting their experiments to the 45nm and 28nm processors. The estimations of the individual benchmark memory impacts were taken from the literature [70, 72] and are listed in the order of the benchmarks. Multi-program workloads were selected to represent systems with several concurrent processes.

In our sensitivity analysis, we evaluate different usage scenarios with four activity and sleep ratios (see Section 5.2) shown in Table 2 representing the load experienced by a cloud server (Server) that is typically online but often underloaded, a high-performance machine (HPC) that is typically constantly online and heavily loaded, a desktop machine (Desktop)

[3]Clock speed is assumed invariant across technology nodes as is commonly the case due to power/thermal concerns.

that is used often, but lightly during the working day, and a mobile device (Mobile) that is mostly asleep, but when it wakes up is heavily loaded [2, 73, 74].

3.2.2 Results

The manufacturing costs of the different architecture choices are shown in Table 3. Manufacturing cost is reported for 90-45nm by Boyd [2] and 28nm is determined from Apple Environmental Reports [1] and normalized to 45nm from Boyd. The manufacturing cost per technology node tends to increase for each generation due to the increase in manufacturing cost per area (see Figure 2) and even though the area decreases significantly for the 28nm node, the increase in manufacturing cost per area still results in a dramatic jump in manufacturing energy.

In contrast, the use phase energy, shown in Figure 5, shows how increasing the core count and cache size can dramatically reduce use phase energy by a combination of increasing performance and savings in use phase power. However, the the appropriate environmental design choice requires a combination of both manufacturing and use phase energy trends.

The indifference analysis of a selected benchmark (GLMN) is shown in Figure 6 to illustrate the design space, with the four scenarios, Server, HPC, Desktop, and Mobile, represented by circles in the top left, bottom left, top right, and bottom right regions of the figures, respectively. For both the Desktop and Server scenarios, the higher manufacturing cost of the smaller process node in the 90nm vs 65nm and 65nm vs 45nm comparisons is not recovered through use phase gains. In contrast, the higher performance of the smaller node in the HPC and Mobile scenarios results in the larger manufacturing energy being offset by use phase gains in less than 2 years, suggesting the more environmentally sound approach is to choose the smaller technology node. In the 45nm vs 28nm comparison, HPC and Server scenarios reach the indifference point in less than 2 years, while the Mobile and Desktop scenarios approach 10 years.

The break even comparison from Figure 7 shows similar trends but with sharper gradients through the design space. Across the three node comparisons, the break even time for the Desktop scenario is larger than 7 years, and always larger than 5 years for the Mobile scenario.

While the Server scenario never breaks even for the 90→65 and 65→45 node comparisons, 45→28 breaks even after 3 years. Consistent with indifference analysis, the HPC scenario demonstrates that upgrading is the most sustainable decision, as long as the new device will be in use for at least a year.

To achieve a more global view, the average indifference points and break even times are shown in Figure 8 for the four scenarios. The results follow similar trends as shown in GLMN example, with HPC systems typically pushing toward the new technology node quickly, Desktop, Server, and Mobile typically not pushing toward the new technology node with the exception of purchasing 45nm mobile and 28nm servers over 65nm and 45nm, respectively if within a 3-year usage time.

3.3 Case Study II: Sensitivity Analysis of the Impact of Cache Sizes on Sustainability

One common architecture configuration option is to change the size of the last level cache. In this case study we fix the processor into a four-core system and vary the LLC capacity from 0.5MB to 4MB and examine the impact on sustainability. The manufacturing cost of varying the LLC capacity is shown in Figure 9. For all technology nodes, the increase of capacity is met with a significant increase in manufacturing cost, which attenuates as the feature size is reduced.

Considering the same scenarios and workloads described in Tables 1 and 2, the indifference point analysis always selected the smaller of the two technology nodes regardless of scenario due to the reduction in manufacturing and use phase cost, with the exception of the 45nm to 28nm transition. This is due to the iso-architecture comparison, where the larger technology node areas are much higher, rather than the more realistic pseudo iso-area comparison from Section 3.2. Interestingly, the 45nm to 28nm indifference points (Figure 10) vary widely by cache size, trending to become smaller as the LLC capacity increases within each usage scenario.

Moreover, this trend differs from the break even study in Figure 11. The Desktop scenario never breaks even from 90nm→65nm at any cache size. For the Desktop 65nm→45nm comparisons, mobile 90nm→65nm and 45nm→28nm, and server 90nm→65nm comparisons, the break even times all exceed four years. The Desktop 45nm→28nm and mobile 65nm→45nm comparisons are both around three years. Finally, the entire HPC scenario and Server 65nm→45nm and 45nm→28nm transitions all break even in less than one year.

To better understand these results we examined the performance in instructions per cycle (IPC). In these experiments, the IPC (Figure 12) stays relatively constant across technology nodes but has varying effects for the different multi-program workloads; for example for RFFB the additional LLC capacity does not noticeably improve performance, while for the other workloads the IPC steadily improves as the capacity increases. Also, as expected, the misses per kilo-instructions (MPKI) decreases as the LLC capacity increases. The change from 0.5MB to 1MB has the largest MPKI decrease with larger LLC capacities having limited additional improvements.

The energy fluctuation (Figure 13) for the different cache sizes within a workload and technology node depends on the trade off of additional performance from the larger LLCs against the increase in static power as the cache size increases. For example, CXFD at 28nm experienced sufficient performance benefits from increasing the LLC size to offset the static power increase resulting in a reduction in energy. In contrast, RFFB at 45nm experiences the opposite trend, as the static power increase offsets the nominal performance gains as the cache size increases. On average, there is a reduction in energy from a 0.5MB to 1MB LLC but for larger LLC sizes, the energy remains relatively consistent with the performance trends.

These trends point to 1MB caches providing the best trade off between performance, energy, and manufacturing cost. This is also supported by the breakeven results, with the 1MB LLC typically among the lowest break even times.

3.4 Case Study III: Impact of Main Memory Density on Sustainability

As the number of cores and cache densities in modern processors increase, larger workloads, often with increasing parallelism, may be supported, which can place pressure on main memory and secondary storage resources. A common example is the coexistence of many virtual machines on a single cloud server, for which the system resources, including core count, memory and secondary storage, must be appropriately partitioned. Thus, this increased pressure engenders the urgency of considering holistic tradeoffs in the whole system including main memory and secondary storage.

In this section, we examine these tradeoffs in the context of large, memory and disk-stressing benchmarks (shown in Table 4) to study such a tradeoff. These benchmarks are from the SPEC OpenMP benchmark suite (OMP2012) [75], as well as the NAS Parallel Benchmark Suite (version 3, dataset D) [76]. For each benchmark, we combined secondary storage (SSD) simulations using NVDimmSim [77] with memory and processor results simulated using Sniper and DramSim2. For manufacturing impacts of DRAM and Flash we used a model based on process LCA data [2]. DRAM was studied at both 65nm and 55nm technology nodes, the CPU used 45nm technology, and the Flash SSD used 90nm technology. The secondary storage simulation results from NVDimmSim used in the indifference point calculations were comprised of ten representative samples from the entire execution of each benchmark[4].

3.4.1 Single Benchmark Detailed Analysis

To illustrate the impact of changing memory allocations, we highlight a sample benchmark from the NAS Parallel Benchmark Suite, "sp," which has an order of magnitude larger memory footprint than that of the largest SPEC CPU benchmark. Using a 65nm technology process for the memory, 4GB and 8GB were selected as possible memory allotments. These memory sizes could be considered the provisioning for a virtual machine (VM) run-

[4]The samples were taken during similar phases of the benchmark execution, based on percentage completion, even if the overall runtime changed due to parameter changes such as larger or smaller main memory allocation.

ning such an application on a server which handles multiple VMs, and where the server is likely provisioned with much larger amounts of total memory. When the application memory footprint exceeded the memory available, swapping to the SSD occurred and was modeled for additional operational energy consumption and performance loss. The SSD allocation was a constant 64GB for each experiment.

Figure 14(a) shows the 4GB and 8GB indifference points. The server, desktop, and mobile usage scenarios all require more than 10 years of operation for the 8GB memory to be more sustainable than the 4GB memory. Even for the HPC scenario, more than 5 years is required before the additional memory overcomes its manufacturing and static power deficits. At the 55nm node [Fig 14(b)], the 4GB vs. 8GB comparison becomes more favorable to using the 8GB memory. While the mobile and desktop scenarios still require over 100 years in order for the operational savings to offset the additional manufacturing cost of the 8GB memory, the server scenario now has an indifference point of approximately 5 years, and the HPC scenario is only about 18 months. Therefore, from a sustainability perspective, for workloads resembling sp, HPC systems would seem to value adding memory capacity, while desktop and mobile systems should remain at 4GB (or possibly investigate even smaller memory sizes). The decision for the server scenario depends heavily on the estimated service lifetime of the system, as well as the desired technology node.

The indifference comparison between 65nm and 55nm for 4GB [Figure 14(c)] indicates indifference times of 1.9, 1.8, 0.8, and 0.3 years before 55nm is the sustainable choice for the desktop, mobile, server, and HPC scenarios, respectively. For any 4GB memory system with planned operation time of approximately 2 years (or more), it is more sustainable to use 55nm instead of 65nm technology. For the 8GB comparison, all indifference times increase slightly, with desktop and mobile approaching 3.3 years. This indicates that for benchmarks similar to sp with an 8GB main memory, the decision whether or not to fabricate using 55nm technology instead of 65nm depends whether or not the planned lifetime of the device exceeds approximately 3.3 years.

3.4.2 Multiple Benchmark Analysis

Moving beyond the design space diagrams for a single memory-intensive benchmark, we considered the main memory size tradeoff for the various benchmarks enumerated in Table 4. Figure 15 shows the 4GB vs. 8GB tradeoff at the 55nm technology node. In all cases, it never makes sense for the mobile or the desktop scenario to use an 8GB memory. For ilbdc and bt, the original server scenario indifference point also does not benefit from the increased capacity in terms of holistic sustainability, while for sp workloads the transition does make sense for systems in place longer than 5 years. For the HPC scenario, applications with finite indifference times range from one to seven years to recover from the additional manufacturing investment. The results for the equivalent analysis at the 65nm node (Figure 16) show similar results for most cases with the 65nm memory capacity indifference time tending to be higher than for the 55nm memory capacity indifference time. One exception is the server scenario for bt, which benefits more quickly from 8GB at the 65nm technology node than at 55nm. This indicates that for these relatively legacy nodes, memory expansions at the smaller technology node are more efficient, and typically can make more sense in terms of holistic sustainability than their predecessors.

We also directly compared transitioning from 65nm and 55nm main memory with the same capacity, shown in Figure 17. In the case of the ilbdc benchmark, in every scenario it does not make sense to transition from 65nm to 55nm, since the more costly 55nm manufacturing investment is never recovered. In the case of workloads similar to the cg benchmark, the higher cost of 55nm manufacturing can be recovered, but only if the system is in use for several decades, which is implausible. As one might gather from the similar indifference magnitudes for the remaining benchmarks and scenarios, the indifference design space charts for these benchmarks when comparing 55nm and 65nm are very similar, with slight offsets in magnitude. In other words, unlike the main memory size comparison, the effect of adjusting the main memory technology node (at least between 65nm and 55nm) is largely consistent between benchmarks, and is not as dependent on the individual benchmark characteristics.

3.4.3 Improving Cloud Server Utilization

With continued advances in virtualization, servers have been able to adaptively and more efficiently utilize their available resources. This has effectively allowed servers to improve the effective activity ratio to better approach the HPC scenario, which typically results in the lowest indifference times. In the context of benchmarks and workloads which require significant communication with secondary storage, it therefore makes sense to not only examine the server scenarios at the original cited activity ratio, but also to pursue a more detailed consideration of the holistic sustainability of the server for different workloads across different activity ratios. This can be observed individually for each benchmark in our indifference design space charts (see Figure 18). By keeping the downtime (i.e., sleep) ratio constant (5%) it is possible to determine the activity ratio required to achieve a positive sustainability tradeoff for a desired product lifetime. In Figure 18, we show this for ten, five, and three year indifference times starting from the original server activity ratio, and increasing the activity ratios to achieve those indifference times. In the following discussion, we isolate the activity ratio and indifference point relationship (assuming a 5% sleep percentage) from these design space charts to view them collectively, as well as articulate the activity ratios required to reach various indifference times.

Figure 19 demonstrates the impact of activity ratio on the indifference time of the 4GB vs. 8GB comparison at 55nm for the different possible server scenarios. Applu and bwaves always have infinite indifference time along the entire spectrum of server activity ratios, indicating that for those workloads, no matter the server's configuration it does not make sense in terms of holistic sustainability to use 8GB main memory instead of 4GB main memory. The ilbdc benchmark's indifference time is infinite until the activity ratio reaches at least 36%. The reason an indifference point cannot be reached is that the static power for being active (even if the system is idle) overcomes the dynamic power reduction of the lower technology node, resulting in an operational power increase compared to the older technology node. With both a manufacturing and operational energy increase, the result is an infinite indifference point. The indifference point drops to 20 years at 57% activity, and 10 years at 79% activity. At 100% activity it is 6.8 years.

For the lu benchmark, the original cloud server (about 30% activity) scenario results in approximately 20 year indifference point. At less than a 10% activity ratio, the indifference time becomes infinite. The ten year indifference point can be achieved if the activity ratio can be raised just over 50%. The smallest indifference point is 4.6 years when the activity ratio reaches 100%. Lu has significantly different behavior than the bt benchmark. For the original cloud scenario the indifference point is infinite, but increasing the activity ratio to just over 40% results in an indifference point of 20 years. Reaching an activity ratio of just over 45% drops this to 10 years, 60% drops it to 4 years, 85% drops it to 2 years, and 100% decreases it to 1.5 years. This distinct behavior for the different benchmarks demonstrates the careful consideration which must be taken when choosing particular server configurations and activity ratios for estimated workloads and lifetimes. When trying to remain as sustainable as possible, knowledge of these potential knees can be essential in avoiding relatively inefficient regions.

Figure 20 also examines the relationship between the activity ratios and indifference times for the original server scenario between 4GB and 8GB main memory sizes at 65nm. As with the 55nm analysis, both applu and bwaves always have an infinite indifference time. Both ilbdc and lu have a minimum indifference time of over 20 years, indicating for these workloads for almost all conceivable lifetimes 4GB is the more sustainable choice. Bt and sp have similar results to each other, bt and sp reach a 4 year indifference point at 57% and 90% activity ratio, and approximately 3 year indifference point at 100% activity ratio.

The server sensitivity analysis for 65nm vs. 55nm can be observed in Figure 21. Unlike in the case of the memory capacity analysis, in the process node comparison ilbdc always has an infinite indifference time. Thus, while in some cases for ilbdc use of a larger memory size makes sense, for no possible activity ratios does the more advanced process node make sense in terms of sustainability. The other outlier was cg, which has its lowest possible indifference time of 33 years. For the rest of the benchmarks, no matter what activity ratio is achieved, the 55nm node will be more sustainable after at most 8 years. This is very dissimilar from the memory comparison, which for all benchmarks in all cases had an activity ratio below which 4GB would always be more efficient than 8GB. For the bt, lu, sp, and bwaves workloads they all have a maximum indifference time (calculated using a 1% activity ratio)

between 2 and 3 years, clearly indicating that these workloads benefit significantly from the technology node upgrade. Even the benchmarks with slightly higher maximum indifference points, applu (4.8%) and ua (8%), remain below a 4 year indifference time for activity ratios above 25%. In general, servers running similar workloads should benefit substantially from the technology node upgrade. However, there are the important exceptions of ilbdc and cg. A staple of cg (conjugate gradient) is irregular memory access and communication, and ilbdc is also a solver (3D lattice Boltzman flow) which may share similar irregular characteristics. Thus, these applications may have less locality and require heavier SSD usage, limiting their performance and energy improvements from more advanced memory. Thus, servers which may experience similar communication patterns may be more sustainable at 65nm vs. the smaller 55nm.

3.5 Conclusion and Future Work

We presented a holistic sustainability evaluation and prediction tool called GreenChip. GreenChip allows detailed manufacturing and use phase energy calculation and comparison of integrated circuits used for constructing computing systems from processors, main memory, and solid state storage. We presented several case studies that evaluate processor, cache, and main memory choices. In many cases, indifference and break even times can be compared with typical expected lifetimes. For example in Case Study I (Section 10.2), the break even points for upgrading desktop computers and mobile devices often exceeded five years and replacement cycles for such systems is often less than two years. Also, it often did not make sense to upgrade servers even when the use phase gain was particularly helpful, recalling that the 45nm→28nm upgrade time still exceeded three years.

One interesting trend is that chasing higher core counts, caches, and memory/storage sizes may not always be the most sustainable solution, and there is potential with reaching fabrication technology limits for manufacturing cost to become an increasingly important factor in design choices. For example in Case Study II (Section 3.3) the results pointed

to a moderate LLC capacity (1MB) providing the best compromise of sustainability and performance.

Additionally, considering main memory sizes, a moderate main memory can outperform a larger main memory from an environmental perspective as shown in Case Study III (Section 3.4). Furthermore, our examination revealed that advances in main memory technology nodes, while beneficial for most applications, had several applications which were significantly less sustainable at the smaller, more operational energy efficient node. The characteristics of these applications were irregular main memory accesses from different solvers, indicating servers performing these applications should potentially consider other factors, such as larger memory allotments or more efficient secondary storage for achieving holistic sustainability.

GreenChip provides a flow to evaluate many future design choices for holistic sustainability such as server consolidation with larger core counts and memory capacity. Incorporating more holistic evaluations into standards such as Energy Star and presenting sustainability metrics for consumer electronics can empower consumers to make more informed choices and lead to new marketing strategies resulting in a more sustainable computing electronics industry.

In our future work we plan to develop more precise models that can extend the manufacturing estimates down to smaller technologies and incorporate the impacts of emerging technologies such as 3D CMOS, CMOS compatible extensions such as non-volatile memories, and others. Coupled with further scaled operational energy estimation flows, this will allow comparisons of more modern designs that leverage some of the expensive process techniques such as multiple patterning lithography.

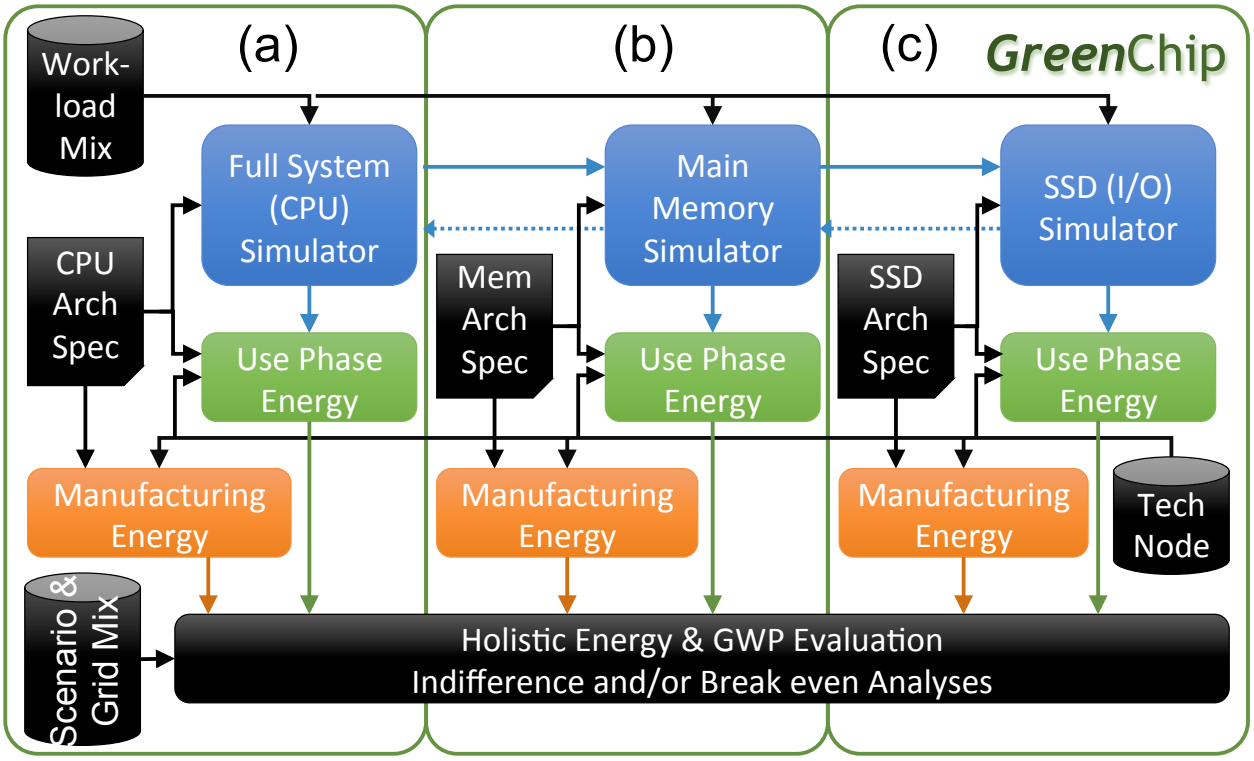


Figure 4: GreenChip evaluation flow.

Table 1: Multi-program Workloads and Memory Footprints for the Parsec and SPEC Benchmarks. Low (L), Medium (M), and High (H) represents those respective memory footprints.

Multi-Program Workload	*Abbr.*	*Memory Footprints*
Parsec Four Program Workloads		
blackscholes-vips-streamcluster-swaptions	BVSS	L-L-L-L
canneal-x264-blackscholes-vips	CXBV	H-H-L-L
canneal-x264-freqmine-dedup	CXFD	H-H-H-H
raytrace-fluidanimate-freqmine-bodytrack	RFFB	L-L-H-H
SPEC-CPU2006 Four Program Workloads		
bzip2-zeusmp-cactusADM-bwaves	BZCB	H-H-H-H
bzip2-gobmk-hmmer-libquantum	BGHL	H-L-L-L
GemsFDTD-lbm-milc-namd	GLMN	H-M-M-L
lbm-perlbench-leslie3d-astar	LPLA	M-M-M-M
mcf-sjeng-cactusADM-calculix	MSCC	H-L-H-M
povray-h264ref-calculix-soplex	PHCS	L-L-M-M
SPEC-CPU2006 Eight Program Workloads		
bzip2-gcc-zeusmp-cactusADM-mcf-GemsFDTD-milc-soplex	HIGH8	H-H-H-H-H-H-M-M
gobmk-hmmer-h264ref-gromacs-namd-povray-tonto-libquantum	LOW8	L-L-L-L-L-L-L-L
gobmk-namd-lbm-perlbench-calculix-soplex-bzip2-gcc	MIX8	L-L-M-M-M-M-H-H

Table 2: Activity and Sleep Scenarios

Name	*Activity Ratio* r_A	*Sleep Ratio* r_S
Server	30%	5%
HPC	95%	5%
Desktop	17%	77%
Mobile	90%	92%

Table 3: Manufacturing costs for chips at different process nodes following product trends (pseudo ISO-area) [1, 2].

Process Node (nm)	90	65	45	28
Core Count	1	2	4	8
LLC size	1MB	2MB	4MB	8MB
Area (mm^2)	207	227	207	158
Manufacturing Energy (MJ)	124	148	164	598

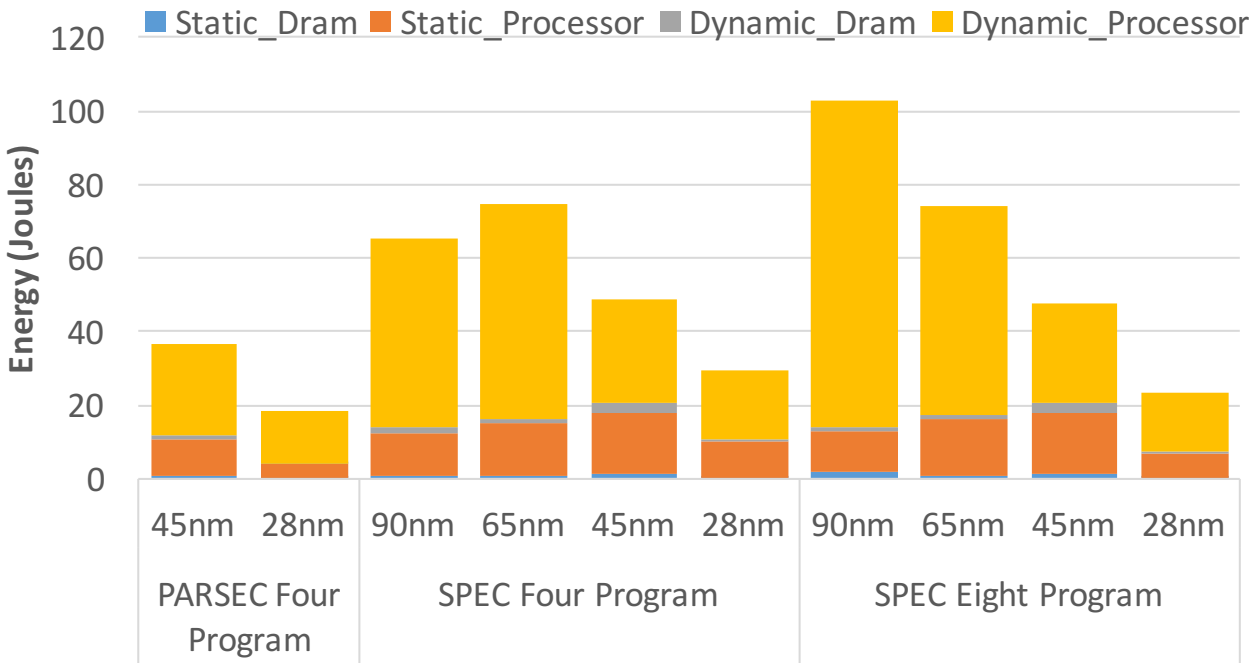
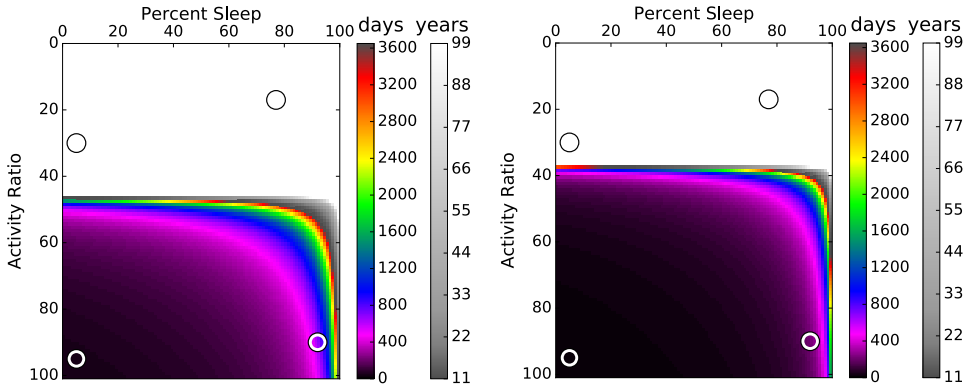
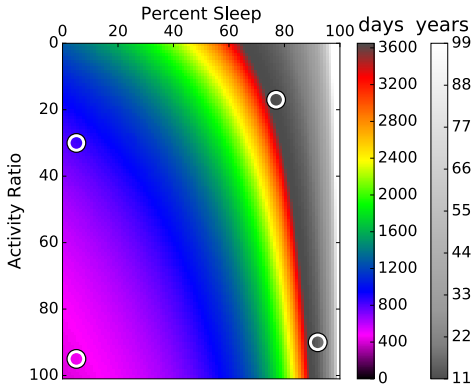


Figure 5: Joules Per 10billion instructions for the Parsec and SPEC multiprogram workloads with different process node. All are run with the same chip area, as part of the iso-area comparison.



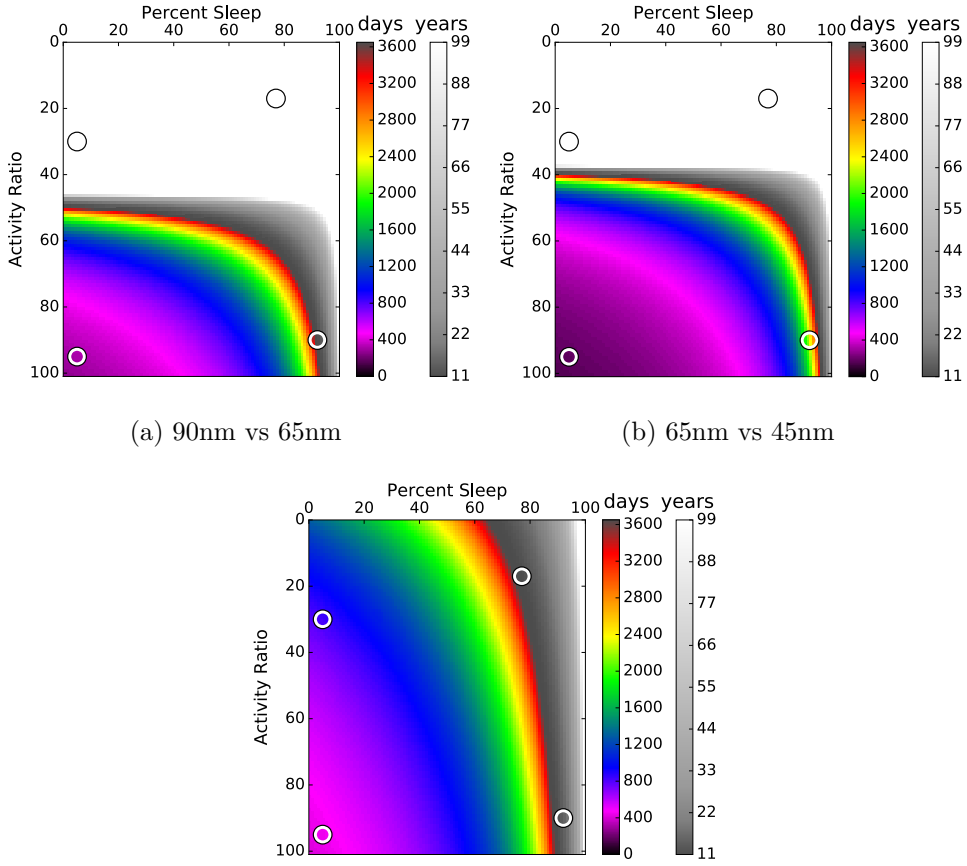
(a) 90nm vs 65nm

(b) 65nm vs 45nm



(c) 45nm vs 28nm

Figure 6: Indifference points (t_I) for the pseudo ISO-area comparisons across adjacent technology nodes for GLMN highlighting four usage scenarios.



(a) 90nm vs 65nm

(b) 65nm vs 45nm

(c) 45nm vs 28nm

Figure 7: Break even time (t_B) to move to the next technology node in a pseudo ISO-area comparison for GLMN highlighting four usage scenarios.

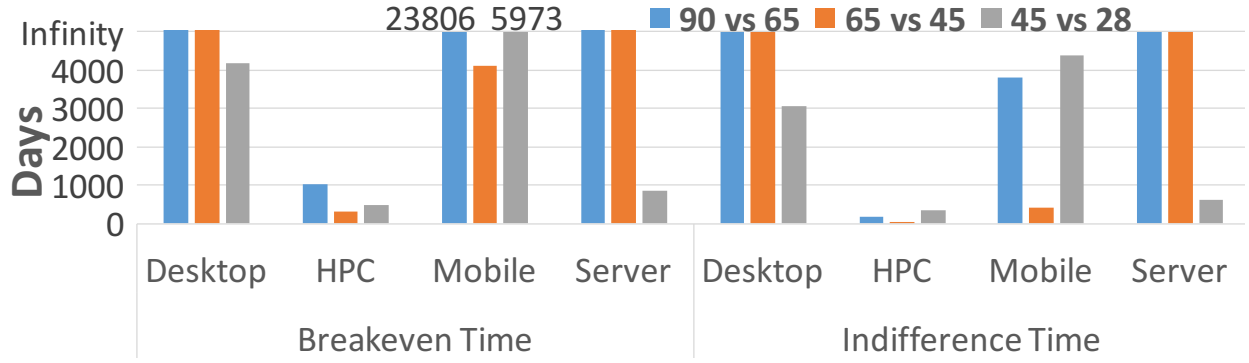


Figure 8: Average break even times and indifference points across all benchmarks for pseudo iso-area comparison.

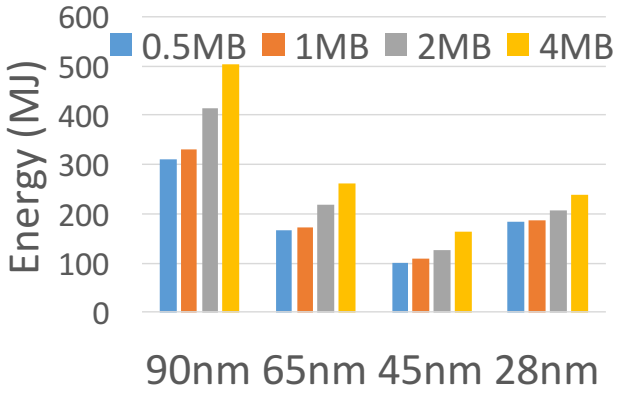


Figure 9: Manufacturing energy for four-core systems with varying LLC capacities across technology nodes.

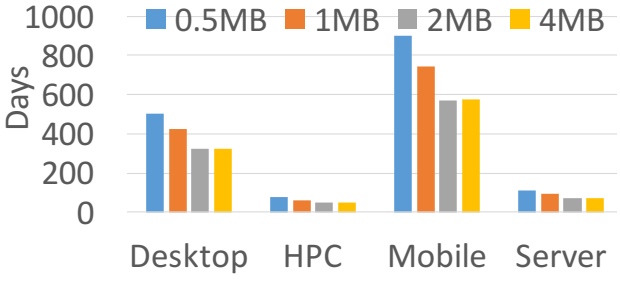


Figure 10: Indifference time (t_I) between 45nm and 28nm for multiple LLC cache capacities.

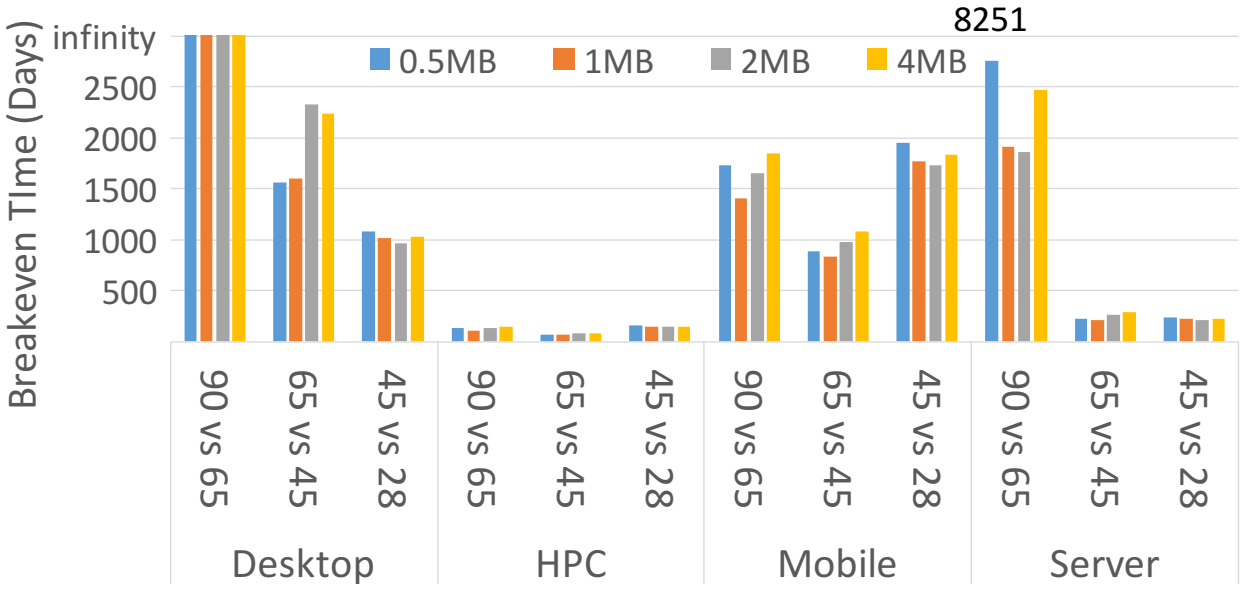


Figure 11: Average break even times across all the benchmarks, iso-architecture comparison with 4 cores. Note: All 90nm vs. 65nm data points for desktop except one benchmark never broke even. Also, one benchmark for the server at 0.5MB never broke even, so the average is the remainder of the benchmarks

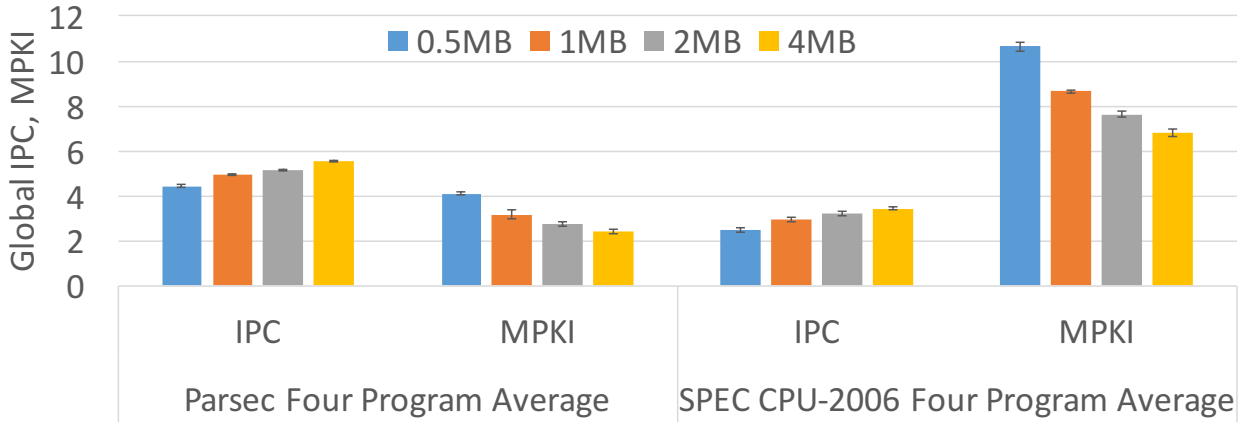


Figure 12: Global IPC and MPKI averaged for Parsec and SPEC four process multi-program workloads across different technology nodes while varying LLC capacity. (min and max shown by error bars)

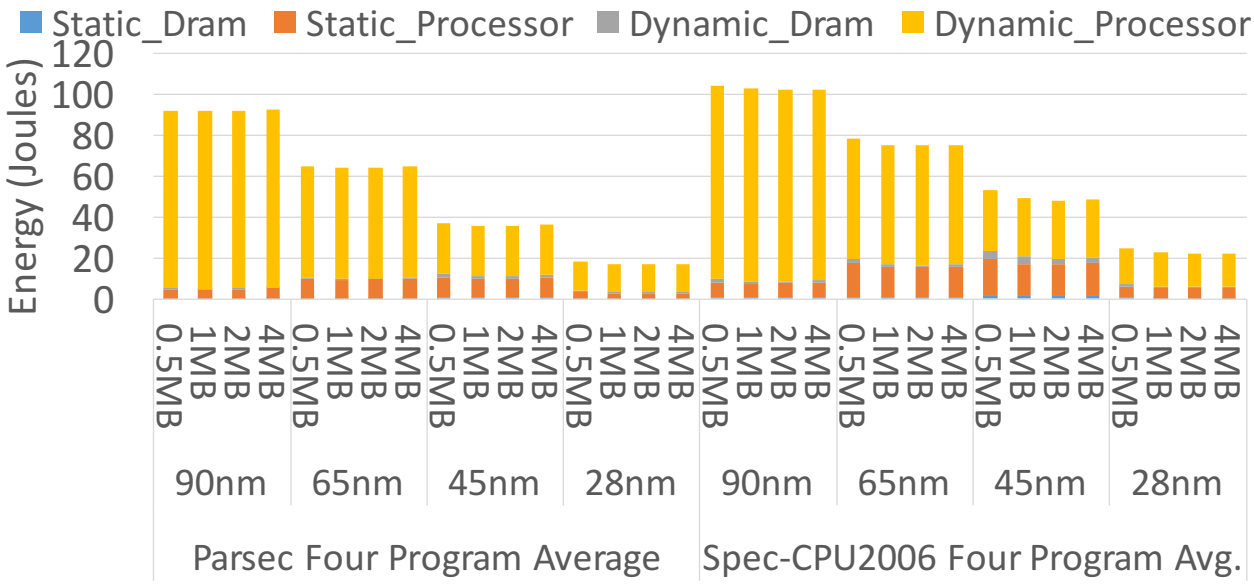
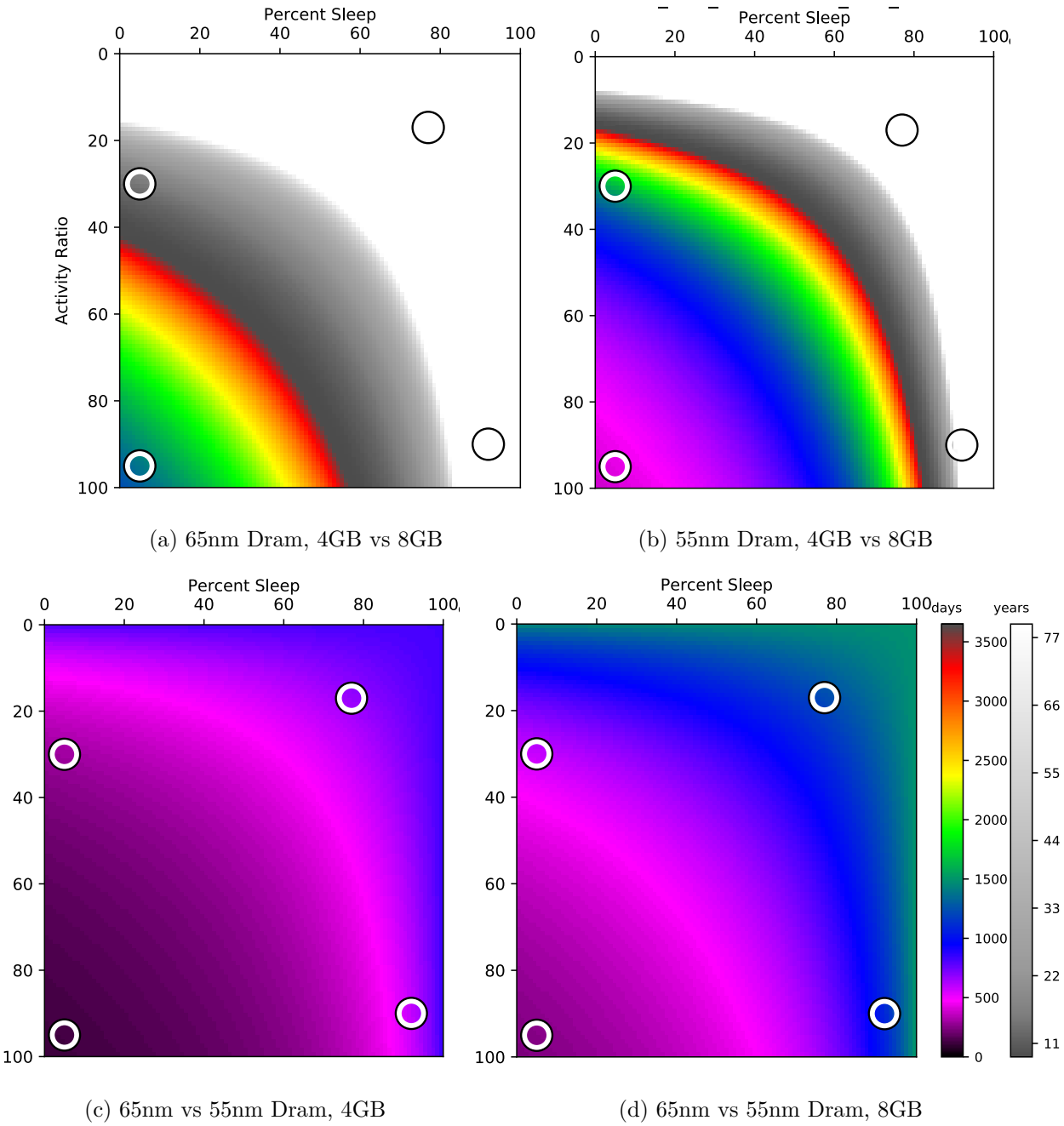


Figure 13: Joules Per 10 billion instructions averaged for Parsec and SPEC four process multi-program workloads across different technology nodes while varying LLC capacity.

Table 4: Disk-Stressing Benchmarks

Name	*Mem(GB)*	*Name*	*Mem(GB)*	*Name*	*Mem(GB)*
SPEC-OMP2012		**NAS Parallel Benchmarks**			
applu	4, 8	bt	4,8	cg	4
bwaves	4, 8	lu	4,8	ua	4
ilbdc	4, 8	sp	4,8		



(a) 65nm Dram, 4GB vs 8GB

(b) 55nm Dram, 4GB vs 8GB

(c) 65nm vs 55nm Dram, 4GB

(d) 65nm vs 55nm Dram, 8GB

Figure 14: Memory indifference points for the NAS Parallel Benchmark "sp."

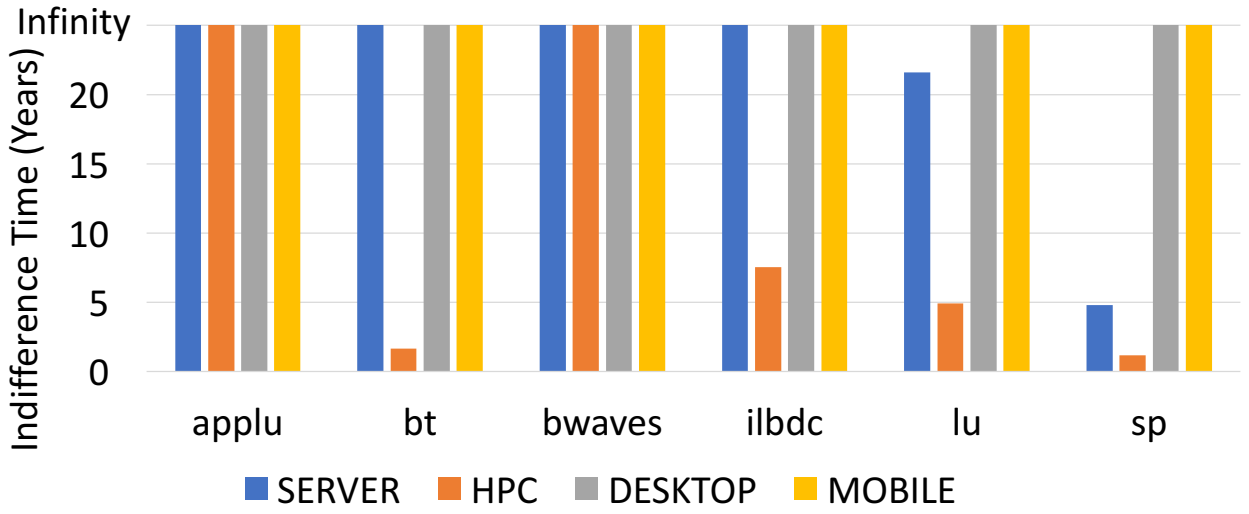


Figure 15: Indifference times (years) for 4GB vs. 8GB comparisons at 55nm. Note that the scale is different from Figure 16.

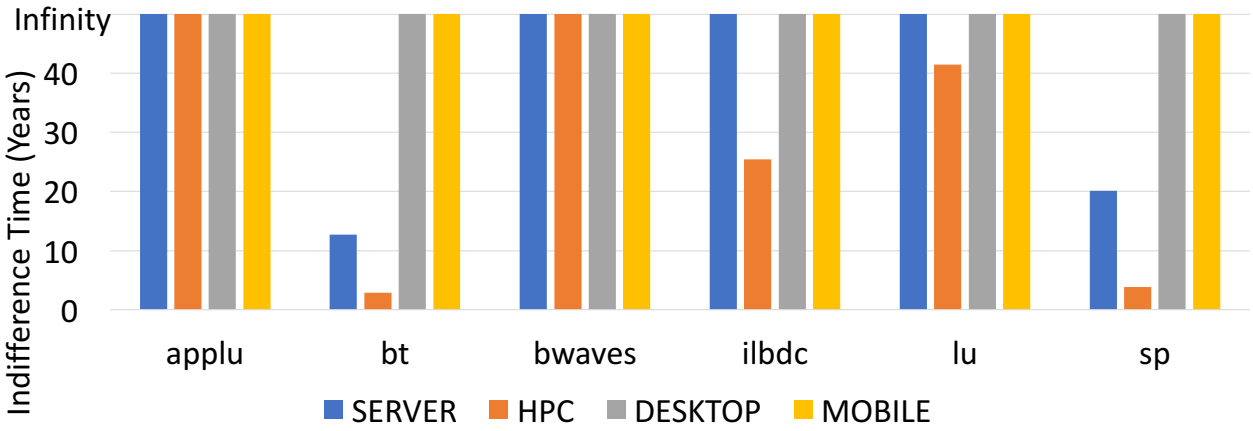


Figure 16: Indifference times (years) for 4GB vs. 8GB comparisons at 65nm. Note that the scale is different from Figure 15.

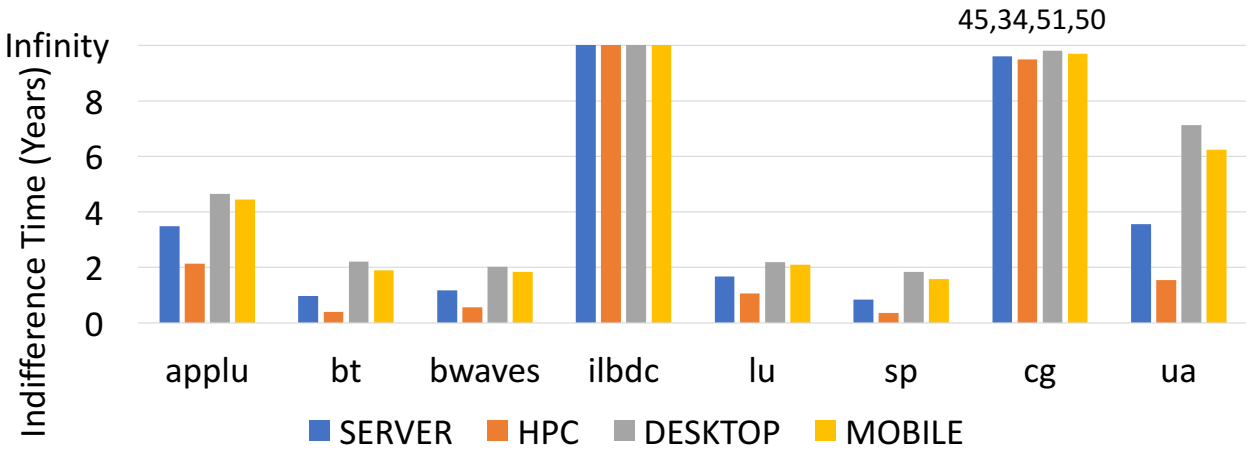


Figure 17: Indifference times (years) for 65nm vs. 55nm comparisons at 4GB.

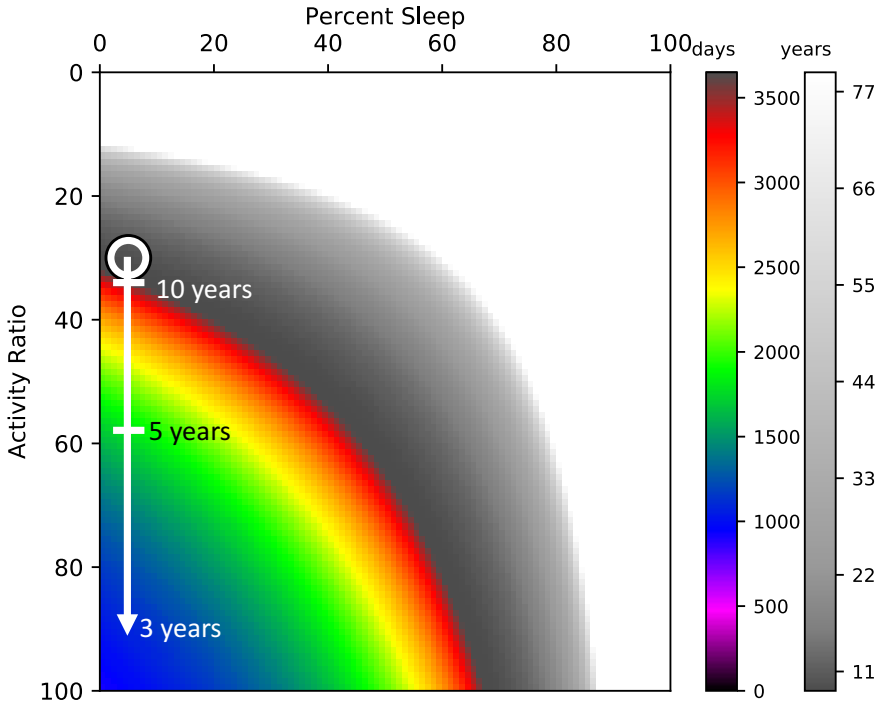


Figure 18: An example of the impact of the different activity ratio options and their corresponding impacts on indifference point for servers. The circle represents the original activity ratio and the arrow shows increased activity ratios to achieve ten, five, and three year indifference points.

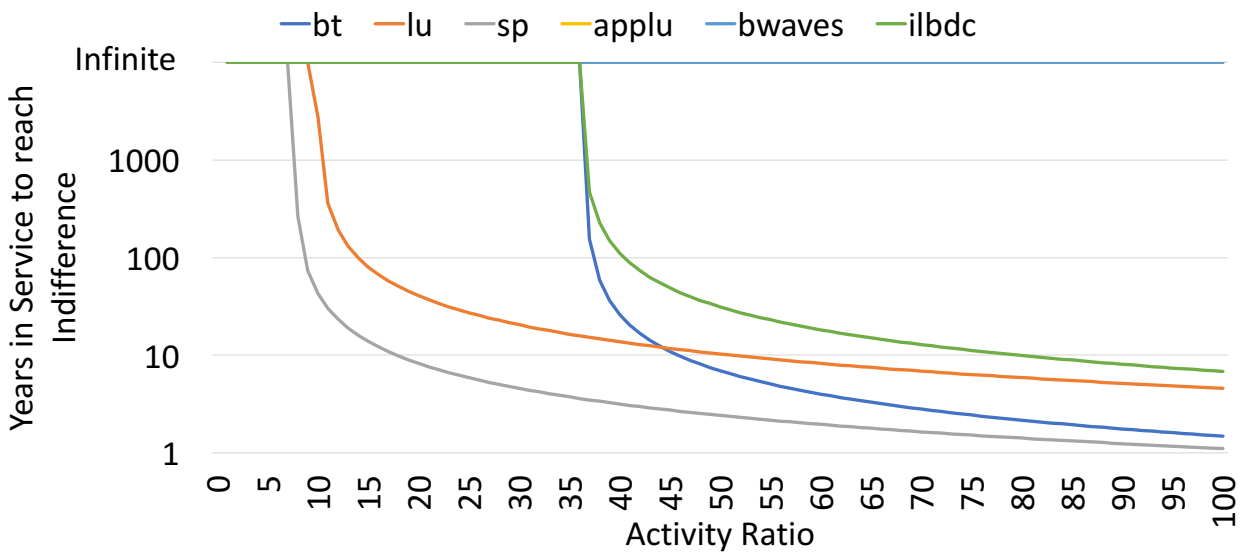


Figure 19: Indifference times (years) for the full range of server activity ratios at the 55nm technology node for the comparison between 4GB and 8GB main memory sizes. Applu and bwaves always have an infinite indifference time.

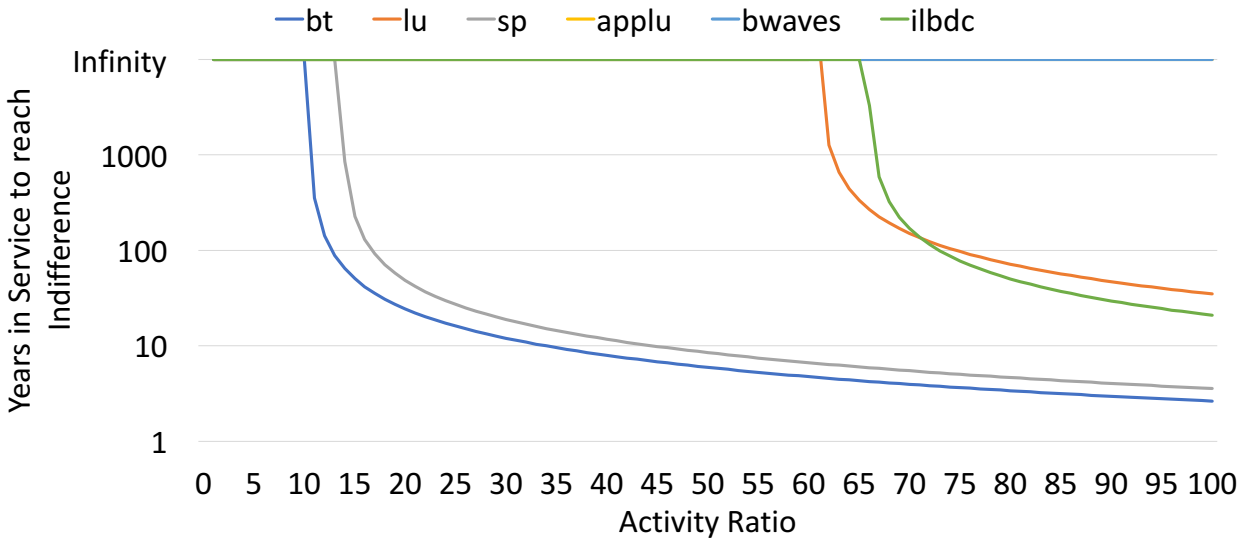


Figure 20: Indifference times (years) for the full range of server activity ratios at the 65nm technology node for the comparison between 4GB and 8GB main memory sizes. Applu and bwaves always have an infinite indifference time.

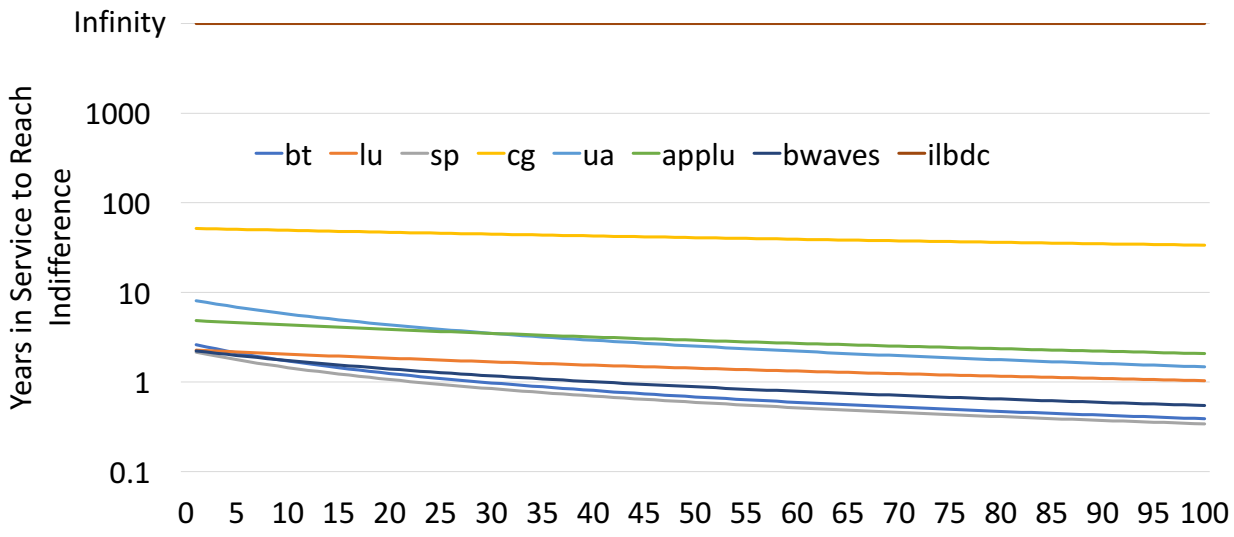


Figure 21: Indifference times (years) for the full range of server activity ratios at the 4GB memory size for the comparison between 65nm and 55nm technology nodes. Cg's curve begins at 33 years, and ilbdc always has an infinite indifference time.

4.0 GreenASIC

4.1 Environmental Impact Model

Computer-aided design tools create useful abstractions of the fabrication process that bridge the gap between details of the process and the hardware description from the IC designer. A careful analysis and characterization of the sustainability impact of these choices will inform both the designer and the tools used in the design process. However, designers currently do not have a way to visualize at design time the effect their decisions have on the sustainability and environmental cost of IC fabrication. Thus, we propose a parameterized environmental impact model based on the approach presented for 130nm [3]. From this model we use a combination of economic information about the relative costs between steps [78] and reported process LCA environmental data [2] to determine environmental costs per process step at different technology nodes. This allows the process steps to be adjusted in order to examine the impacts of changes in the process such as changing the metal stack. We discuss a methodology to quantify these impacts for a fixed process in the next section.

4.1.1 Parameterizing the Fabrication Process

Determining the impact of changing the number of process steps when making adjustments such as altering the metal stack for a fixed CMOS technology requires a method to breakdown the environmental impact at each stage of the process. Then by changing the number of steps in the particular process, the aggregrate impact can be estimated. Table 5 shows a detailed breakdown of the parameterized process model for 130nm technology. In Table 5(a) the number of process steps are enumerated for different numbers of metal layers. Ion implantation and rapid thermal processing furnace steps remain unchanged as they apply only to the underlying CMOS layer. Other steps decrease as the number of metal layers decreases including chemical vapor deposition (CVD), cleaning, photo-lithography, etching, metallization, and polishing (CMP). For each metal layer reduced we decrease CVD

and metallization by one step due to eliminating the dielectrics and metal of those layers. Photo-lithography, etching, polishing, and furnace are each reduced by two steps per layer due to the separate via and interconnect sublayers of the stack.

Table 5: Parameterized model [3] with 130nm process LCA data [2] differentiating 10, 8, and 6-layer metal stacks for a 200mm wafer.

	Steps(a)			Process Parameters(b)					5500 Wafer/Week(c)			Energy[kWh/Wfr](d)		
unit op	metal layers			wafer	wafer	T	active	idle	machines	active	idle	metal layers		
	10	8	6	per run	per h	(h)	pwr(kW)	pwr(kW)	per Step	time	time	10	8	6
implant	16	16	16	25	20	0.050	266	148	3	275	127	221	221	221
CVD	15	13	11	10	15	0.067	103	90	3	336	36	109	94	80
clean	39	35	31	50	150	0.007	800	750	1	36	98	213	191	169
furnace	25	21	17	150	35	0.029	251	192	2	157	111	185	155	126
furnace-RTP	7	7	7	1	10	0.1	21	20	5	550	120	17	17	17
lithography	31	27	23	1	60	0.017	997	413	1	91	43	522	454	387
dry etch	28	24	20	1	35	0.029	365	95	2	157	111	295	258	221
ash etch	31	27	23	1	20	0.05	8	7	3	275	127	13	11	9
metallization	13	11	9	1	25	0.04	663	360	2	220	48	359	313	266
CMP	22	18	14	1	25	0.04	172	47	2	220	48	153	130	106
											Total:	2087	1845	1603

Table 5(b) provides details on the equipment used to complete each step including its active and idle power consumption. In some cases the equipment works on a single wafer and in other cases it works on a batch. From this, the throughput of a piece of equipment is determined. Depending on the throughput requirement of the facility, the number of machines required to meet the demand can be determined (Equation 4.1). This is shown for a unit process step in Table 5(c) including the total active versus idle time from the aggregate of the machines. The total energy from all the process steps of that type is shown in Table 5(d) using the active and idle times for the number of process steps.

$$Machines = \lceil wafers * steps_{op}/throughput \rceil \tag{4.1}$$

To examine more closely the specific example of metalization, the sum of the energies in Table 5(d) reveal that it is 13% and 30% more expensive to increase from six to eight and ten metal layers, respectively, per wafer at 130nm technology. This indicates that the number of metal layers can significantly impact the overall cost of fabricating CMOS at 130nm. To

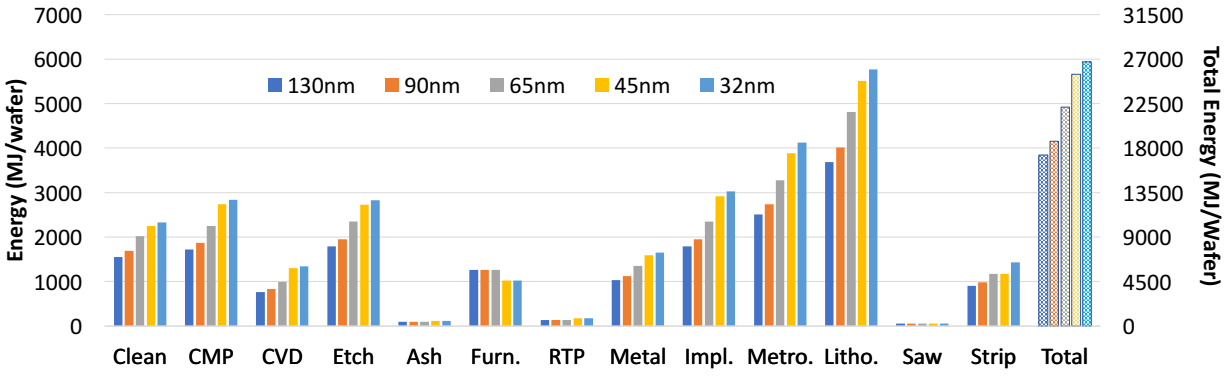


Figure 22: Total energy consumption across the different process steps for technologies between 130nm and 32nm as calculated by our process model. The total across all steps [2] (patterned bars of the same colors) use the secondary axis.

determine the impact of these decisions as we continue to shrink requires that the process model be developed at each technology node. Thus, we provide a scaling method for this model in the next section.

Table 6: Process steps and scaled 300mm wafer energy for 90nm, 65nm, 45nm, and 32nm processes based on the IC model from Figure 22.

unit op	90nm						65nm						45nm						32nm					
	Steps			Energy[kWh/Wfr]			Steps			Energy[kWh/Wfr]			Steps			Energy[kWh/Wfr]			Steps			Energy[kWh/Wfr]		
	Metal Layers						Metal Layers						Metal Layers						Metal Layers					
	10	8	6	10	8	6	10	8	6	10	8	6	10	8	6	10	8	6	10	8	6	10	8	6
Implant	16	16	16	543	543	543	16	16	16	651	651	651	20	20	20	812	812	812	20	20	20	1742	1742	1742
CVD	15	13	11	266	231	195	15	13	11	319	277	234	17	15	13	361	319	276	17	15	13	775	684	592
clean	39	35	31	522	469	415	39	35	31	627	562	498	39	35	31	625	561	497	39	35	31	1341	1203	1066
furnace	25	21	17	416	350	283	25	21	17	416	350	283	17	13	9	283	217	150	17	13	9	283	217	150
furnace (RTP)	7	7	7	39	39	39	7	7	7	39	39	39	9	9	9	50	50	50	9	9	9	50	50	50
lithography	31	27	23	1280	1115	949	31	27	23	1536	1337	1139	31	27	23	1531	1333	1136	47	43	39	5098	4664	4230
dry etch	28	24	20	633	543	452	28	24	20	760	651	543	28	24	20	758	649	541	46	42	38	2555	2333	2111
ash etch	31	27	23	32	28	24	31	27	23	32	28	24	31	27	23	32	28	24	31	27	23	32	28	24
metallization	13	11	9	369	312	256	13	11	9	443	375	307	13	11	9	442	374	306	13	11	9	948	802	656
CMP	22	18	14	636	520	405	22	18	14	763	624	486	22	18	14	761	622	484	22	18	14	1633	1336	1039
Total:				**4737**	**4149**	**3561**				**5586**	**4895**	**4204**				**5654**	**4965**	**4275**				**14457**	**13059**	**11661**

4.1.2 Scaling the Parameterized Model

Detailed, process specific information is not widely available per fabrication step at each process node preventing the direct construction of a parameterized model where this information is not available. However, aggregate environmental data has been published for CMOS technologies reaching 32nm [2]. Thus, if we assume this aggregated data is for a single CMOS layer and uses the maximum size metal stack available at that technology we can scale the parameterized model. Recalling from the background that environmental impact trends tend to follow economic cost trends, we used economic data [78] and environmental data [3] reported for 130nm to determine the relative weights of the individual process steps. We also adapted the number of steps in the core process based on changes to the process at different feature sizes as reported in ITRS [79]. For example, at 45nm and below, dopant diffusion and thermal oxidation for the gate oxide are no longer employed, and additional implantation and CVD process steps are performed in their place. We adjust the relevant step parameters accordingly, adding implantation steps, RTP steps, and CVD steps while removing furnace steps. From 130nm to 65nm we assumed the metal stack contained up to eight layers, for 45nm and 32nm we assume the stack contained up to ten layers.

The results of our model for energies consumed for different process steps is shown in Figure 22 at technologies between 130nm and 32nm. The resulting output of the model demonstrates that there are typically linear increases in expected process steps between technologies such as lithography, metrology, deposition, and etching, which was one of our sanity checks. Additionally, the rate of increase (slope) in lithography and metrology is faster than other steps such as deposition and etching, which also matches industry trends.

Moreover, certain process steps such as the furnace step remain relatively invariant to process node unless the process steps change. This provides confidence that the model is representative of the actual process impacts. Moreover, although our model data is reported in energy, making it relatively easy to compare between fabrication and operational phases, the model is capable of reporting other environmental impacts such as GWP, carcinogenic chemicals, volatile organic compounds (VOCs), and wastewater generated. We show this in

more detail in the output of our experiments for different circuit benchmarks in the following section.

Based on the model we conducted a similar analysis, modifying the number of metal layers shown in Table 5 for the technology nodes represented, with the results for a 300mm wafer shown in Table 6[1]. The model demonstrates that increasing the metal layers costs 14%, 32-33% more embodied energy per wafer for moving from six layers to eight and ten layers, respectively, for 90nm, 65nm, and 45nm nodes.

Our model assumes use of LELE (2× litho-etch) at the 32nm node as the physical limits of 193nm+immersion lithography is about 30nm [79]. As a result, the lithography and etch portions of the lower chip levels that define the transistors become more expensive. Thus, increasing the metal layers that use traditional lithography have a reduced relative impact, dropping to 11%, and 24% for moving from six layers to eight and ten layers, respectively. While this may seem to imply that $< 32nm$ technologies may minimize the impact of pruning the metal stack, some metal-specific challenges are expected to keep it relevant, such as additional steps for porous and air-gapped dielectrics, etc [79].

Unfortunately, determining the environmental impact per wafer is not enough to truly understand the impact of these design choices on fabricated ICs. Changing the metal stack for a particular process can have a significant impact on the area or energy of the resulting IC. As a result, the fabrication impacts per die is impacted positively by reducing the process steps, but impacted negatively by potentially increasing the die area, latency, and/or power. As a result, the overall savings (or potential increase) is unknown without actually determining a particular placement and routing in the target technology. Thus, in the next section, we examine the environmental impacts of changing the metal stack for a series of benchmark designs prepared for fabrication using representative technology design kits at 130, 90, and 65nm.

[1]Note, the results from Table 5 are for a 200mm wafer. To compare the results between these two tables requires an approximately 2.25× scaling factor due to the area change.

4.2 Results

Our parameterized model, in combination with running the design through a commercial design flow can provide insights into the environmental impact of a design. In this section we explore in more detail the process change of different metal stacks and the impact on holistic sustainability through indifference analysis.

4.2.1 Experimental Setup

To study the impacts of using eight and six layer metal stacks, we ran eight benchmark designs from the IWLS benchmark suite [80] ranging from 6,521 to 47,223 cells through synthesis using Cadence RTL Compiler and commercial standard cell libraries for 130nm, 90nm, and 65nm processes. These designs include controllers (DMA, vga_lcd), encryption block ciphers (aes), and subsets of circuitry from micro processors (DSP, b17,b18,b22). The benchmark designs were run through place and route with six and eight layer metal stacks using Cadence Encounter Digital Implementation (EDI) System, which produced estimated area, power, and timing results. We used our scalable parameterized model (Section 4.1) to estimate manufacturing impacts. To make comparisons, we utilize the energy metric so as to allow for holistic comparisons between the manufacturing and operational phases.

4.2.2 Minimizing Purely Manufacturing Impacts

First we compared a minimal area design for eight and six layer metal stacks at each technology node. This resulted in the fabrication savings shown in Table 7. In some designs below 130nm, there was no area increase, indicating the highest cell utilization for both designs was identical and six metal layers were sufficient for a working IC with the original area. In other cases, such as with wb_con at 65nm, the negative impact of the reduced die/wafer compared to the pure fabrication gains (14.11%) can be observed as the difference between the pure fabrication gains and the final fabrication savings. Unfortunately, such tight area constraints resulted in significant increases in operational power and delay, offset-

ting the manufacturing savings. Thus, in the next section we examine a "relaxed" design that considers impacts from both manufacturing and operational phases.

4.2.3 Optimizing for Holistic Sustainability

To balance the operational and manufacturing impacts, we determined the minimal area design for both eight and six metal layer processes, such that the area was as low as possible while producing reasonable power and timing results. As the design approaches the maximum cell utilization, power and latency spike for fractional gains. Backing off the area requirement to just slightly under the maximum results in much more reasonable power and delay, which can be manually tuned to find the knee that allows the selection of a minimal area prior to the power and performance spike. In all cases, this chosen cell utilization was within 6% of the maximum possible cell density.

The results for these experiments are shown in Table 8. For all benchmarks run at 65nm and 90nm, this constrained minimum area was the same between eight and six layers. Consequently, the fabrication savings for the results in Table 8 at 65nm and 90nm were independent of benchmark and were 14.11% and 14.55%, respectively. These gains come directly from reducing the metal layers in our model as discussed in Section 4.1. For 130nm, the maximum cell utilization for eight and six layers was often separated by 5% or more, because the six layer design did not have enough routing flexibility to replicate the original cell density. As a result, the largest utilization with reasonable power and delay was different for eight and six layers, resulting in different area and resulting fabrication savings per benchmark.

While the latency generally rises when changing from eight to six layers, particularly for minimal area designs, the trend here is more complicated. Because the area is slightly

Table 7: Fabrication savings when using the minimum possible area.

	DSP	DMA	aes	wb_con	vga_lcd	b17	b18	b22	Avg.
Fabrication Savings, 65nm	14.11%	10.47%	14.11%	13.22%	10.54%	14.11%	14.11%	13.21%	12.99%
Fabrication Savings, 90nm	13.59%	14.55%	14.55%	10.56%	14.55%	14.55%	14.55%	14.55%	13.84%
Fabrication Savings, 130nm	6.96%	10.88%	10.72%	13.43%	7.38%	7.00%	6.49%	13.24%	9.51%

relaxed, the six layer designs can have reductions in delay over the eight layer designs, but often with increases in power, and vice-versa. In two cases, b17 at 65nm and vgalcd at 90nm, both the power and the delay decrease when moving from eight layers to six layers. Intuitively, for these cases it makes no sense to fabricate the eight layer stack over the six layer stack. Otherwise, a more holistic sustainability analysis is required to combine both the manufacturing and the use phase results.

Thus, we adopt indifference analysis to calculate the time when the total energy of eight and six metal layers are equivalent for both use and manufacturing phase (indifference point) using Equation 6.1 [6], where P_0 and P_1 represent the total use-phase power of two systems and M_0 and M_1 represent the manufacturing energy of those respective systems. The use-phase powers in the denominator of the expression can be calculated based on different usage scenarios (Equation 6.2) [5], where r_S is the sleep ratio, r_A represents the active to idle ratio, P_D is the dynamic power, P_S is the static power, and P_L is the sleep power. For each benchmark, we calculate the indifference points for four active and sleep scenarios: a

Table 8: Results from an exploration of the effects of using eight metal layers versus six metal layers in the same process. Results are within 6% of max cell utilization.

		DSP	DMA	aes	wb_con	vga_lcd	b17	b18	b22	Avg.
65nm	# of Cells	18512	7250	20695	23613	32847	18445	37410	18457	22187
	Eight Layer Power (mW)	19.977	24.306	48.677	20.471	385.626	107.456	56.696	57.968	—
	Six Layer Power (mW)	20.641	24.337	49.901	22.863	397.008	106.207	57.608	63.146	—
	Power Increase	3.32%	0.13%	2.51%	11.68%	2.95%	-1.16%	1.61%	8.93%	3.75%
	Eight Layer Delay (ns)	1.578	2.19	1.807	3.32	1.439	3.211	3.274	3.846	—
	Six Layer Delay (ns)	1.564	2.2	1.762	3.431	1.392	3.193	3.723	3.796	—
	Delay Increase	-0.89%	0.46%	-2.49%	3.34%	-3.27%	-0.56%	13.71%	-1.3%	1.13%
	Fabrication Energy eight layers (MJ)	0.069	0.029	0.063	0.059	0.257	0.73	0.118	0.052	—
	Fabrication Energy six layers (MJ)	0.059	0.025	0.054	0.05	0.22	0.063	0.102	0.044	—
90nm	# of Cells	17897	6303	14325	22014	31624	12096	36062	16904	19654
	Eight Layer Power (mW)	19.961	27.606	24.386	21.189	250.792	24.081	73.903	59.76	—
	Six Layer Power (mW)	20.215	27.784	25.595	26.218	250.24	24.341	73.016	64.521	—
	Power Increase	1.27%	0.64%	4.96%	23.73%	-0.22%	1.08%	-1.20%	7.97%	4.78%
	Eight Layer Delay (ns)	2.144	2.02	1.918	3.769	2.636	3.073	3.241	2.598	—
	Six Layer Delay (ns)	2.265	1.978	2.062	3.765	2.537	3.171	3.26	2.688	—
	Delay Increase	5.64%	-2.08%	7.51%	-0.11%	-3.76%	3.19%	0.59%	3.46%	1.81%
	Fabrication Energy Eight layers (MJ)	0.089	0.037	0.043	0.072	0.256	0.052	0.277	0.061	—
	Fabrication Energy Six layers (MJ)	0.076	0.031	0.036	0.062	0.219	0.044	0.237	0.052	—
130nm	# of Cells	19253	10835	28996	25372	47223	20394	38825	25520	27052
	Eight Layer Power (mW)	29.520	36.313	32.491	41.413	234.724	49.212	109.897	33.319	—
	Six Layer Power (mW)	33.828	37.086	33.986	48.524	236.017	47.876	112.7113	32.959	—
	Power Increase	14.6%	2.13%	4.6%	17.17%	0.55%	-2.71%	2.56%	-1.08%	2.45%
	Eight Layer Delay (ns)	2.978	2.677	2.809	4.108	4.59	3.234	4.451	6.889	—
	Six Layer Delay (ns)	2.829	2.688	2.872	3.933	4.6	3.24	4.432	7.088	—
	Delay Increase	-5.0%	0.41%	2.24%	-4.26%	0.13%	0.19%	-0.43%	2.89%	0.71%
	Fabrication Energy Eight layers (MJ)	0.168	0.09	0.107	0.151	0.639	0.114	0.342	0.129	—
	Fabrication Energy Six layers (MJ)	0.166	0.082	0.096	0.14	0.606	0.104	0.31	0.112	—
	Fabrication Savings	**1.63%**	**9.2%**	**10.48%**	**6.88%**	**5.16%**	**8.16%**	**9.19%**	**13.13%**	**7.98%**

mobile computing system, a cloud server, a high-performance computing (HPC) system, and a desktop system [73, 74, 2].

$$t_I = \frac{M_1 - M_0}{P_0 - P_1} \tag{4.2}$$

$$P = (1 - r_S)(r_A(P_D + P_S) + (1 - r_A)P_S) + P_L \tag{4.3}$$

Figure 23 shows the indifference time in years between the eight and six layer energy costs for the mobile device scenario (r_A=0.9, r_S=0.92) [74]. This scenario approximates the usage of a mobile device, which is often sleeping but very active when awake. The indifference time indicates the interval after which the total energy (manufacturing and use phase) consumed by both the eight and six layer design will be equivalent. Prior to the indifference point, the six layer design is more efficient (since it always starts with a lower manufacturing cost), and after the indifference point the eight layer design has a lower total energy cost. DMA at 90nm, AES at 65nm, VGA at 90nm and 65nm, b17 at 65nm and 130nm, and b18 at 90nm and 130nm have an infinite indifference time, indicating the six layer design is lower in both manufacturing and use phase for those benchmarks at those nodes. For the remaining points, the optimal design depends on the expected lifetime of the circuit. For example, if the chosen lifetime is four years, the most sustainable solution is for DMA and VGA to be manufactured at six layers, and WB to be manufactured at eight layers, with the other four benchmarks' lowest energy cost solution depending on the chosen technology node.

Figure 24 displays the cloud server (r_A=0.3, r_s=0.05) [2] indifference points for the IWLS benchmarks studied. The benchmarks and technology nodes with infinite indifference times mentioned for the mobile scenario also have infinite indifference times for the cloud server scenario (and all possible active and sleep scenarios), since both the manufacturing and the use phase are always lower for six layers. For the finite cases, if a service lifetime of four years is chosen for the circuit, then VGA and b18 at 130nm have lower total energy costs

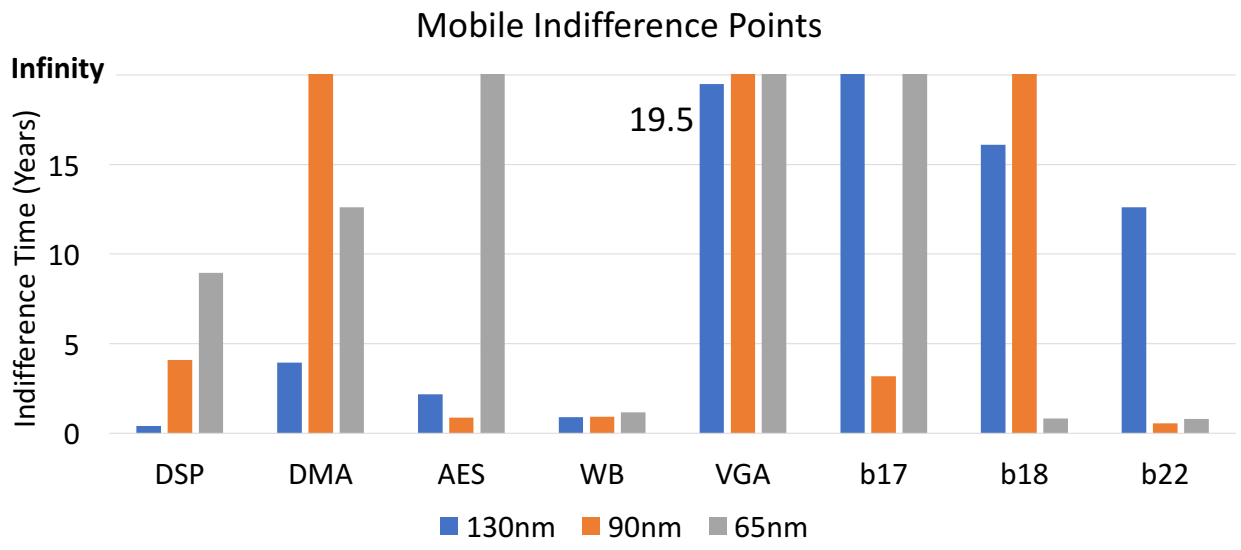


Figure 23: Mobile scenario eight vs. six layer indifference time (years).

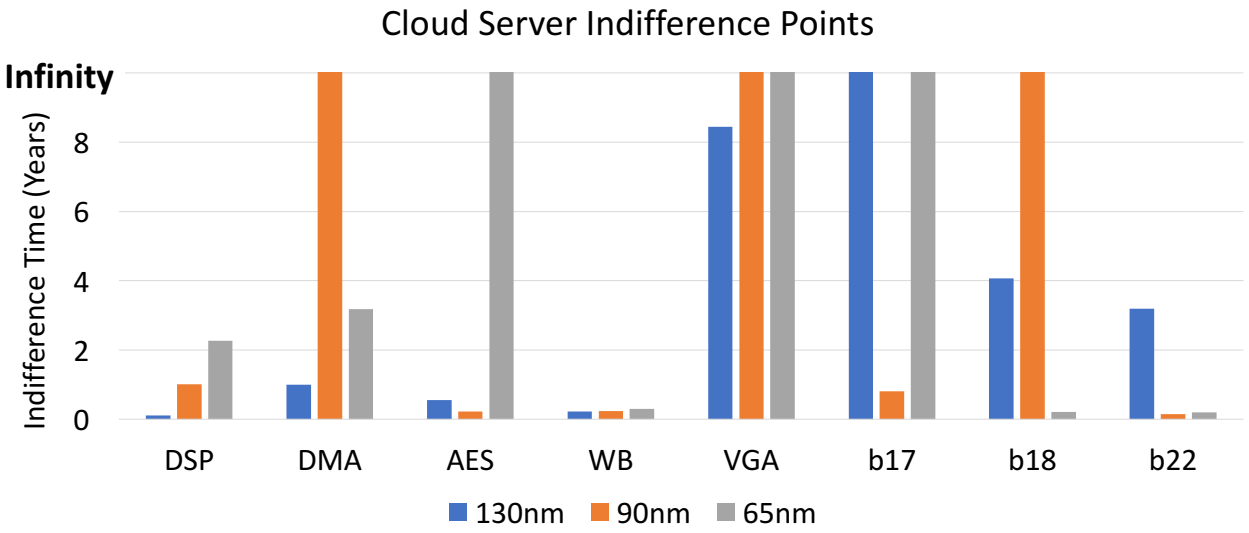


Figure 24: Cloud scenario eight vs. six layer indifference time (years).

for the six layer metal stacks. For the remainder of the cases for the cloud server scenario, if the lifetime is four or more years, eight layers is the better choice for sustainability.

Indifference points for the desktop (r_A=0.17, r_S=0.77) [2] and HPC (r_A=0.95, r_S=0.05) [73] scenarios are shown in Figures 25 and 26, respectively. The desktop system follows a similar trend as the cloud server, but the indifference times are on a much higher scale, making it

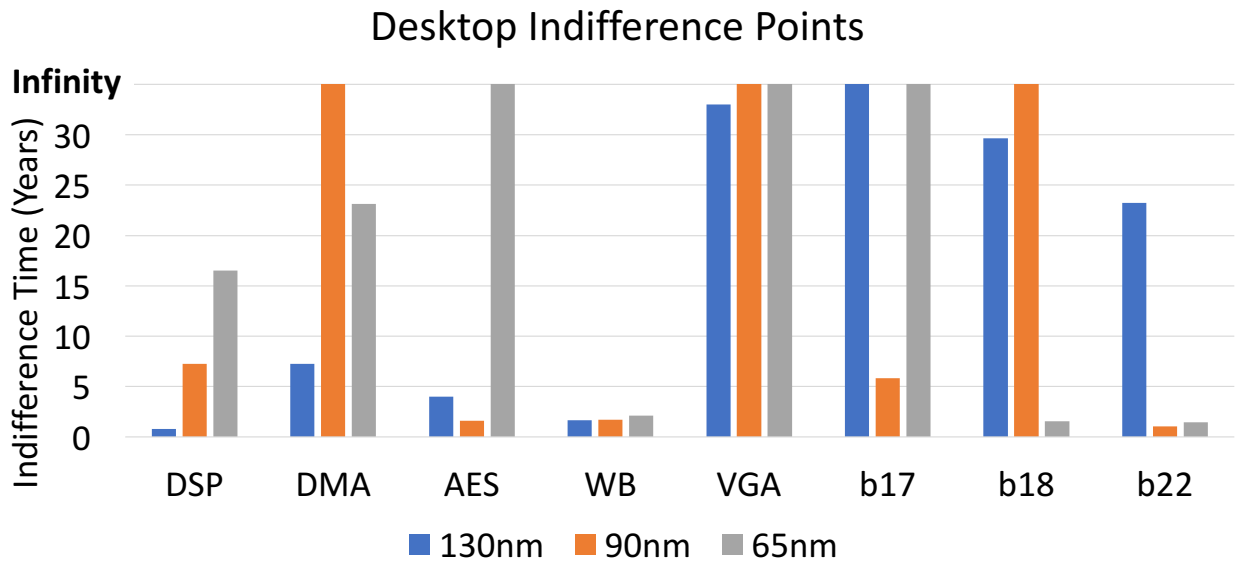


Figure 25: Desktop scenario eight vs. six layer indifference time (years).

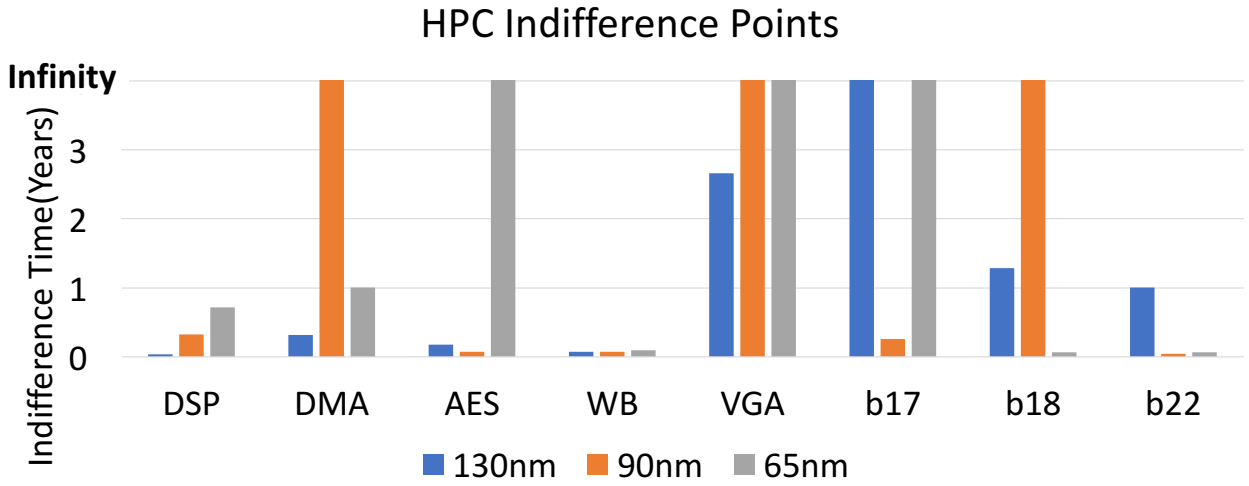


Figure 26: HPC scenario eight vs. six layer indifference time (years).

less attractive to consider using eight layer designs in a desktop setting. While the total energy consumed in the mobile scenario is often dominated by the manufacturing energy due to the high sleep ratio that minimizes the use phase's contribution, the HPC scenario is dominated by its use phase contribution. In all finite cases, the indifference times are

Table 9: Report on sustainability metrics, eight metal layers.

		DSP	DMA	aes	wb_con	vga_lcd	b17	b18	b22	Avg.
65nm	Smog (g NOx)	0.2386	0.1002	0.2170	0.2026	0.8846	0.2511	0.4085	0.1786	0.3101
	Acidification H^+ (millimoles)	14.0991	5.9217	12.8205	11.9726	52.2705	14.8363	24.1367	10.5537	18.3264
	Ecotoxicity (g 2,4-D)	1.1496	0.4828	1.0454	0.9762	4.2621	1.2097	1.9681	0.8605	1.4943
	Carcinogens (g C_6H_6)	$9.34{\cdot}10^{-5}$	$3.92{\cdot}10^{-5}$	$8.49{\cdot}10^{-5}$	$7.93{\cdot}10^{-5}$	$3.46{\cdot}10^{-4}$	$9.83{\cdot}10^{-5}$	$1.60{\cdot}10^{-4}$	$6.99{\cdot}10^{-5}$	$1.21{\cdot}10^{-4}$
	Total DALYs	$2.16{\cdot}10^{-6}$	$9.06{\cdot}10^{-7}$	$1.96{\cdot}10^{-6}$	$1.83{\cdot}10^{-6}$	$8.00{\cdot}10^{-6}$	$2.27{\cdot}10^{-6}$	$3.69{\cdot}10^{-6}$	$1.62{\cdot}10^{-6}$	$2.81{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	97.6090	40.9962	88.7571	82.8874	361.8730	102.7130	167.1001	73.0643	126.8750
	Eutrophication-air (g N)	0.0086	0.0036	0.0078	0.0073	0.0319	0.0091	0.0147	0.0064	0.0112
	Eutrophication-water (g N)	0.0134	0.0056	0.0122	0.0114	0.0499	0.0142	0.0230	0.0101	0.0175
90nm	Smog (g NOx)	0.2758	0.1131	0.1323	0.2243	0.7936	0.1601	0.8582	0.1881	0.3432
	Acidification H^+ (millimoles)	16.0900	6.5970	7.7166	13.0830	46.2910	9.3374	50.0644	10.9750	20.0193
	Ecotoxicity (g 2,4-D)	1.3178	0.5403	0.6320	1.0716	3.7915	0.7648	4.1005	0.8989	1.6397
	Carcinogens (g C_6H_6)	$1.15{\cdot}10^{-4}$	$4.70{\cdot}10^{-5}$	$5.50{\cdot}10^{-5}$	$9.32{\cdot}10^{-5}$	$3.30{\cdot}10^{-4}$	$6.65{\cdot}10^{-5}$	$3.57{\cdot}10^{-4}$	$7.82{\cdot}10^{-5}$	$1.43{\cdot}10^{-4}$
	Total DALYs	$2.45{\cdot}10^{-6}$	$1.01{\cdot}10^{-6}$	$1.18{\cdot}10^{-6}$	$1.99{\cdot}10^{-6}$	$7.05{\cdot}10^{-6}$	$1.42{\cdot}10^{-6}$	$7.63{\cdot}10^{-6}$	$1.67{\cdot}10^{-6}$	$3.05{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	113.3959	46.4930	54.3835	92.2043	326.2411	65.8061	352.8349	77.3476	141.0883
	Eutrophication-air (g N)	0.0098	0.0040	0.0047	0.0080	0.0283	0.0057	0.0306	0.0067	0.0122
	Eutrophication-water (g N)	0.0216	0.0089	0.0104	0.0176	0.0622	0.0125	0.0672	0.0147	0.0269
130nm	Smog (g NOx)	0.4142	0.3083	0.5044	0.4586	1.8975	0.2685	1.1581	0.2912	0.6626
	Acidification H^+ (millimoles)	24.2158	18.0266	29.4879	26.8116	110.9288	15.6986	67.7051	17.0267	38.7376
	Ecotoxicity (g 2,4-D)	2.0074	1.4943	2.4444	2.2225	9.1954	1.3013	5.6124	1.4114	3.2111
	Carcinogens (g C_6H_6)	$1.33{\cdot}10^{-4}$	$9.87{\cdot}10^{-5}$	$1.61{\cdot}10^{-4}$	$1.47{\cdot}10^{-4}$	$6.07{\cdot}10^{-4}$	$8.59{\cdot}10^{-5}$	$3.71{\cdot}10^{-4}$	$9.32{\cdot}10^{-5}$	$2.12{\cdot}10^{-4}$
	Total DALYs	$3.70{\cdot}10^{-6}$	$2.75{\cdot}10^{-6}$	$4.50{\cdot}10^{-6}$	$4.09{\cdot}10^{-6}$	$1.69{\cdot}10^{-5}$	$2.40{\cdot}10^{-6}$	$1.03{\cdot}10^{-5}$	$2.60{\cdot}10^{-6}$	$5.91{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	172.0596	128.0837	209.5192	190.5033	788.1782	111.5423	481.0624	120.9794	275.2410
	Eutrophication-air (g N)	0.0148	0.0111	0.0181	0.0164	0.0680	0.0096	0.0415	0.0104	0.0238
	Eutrophication-water (g N)	0.0440	0.0327	0.0535	0.0487	0.2014	0.0285	0.1229	0.0309	0.0703

Table 10: Report on sustainability metrics, six metal layers.

		DSP	DMA	aes	wb_con	vga_lcd	b17	b18	b22	Avg.
65nm	Smog (g NOx)	0.2049	0.0861	0.1863	0.1740	0.7598	0.2156	0.3508	0.1534	0.2664
	Acidification H^+ (millimoles)	12.1096	5.0861	11.0114	10.2832	44.8948	12.7428	20.7308	9.0645	15.7404
	Ecotoxicity (g 2,4-D)	0.9874	0.4147	0.8979	0.8385	3.6607	1.0390	1.6904	0.7391	1.2834
	Carcinogens (g C_6H_6)	$8.02{\cdot}10^{-5}$	$3.37{\cdot}10^{-5}$	$7.29{\cdot}10^{-5}$	$6.81{\cdot}10^{-5}$	$2.97{\cdot}10^{-4}$	$8.44{\cdot}10^{-5}$	$1.37{\cdot}10^{-4}$	$6.00{\cdot}10^{-5}$	$1.04{\cdot}10^{-4}$
	Total DALYs	$1.85{\cdot}10^{-6}$	$7.79{\cdot}10^{-7}$	$1.69{\cdot}10^{-6}$	$1.57{\cdot}10^{-6}$	$6.87{\cdot}10^{-6}$	$1.95{\cdot}10^{-6}$	$3.17{\cdot}10^{-6}$	$1.39{\cdot}10^{-6}$	$2.41{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	83.8357	35.2113	76.2329	71.1914	310.8101	88.2195	143.5211	62.7544	108.9721
	Eutrophication-air (g N)	0.0074	0.0031	0.0067	0.0063	0.0274	0.0078	0.0127	0.0055	0.0096
	Eutrophication-water (g N)	0.0116	0.0049	0.0105	0.0098	0.0428	0.0122	0.0198	0.0086	0.0150
90nm	Smog (g NOx)	0.2357	0.0966	0.1130	0.1917	0.6781	0.1368	0.7334	0.1608	0.2933
	Acidification H^+ (millimoles)	13.7496	5.6374	6.5942	11.1801	39.5577	7.9792	42.7823	9.3786	17.1074
	Ecotoxicity (g 2,4-D)	1.1262	0.4617	0.5401	0.9157	3.2400	0.6535	3.5041	0.7682	1.4012
	Carcinogens (g C_6H_6)	$9.79{\cdot}10^{-5}$	$4.02{\cdot}10^{-5}$	$4.70{\cdot}10^{-5}$	$7.96{\cdot}10^{-5}$	$2.82{\cdot}10^{-4}$	$5.68{\cdot}10^{-5}$	$3.05{\cdot}10^{-4}$	$6.68{\cdot}10^{-5}$	$1.22{\cdot}10^{-4}$
	Total DALYs	$2.10{\cdot}10^{-6}$	$8.59{\cdot}10^{-7}$	$1.00{\cdot}10^{-6}$	$1.70{\cdot}10^{-6}$	$6.03{\cdot}10^{-6}$	$1.22{\cdot}10^{-6}$	$6.52{\cdot}10^{-6}$	$1.43{\cdot}10^{-6}$	$2.61{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	96.9019	39.7304	46.4731	78.7927	278.7877	56.2343	301.5132	66.0970	120.5663
	Eutrophication-air (g N)	0.0084	0.0034	0.0040	0.0068	0.0242	0.0049	0.0262	0.0057	0.0105
	Eutrophication-water (g N)	0.0185	0.0076	0.0089	0.0150	0.0531	0.0107	0.0575	0.0126	0.0230
130nm	Smog (g NOx)	0.3854	0.2748	0.4503	0.3970	1.7575	0.2497	1.0829	0.2527	0.6063
	Acidification H^+ (millimoles)	22.5313	16.0644	26.3278	23.2106	102.7459	14.6003	63.3082	14.7720	35.4451
	Ecotoxicity (g 2,4-D)	1.8677	1.3317	2.1824	1.9240	8.5171	1.2103	5.2479	1.2245	2.9382
	Carcinogens (g C_6H_6)	$1.23{\cdot}10^{-4}$	$8.79{\cdot}10^{-5}$	$1.44{\cdot}10^{-4}$	$1.27{\cdot}10^{-4}$	$5.62{\cdot}10^{-4}$	$7.99{\cdot}10^{-5}$	$3.47{\cdot}10^{-4}$	$8.09{\cdot}10^{-5}$	$1.94{\cdot}10^{-4}$
	Total DALYs	$3.44{\cdot}10^{-6}$	$2.45{\cdot}10^{-6}$	$4.02{\cdot}10^{-6}$	$3.54{\cdot}10^{-6}$	$1.57{\cdot}10^{-5}$	$2.23{\cdot}10^{-6}$	$9.66{\cdot}10^{-6}$	$2.25{\cdot}10^{-6}$	$5.41{\cdot}10^{-6}$
	Noncancerous C_7H_7 (g)	160.0906	114.1418	187.0663	164.9173	730.0366	103.7389	449.8212	104.9592	251.8465
	Eutrophication-air (g N)	0.0138	0.0099	0.0161	0.0142	0.0630	0.0090	0.0388	0.0091	0.0217
	Eutrophication-water (g N)	0.0409	0.0292	0.0478	0.0421	0.1866	0.0265	0.1150	0.0268	0.0644

much smaller than for mobile, and only three circuits exceed one year. For these cases and three or more years of expected lifetime, the use phase advantage of eight layers exceeds the manufacturing advantage of six layers, and the eight layer stack should be used.

4.2.4 Additional Sustainability Reports

Based on the designs optimized to best tradeoff embodied and operational energy, our tool can report more detailed data on other sustainability metrics aside from energy and GWP. Tables 9 and 10 report smog in terms of Nitrogen Oxide (NOx), acidification in terms of Hydron concentration (H^+), ecotoxicity in terms of grams of dimethylamine (2,4-D), carcinogenic hydrocarbons in terms of grams of C_6H_6, disability adjusted life years (DALYs), non cancerous grams of C_7H_7, and eutrophication of air and water measured in grams of Nitrogen, for eight and six metal layer wafers, respectively. Contributions to smog damage the air quality. Acidification changes the pH of water making it damaging to wildlife. Dimethylamine (2,4-D) can also harm plants similar to a pesticide. Carcinogenic compounds have direct links to cancer. Eutrophication can encourage overgrowth of plants (including algae) causing Oxygen depletion in water and starving animals that depend on the Oxygen supply. DALYs quantifies the number of years lost due to ill-health, disability or early death due to environmental impacts.

Interestingly, scaling to newer technology nodes does provide a benefit to many of these factors, such as smog, DALYs and carcinogens, due to the area reduction of the die. This is not true for energy and GWP and other factors such as eutrophication. Thus, while holistic energy is a clearly important metric, the most sustainable design decision should also consider factors outside of energy. However, the trends consistently show that reducing the metal stack from eight to six layers provides a 10-12% reduction in these environmental impact categories, making it desirable to simplify the process if energy and performance allow.

4.3 Conclusions and Future Work

In this work, we proposed a scaled parameterized model for evaluating the environmental impacts of IC fabrication. We demonstrated a practical use of this model to estimate the manufacturing energy of six and eight layer metal stacks for 130nm, 90nm, and 65nm designs.

Using the output of the model and use phase energy estimations, we provided a holistic sustainability evaluation of the eight versus six metal layer decision. For any of the studied benchmarks, technology nodes, and usage scenarios, our analysis can be used to determine whether making a process change (*e.g.,* six or eight metal layers) is the more sustainable design for a desired service lifetime of the circuit.

This model can be applied to other design choices such as 3D CMOS and hybrid post-CMOS designs. We also hope to explore design kits for newer technology nodes as they become available to academic researchers. Moreover, conceptually, the model itself can be extended further; however, changes in the process for nodes $\leq$22nm become increasingly complex as lithography moves to multiple patterning and additional steps are required for more exotic transistors, increasingly low-κ dielectrics, etc. We will explore these directions in future work.

5.0 LARS Indifference Analysis

5.1 LARS Concept and Implementation

The indifference analysis from Eq. 6.1 where M_i is the embodied (*i.e.,* manufacturing) energy and P_i is the operational power of system i, was proposed and implemented in GreenChip [4]. It is sufficiently flexible for different holistic sustainability evaluations, when system reliability is constant between design choices. However, when evaluating an approach like MACE, different configurations will have an impact on lifetime due to memory wearout. If a system has lower use-phase energy and lower embodied energy, but it must be replaced every month, this replacement embodied energy should be taken into account when comparing it with another system. Thus, we propose an extension to indifference theory called Lifetime Amortized Replacement for Servers indifference analysis.

$$t_I = \frac{M_1 - M_0}{P_0 - P_1} \qquad (5.1)$$

$$t_{I_{LARS}} = \frac{M_1 - M_0}{(P_0 + A_0) - (P_1 + A_1)} \qquad (5.2)$$

The fundamental difference between LARS indifference analysis and prior sustainability indifference analysis [4, 5] is the inclusion of amortized embodied energy of replacements reflected in Eq. 5.2. In Eq. 6.1, the indifference point reports the time it takes for a system with lower operational power to save the equivalent energy of the larger embodied energy from more complicated manufacturing. However, when the mean-time-to-failure (MTTF) occurs for one system prior to the indifference time, traditional analysis is less meaningful. With LARS indifference analysis, we consider replacement embodied energy as a cost per time shown in Eq. 5.3, in a similar fashion to operational power. A_i is the embodied energy, M_i, divided by the lifetime, L_i, for a system i. L_i can be determined as the ratio of writes before failure WBF_i to the write velocity (writes per second) W_i of system i. The LARS comparison assumes two systems under comparison operate until failure, are replaced with

the same system, and the cycle continues indefinitely. Essentially, the LARS indifference time now considers replacement cycle along with embodied energy, operational energy, and usage scenario.

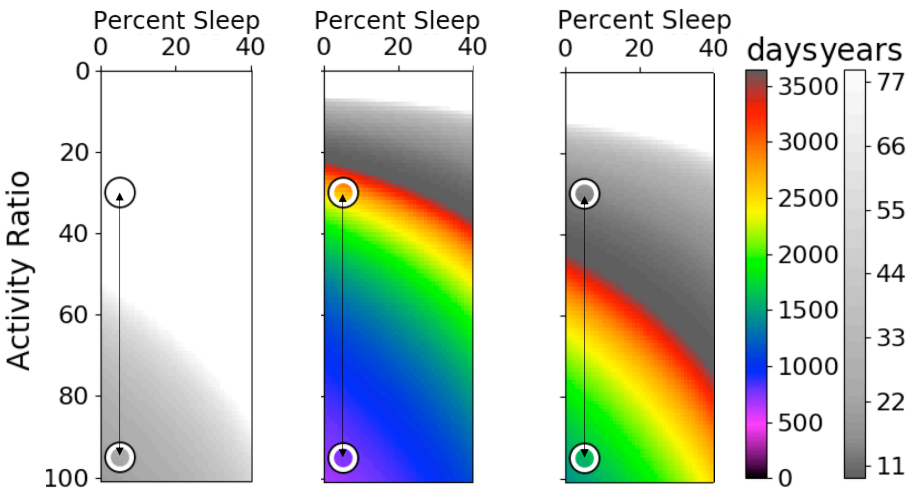
$$A_i = \frac{M_i}{L_i} = \frac{M_i \cdot W_i}{WBF_i} \tag{5.3}$$

5.2 LARS Case Study

Based on the results of the lifetime studies, a sustainability analysis was conducted using a modified version of the GreenChip [4] tool to include LARS indifference analysis. The observed lifetimes for each of the configurations were used alongside estimates of both embodied and operational energy consumption to evaluate the lifetime energy footprint.

Figures 27(a), 27(b), 27(c) show indifference point plots, calculated using Eq. 6.1 for (a) ECP6 versus MACE 32,6 (32 cosets, 6 pointers), (b) MACE WINDU (MW), which reverts to MACE 32,2 when compression is not possible, versus MACE 32,6 for a 1TB memory, and (c) the same MW versus MACE comparison for a 4TB memory. Note, the MACE configuration consistently has a larger area overhead, *i.e.,* embodied energy, with a lower operational energy. The figures highlight a range of usage scenarios for blade servers, from from 95% uptime and low utilization (underloaded cloud) to similar uptime and high utilization (heavily loaded cloud or supercomputer). Figure 27(a) indicates that MACE requires $\geq$29 years to recoup the additional embodied costs from the area overheads. Similarly, Figures 27(b) and 27(c) show MACE requires 2.1-7.3 years and 4.3-17.5 years, reflecting highest to lowest activity, for 1TB and 4TB memories, respectively.

Recalling from Section 5.2, that Eq. 6.1 does not accurately reflect indifference when reliability and replacement cycle are a function of the system comparison. Figure 28 shows the same comparison as Figure 27, except using Eq. 5.2 which takes into account amortized embodied energy from replacements (Eq. 5.3). The biggest change is in Figure 28(a), where indifference time of 29 years drops to $<$5 months, and low utilization approaching



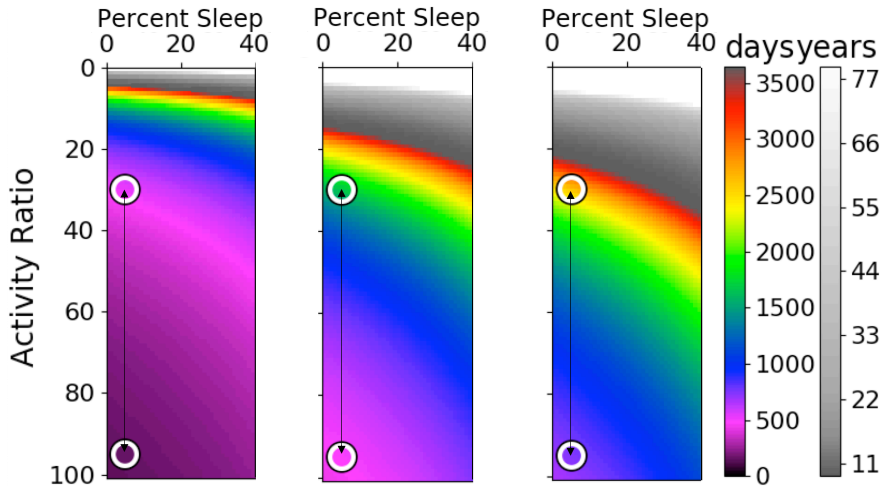
(a) ECP v MACE (b) MW v MACE (c) MW v MACE (4TB)

Figure 27: Indifference points (t_I) for the traditional GreenChip tool for ECP6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted.

∞ becomes <18 months. The MW versus MACE comparisons show similar adjustments where Figures 28(b) and 28(c) report 1.3-4.5 years and 2.0-7.1 years, respectively, reflecting approximately 60% and 2× lower indifference time for 1TB and 4TB memories, respectively. Clearly, from Figure 28, MACE is the considerably more, sustainable choice than ECP, but the choice of MACE WINDU versus MACE is more dependent on system configurations. MACE WINDU is more sustainable for moderate to large memories with MACE being more attractive for very large memories. This suggests a tradeoff between total capacity, effective capacity (total capacity less correction overhead), and activity ratio of a server.

5.3 Conclusion

The LARS indifference analysis extends the indifference analysis of GreenChip to include the memory lifetime. This permits a holistic study of the tradespace between activity, capacity, and correction overhead in the context of nonvolatile memories. For PCM memory



(a) ECP vs MACE (b) MW v MACE (c) MW v MACE (4TB)

Figure 28: LARS indifference points (t_I) for the GreenChip tool for ECP6, MACE 32,6, and MACE WINDU (MACE 32,2) for 1TB PCM, except where noted.

with AES-XTS encryption, the LARS indifference analysis demonstrates that the endurance and energy benefits of MACE and MACE-WINDU are realized on a relatively short time scale. The system designer can apply LARS analysis to explore the tradespace between the techniques, to better meet their capacity and activity needs and still realize an energy improvement within the system lifetime.

6.0 SFaultMap

6.1 SFaultMap Design

The central organizational concept of SFaultMap is that the fault data per-row is organized into row-indexed faultmap segments (*row-segments*), which pack fault data from multiple adjacent rows together. The encoding for an individual 512-bit memory row (covering between zero and four faults) is shown in Figure 29. The first bit represents zero ('0') or nonzero ('1') faults. If '0', the row is fault free and the row encoding is complete. If '1', two bits record the number of faults (one to four). The same number of fault locations within the row are stored with 9-bit pointers. In the case that four pointers were specified, the final pointer is followed by another row entry that continues the current row entry, representing an additional zero to four faults. Stacking entries in this manner allows an arbitrary number of pointers to be encoded in a row. Fault entries for adjacent rows are stored in a single row-segment (e.g., a 512-bit row) until the next row entry cannot fit.

Each row-segment stores its first row index (or "row-segment start" in Figure 30) to identify the appropriate row-segment during a memory access. To access the fault map, highest row-segment start that is less than or equal to the memory address determined. Then the row-segment is read and parsed linearly to find the row of interest, returning the fault information for that memory row.

An example of an implemented fault map for 8-bit rows (3-bit pointers) is shown in Figure30. During an access to address 2, the row-segment start of 0 is selected as highest row-segment start of ≤ 2. Rows 0 and 1 are parsed in succession, each showing a '0' leading bit and containing no faults. Row 2 has a leading '1' followed by "00" indicating one fault, with a pointer to bit 5 ("101"). This fault information is returned completing the search. A second example would be an access to row 12, which identifies the row-segment start point of 11. Row 11 has a leading bit of '1' indicating faults, followed by "00" indicating one fault. Since we are not concerned with row 11, the next three bits are ignored to move to row 12. The leading '0' returns that row 12 is fault free.

Faults	Encoding Bits				9 bits	9 bits	9 bits	9 bits	
0	**1**	**0**							
1	**12**	**1**	**0**	**0**	**Ptr1**				
2	**21**	**1**	**0**	**1**	**Ptr1**	**Ptr2**			
3	**30**	**1**	**1**	**0**	**Ptr1**	**Ptr2**	**Ptr3**		
4	**40**	**1**	**1**	**1**	**Ptr1**	**Ptr2**	**Ptr3**	**Ptr4**	**0**

Figure 29: Encoding strategy for data-agnostic bit-level fault map.

Row-Segment-Start	Fault Map (Each row in table is a Row-Segment)									
0	0	0	1	0	0	1	0	1	0	
4	1	0	1	1	0	1	0	0	1	0
6	0	0	0	0	0					
11	1	0	0	0	0	0	0	0		
14	1	0	1	1	0	1	0	1	1	

Figure 30: Example of the data-agnostic fault map, with a row-segment size of 10 bits and a row size of 8 bits (3 bits per pointer). Even rows are orange.

6.1.1 Performance Improvement 1: Offset Segment Lookup

Conducting a binary search to determine the row-segment start provides a relatively efficient access rate into the relevant row-segment requiring $2\log_2 S$ tests, where S is the number of row-segments. However, maintaining an offset table for quickly indexed search can further reduce the access latency as follows. First, the average number of rows per row-segment (R_s) is calculated. Then the *bit offset*, B, is calculated where $B = \lfloor log_2 R_s \rfloor$. The table is maintained for every 2^B rows.

During an access, the lower B bits are truncated from the address and the table is accessed to return a row-segment in the neighborhood of the address. The offset table typically reduces the $2\log_2 S$ search to between 1-3 row-segment checks.

6.1.2 Performance Improvement 2: Zero-fault Bit

While the representation of relatively high error rates, such as 10^{-3} and 10^{-4} are very compact with the proposed fault map, this density causes significant lookup delays for many rows. For these error rates, most rows in the memory are still fault free, which makes their lookup wasteful. Thus, we propose the addition of a single "fault free indicator" auxiliary bit to each memory row. Thus, for low fault rates the only entries which must traverse the fault map are the few rows which have faults. Note, SFaultMap must still store '0's for fault-free rows to ensure correct indexing. This approach adds a memory storage overhead of only 0.19% (1/512) while enabling considerable performance improvement for the fault map. We refer to the version of SFaultMap that employs these extensions as SFaultMap+, which we will discuss further in Section 10.2.

6.1.3 Fault Map Extensions and Discussion

SFaultMap enumerates the locations of potential faults. This allows error mitigation schemes to make decisions about how to store their data to avoid these faults. For example, the data may be able to be encoded to avoid bad data patterns which coincide with faulty cells such as bad patterns for write disturbance in PCM [31] or bitline crosstalk in DRAM [12].

Furthermore, knowing the location and number of faulty cells could allow compressed data using a lightweight compression technique [81] to be stored in order to skip faulty cells.

However, SFaultMap can be extended to serve as the fault tolerance approach directly. Each pointer can be extended with a replacement bit value (e.g., a pointer for a 512-bit block can be extended to ten bits). During a write to the location, the fault map is updated with data corresponding to the bits that would be stored in the faulty cells. If the fault map is initialized by saving space for these data bits, the fault map can be updated during each write without causing a change in structure.

In addition, SFaultMap is a static map that assumes faults are discoverable at test time and do not change (such as for cells vulnerable to wordline crosstalk [14] and bitline crosstalk [17]). However, for some forms of faults, such as endurance faults in technologies like PCM, the map must be adjusted over time as new faults are added. The compact nature of the fault map would require an entire rewriting process whenever an additional fault occurs.

6.2 Experimental Setup

To model weak cells of the memory, 4GB maps of weak cells were created using a Bayesian distribution to mimic the impact of process variation and include spatial correlation of faults [21, 82]. In particular, the model described in [21] was used to generate weak cell maps for a 4GB DRAM. These maps are appropriate to model weak cells for wordline or bitline crosstalk as well as low retention time. Similar to ECP, it is assumed ECC-1 would be used to protect against unrelated transient faults such as single event upsets.

A custom fault map simulator, based on the SNIPER full system simulator [65], was created to implement recovery schemes that utilize these weak cell maps. From the simulator, the number of bits required to represent the weak cells in the fault maps (static form), the average number of rows represented per row-segment, and other statistics for each configuration of SFaultMap can be generated. The simulator was also extended to allow inclusion of the binary/offset-based row-segment lookup and the flag bit indicating whether the row is

fault free (SFaultMap+). Additionally, the row-segment size can be varied in the simulator to study the area and performance tradeoffs for a given fault map.

To study the detailed performance and power implications of the fault map, a design for the SFaultMap decoder was created in VHDL, and synthesized using Synopsys Design Compiler targeting a 45nm FreePDK [83]. Because the power and delay of the circuit per access depends on the row address and the average number of addresses to traverse in the row-segment, the reported energy and delay correspond to traversing half of the average rows per row-segment. The access delay from the hardware implementation was combined with the row access time and decoding time (unless the fault-free bit was used and set) for the fault map entry once located from the row-segment. The simulator was tested on workloads from the SPEC CPU benchmarks [84]. The detailed parameters of the system simulated are shown in Table 11.

6.3 Evaluation

To demonstrate the effectiveness of the proposed sustainable fault-map we examine the area, power, and performance impacts and tradeoffs of the different fault map configurations in comparison to ECP and ArchShield. Based on these results we present holistic energy analyses of the tradeoffs in order to determine the configurations which provide the most sustainable solution for different workloads and usage lifetimes.

Table 11: Architecture parameters.

CPU	**Cache**
4 out-of-order cores	Private L1 32KB Inst, 32KB Data
4 issue width, 4GHz clk	Private L2 2MB/Core
45 nm Technology	Associativity: 8 (L1 data and L2 caches)
1GHz Frequency	Block Size: 64B

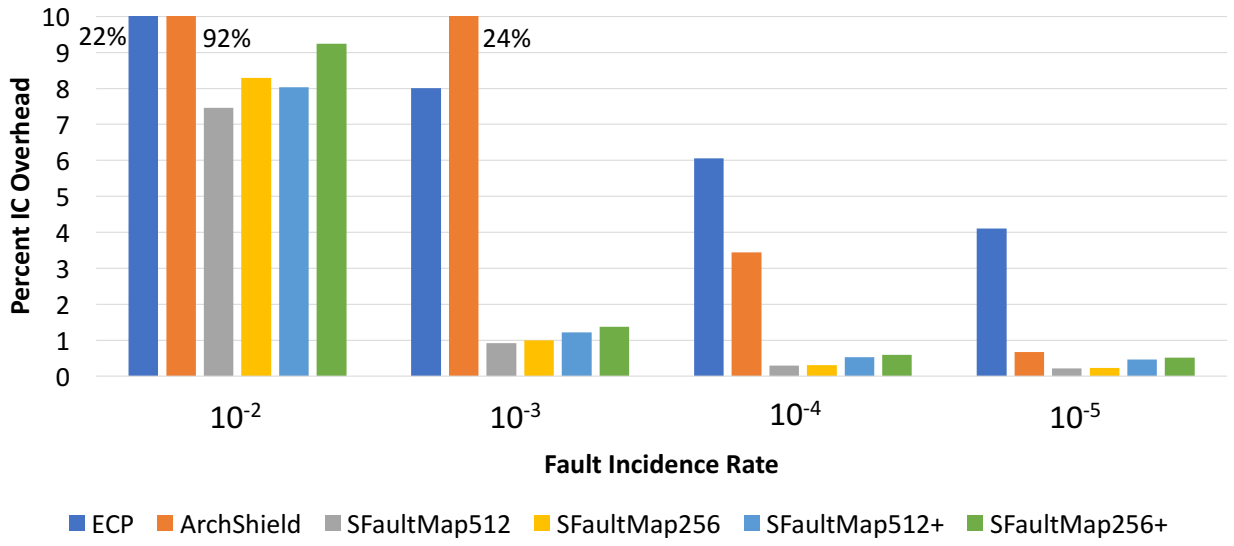


Figure 31: IC (Area) Overhead for ECP, ArchShield, and the Fault Map at different initial weak cell rates.

6.3.1 Area Overheads and Embodied Energy

Figure 31 displays the IC overhead required for SFaultMap, ArchShield, and ECP in order to guarantee correct operation for different fault frequencies. In order to determine the number of ECP pointers required for each fault frequency, a probabilistic weak cell map (see Section 6.2) was generated for each of several incidence rates. Due to the probabilistic nature of the weak cell map generation and in order to not overly penalize ECP, a 25% larger fault map (5G) was generated to select a 4G weak cell map with best case fault distribution for ECP. To maintain spatial correlation the weak cell map is generated in 256M portions. This requires that a total of 20 portions are generated and the worst four are pruned. Based on these tests, the minimum required ECP pointers were 11, 4, 3, and 2 pointers for 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5} incidence fault rates, respectively.

To model the overhead for ArchShield, two configuration bits were added per row for the memory to indicate if the row contains zero, one, or two or more faults. Additional redundant rows (spare rows) were allocated for faulty rows. We did not penalize ArchShield

for the overhead of SECDED ECC in our experiments as it is often available by default in many systems to address transient errors.

For SFaultMap we specify a 512- or 256-bit row-segment size (SFaultMap512 and SFaultMap256, respectively). Recall that the row-segment size is the number of bits that sequentially packs fault descriptions for a variable number of memory rows (see Section 6.1). A smaller row-segment size requires a higher IC area overhead than a larger size due to its higher *wasted bit ratio* from packing faults into segments (analogous to padding in memory). Additionally, the area overhead required by SFaultMap scales proportionally with the number of faults. For example, SFaultMap512 ranges from 0.214% to 7.46% for incidence fault rates between 10^{-5} and 10^{-2}, while SFaultMap256 ranges from 0.227% to 8.29% over the same fault incidence interval.

Recalling that the "improved" versions of those two segment sizes include both the offset-segment lookup and a fault-free flag bit per row. The area investment required by SFaultmap+ ranges from 1.08–2.14× over SFaultMap512 and from 1.11–2.28× over SFaultMap256 for fault incidence rates from 10^{-2} to 10^{-5}) . Despite this additional area investment, the least area efficient SFaultMap (SFaultMap256+) still saves between 2.35–10.16× the area over the corresponding ECP protection for fault incidence rates of at 10^{-2} to 10^{-4}. Within ArchShield's intended fault-incidence range the area overhead of ArchShield is between 1.3× higher (corresponds to 10^{-5} fault rate) and 5.8× higher (corresponds to 10^{-4} fault rate) than the largest overhead SFaultMap+ schemes.

This area savings has significant implications for the indifference times of the fault map, which will be discussed in the next subsection.

6.3.2 Runtime Overheads and Operational Energy

Figure 32 displays the energy consumption of the decoding circuitry for both ECP and SFaultMap. Note that SFaultMap+ and SFaultMap have nearly indistinguishable energy consumption when the row-segment is inspected. The decoding implementation of ECP looks up and flips bits from its valid pointers in parallel, while by design each row-segment in SFaultMap must be searched linearly to find the desired row. An interesting trend is that

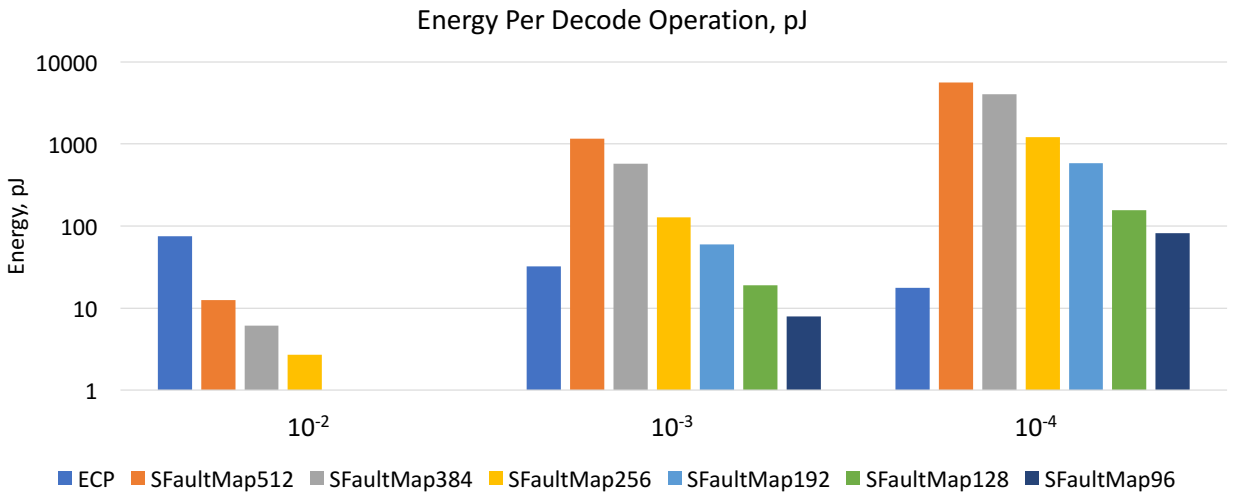


Figure 32: Energy consumption for decoding ECP and the improved fault map at different sizes and initial fault rates. 10^{-2} cannot have a fault map size of 192 or lower because certain rows require at least that many bits alone.

as the fault rate increases, the resulting ECP energy required increases, while the SFaultMap energy decreases. This is because as the SFaultMap becomes more dense, fewer traversal operations are required to find the average element within the row-segment.

When considering SFaultMap+ we recall that the fault-free flag from SFaultMap+ can prevent the row-segment inspection altogether. Specifically, the row-segment inspection can be bypassed for 8%, 77%, and 97% of accesses for fault incidence rates of 10^{-2}, 10^{-3}, and 10^{-4}, respectively. Further, for the fault map structures, the reduction in row-segment size causes a reduction in average access power, since the reduced amount of searching within a row-segment dominates the small increase in effort required to index the row-segment. This has a clear area/power tradeoff, which we discuss in the larger picture of holistic sustainability in the next subsection.

Figure 33 shows the relative impacts of instructions per cycle (IPC) for SFaultmap256 at different fault incidence rates. Certain SPEC CPU benchmarks, specifically calculix, gobmk, hmmer, namd, and sjeng are fairly memory performance invariant. In contrast, memory performance dependent benchmarks, such as bzip2, cactus, lbm, leslie3d, libquantum, mcf,

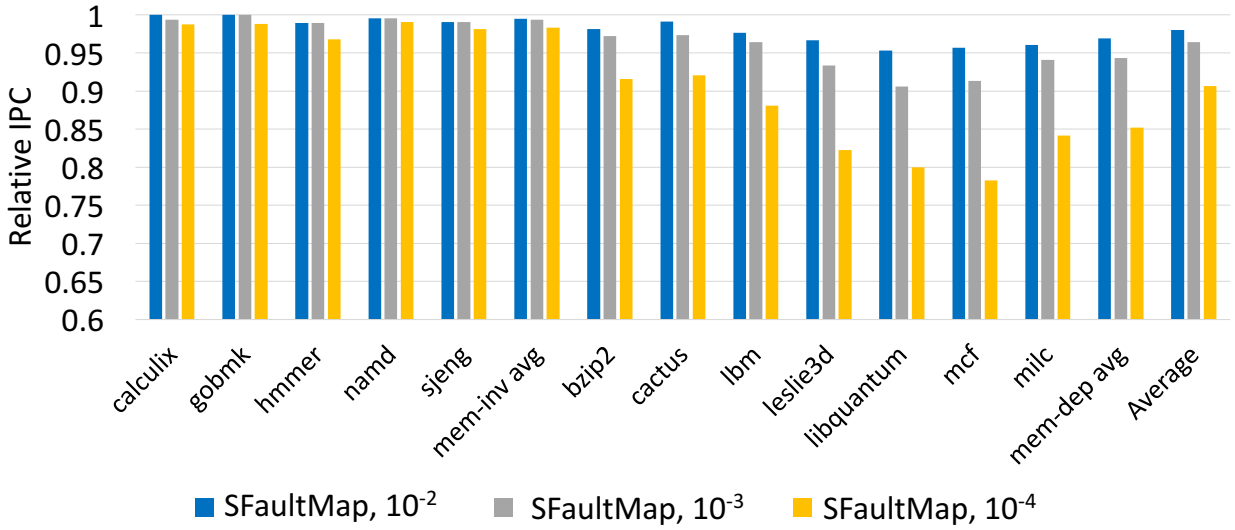


Figure 33: IPC impact of 256 bit block fault map designs across SPEC benchmarks normalized to a fault-free IPC at different fault incidence rates.

and milc see dramatic slowdowns. On average, the degradations for SFaultMap256 are 2%, 3.5%, and 9.3% for 10^{-2}, 10^{-3}, and 10^{-4} incident fault rates, respectively, but the dramatic slowdowns for a large segment of benchmarks (as much as 15%) is clearly problematic. Thus, moving forward we consider averages of memory performance dependent benchmarks from SPEC to highlight solutions to this performance bottleneck. The higher IPC for higher fault maps occurs because of the reduced search time within the row-segment for high fault rates, which comes at the cost of higher area (Figure 31).

Figure 34 shows the average IPC for memory dependent (mem-dep) and invariant groups (mem-inv) of SPEC benchmarks for both SFaultMap and SFaultMap+ at various fault incidence rates. For all fault rates, SFaultMap performs adequately for mem-inv benchmarks. For mem-dep benchmarks, particularly at the lower 10^{-4} fault rate, SFaultMap has a 15-25% IPC reduction depending on row-segment size. SFaultMap+ reduces this to a 5-10% degradation. Moreover, for higher fault rates, SFaultMap+ keeps IPC within 2-3% of a system without a fault map and comparable to ArchShield at lower fault rates.

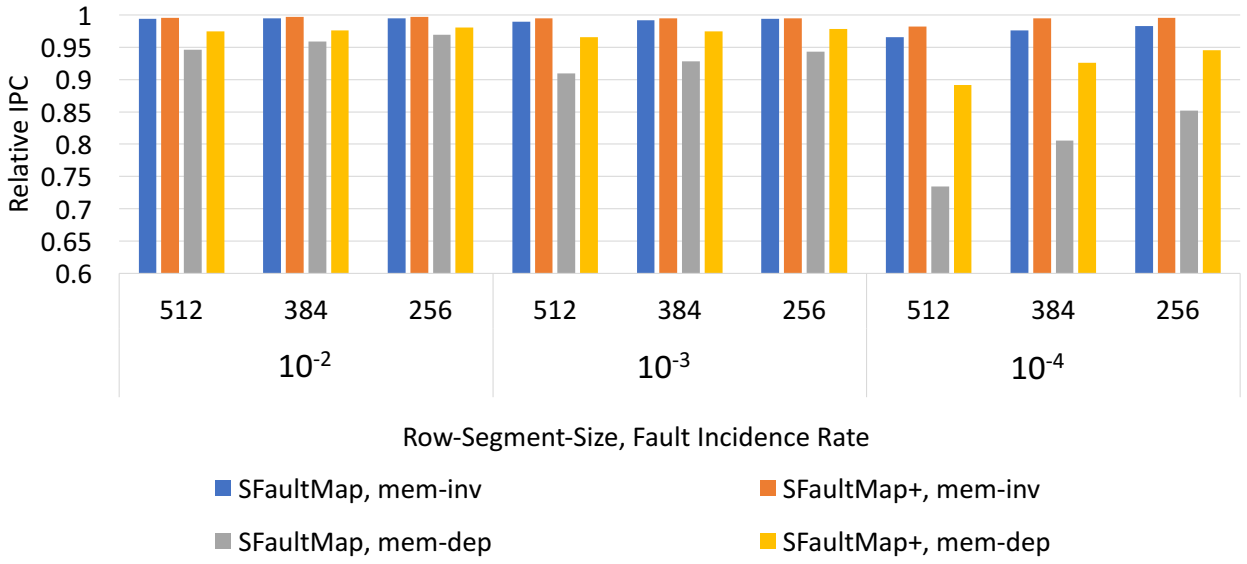


Figure 34: Average IPC across SPEC benchmarks, relative to a system without a fault map.

6.3.3 Holistic Energy Analysis

To evaluate sustainability, we use indifference analysis from GreenChip. The indifference time is the time when the total energy, including embodied and operational energy, of two different designs or architectures are equivalent according to Eq. 6.1[1]. M_x and P_x are the manufacturing energy and use-phase power of one of the two designs. We compute operational power from Eq. 6.2 where r_S is the sleep ratio, r_A is the active to idle ratio when awake, and P_D, P_S, and P_L are the dynamic, static, and sleep power, respectively, reported by McPAT[42]. We calculate indifference points for three system scenarios: a high-performance-computer (HPC) (r_A=0.95,r_S=0.05), a mobile computing system (r_A=0.9,r_S=0.92), and a cloud server (r_A=0.3,r_S=0.05) [5]. To account for the performance impact of difference design choices we adjust r_A in Eq. 6.2 in the system with a fault map to $r'_A = r_A(\frac{IPC_0}{IPC_1})$ [5].

[1]This equation can result in a negative result, indicating there is no indifference point. In this case the system with the lower embodied energy also has a lower operational energy and thus, should always be selected.

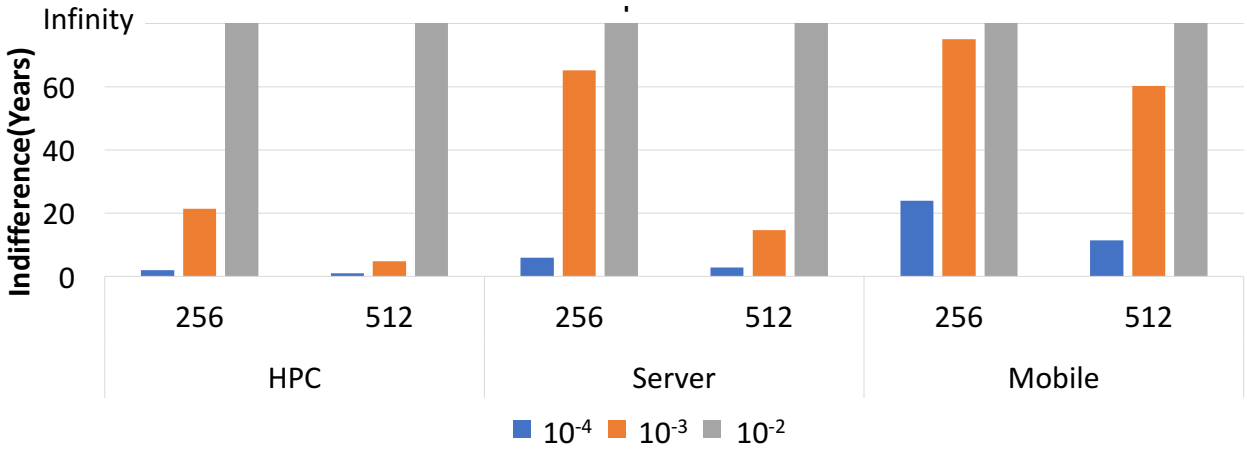


Figure 35: ECP vs. SFaultMap indifference times for different initial fault ratios, usage scenarios, and row-segment sizes.

$$t_I = \frac{M_1 - M_0}{P_0 - P_1} \tag{6.1}$$

$$P = (1 - r_S)(r_A(P_D + P_S) + (1 - r_A)P_S) + P_L \tag{6.2}$$

Figure 35 reports the indifference times of SFaultMap compared to ECP with the storage overheads sized to satisfy each fault incidence rate for each of three usage scenarios. Across all scenarios, the comparisons at a 10^{-2} incidence fault rate resulted in an infinite indifference time making SFaultMap always the most energy efficient solution. In fact, SFaultMap is always more energy efficient for ECP-4 at 10^{-2} or higher, not just ECP-11, even though ECP-$\leq$10 will not guarantee fault tolerance. At 10^{-3} the most sustainable strategy depends on scenario and row-segment size. If the product lifetime is estimated at ten years or less, than SFaultMap is more sustainable for all scenarios and segment sizes other than an HPC system with a 512-bit row-segment size, which has a indifference time of 4.8 years before ECP-4 becomes superior.

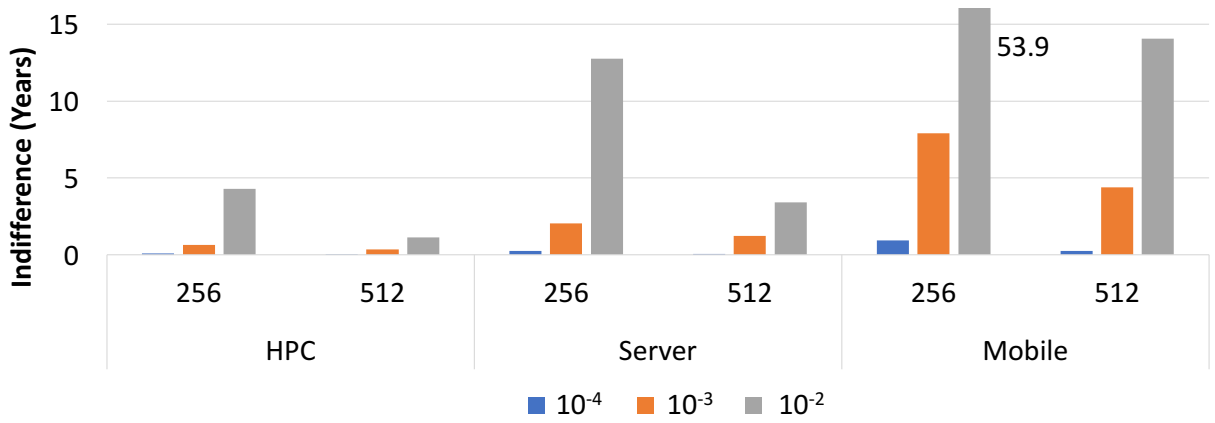


Figure 36: Indifference times SFaultMap and SFaultMap+. SFaultMap has the lower embodied energy.

Figure 36 shows the indifference times between SFaultMap and SFaultMap+. At 10^{-4}, all indifference times are less than two years before SFaultMap+'s operational energy advantage overcomes its embodied energy overhead. For 10^{-3}, the indifference times are mostly under 5 years, with the exception of the mobile scenario with the 256-bit row-segment size, where the use phase improvements might not be worth their incurred embodied energy overhead. At 10^{-2}, the HPC scenario benefits from the performance optimizations, but a mobile device does not, with the latter's indifference time for a 256-bit row-segment requiring over 50 years. In this particular scenario, SFaultMap would be the most sustainable solution.

6.4 Conclusion

SFaultMap provides a holistically energy-efficient solution to high fault rates in next generation memory devices. Like ECP, SFaultMap can be used for tolerance of faults caused by process variation such as bitline and wordline crosstalk in DRAM and endurance faults in PCM. However, it provides this fault tolerance with a minimal area footprint at the cost of some additional runtime energy consumption. Making small additions to the minimal area footprint in SFaultMap+ can greatly benefit performance, to the extent that it can be

the more sustainable solution within typical usage lifetimes. In other cases, e.g., high error rates, performance adjustments are insufficient to warrant the additional embodied energy.

7.0 FLOWER

7.1 The FLOWER Fault Map

FLOWER is a bit-level fault map which scales to extremely high fault rates to enable protection of bits which are either susceptible to faults (*i.e.,* weak) or have permanently failed. This allows FLOWER to be applied to many memory technologies and fault types. A particular strength of FLOWER is its relatively low area overhead compared to other techniques which protect against very high error rates due in part to its Bloom filter-based design. However, FLOWER provides several key innovations to the Bloom filter concept to make it suitable for high fault-rate memory instrumentation.

FLOWER is implemented using a Bloom filter and returns a *fault vector* that is the same width as the memory row. Each location with a fault is represented with a '1' and fault free cells with a '0.' In a conventional Bloom filter design a *unified array* is used to store each dimension of the filter. As such, a memory row addressed with N bits is mapped to an array of rows addressed by k bits, such that $k < N$. The Bloom filter uses a hash function to map each of the 2^N memory rows to the 2^k rows of the Bloom filter storage. Of course, as $k < N$, multiple memory rows will map to the same row in the Bloom filter storage. Thus, faults recorded in the Bloom filter will always be reported for the row in which they exist, but false positives will also be inadvertently reported for other rows that map to this location.

To minimize false positives with d-dimensional Bloom filters, d independent hash functions map each row to d different locations, effectively superimposing d Bloom filters into a unified array. For a fault to be reported, all hash functions, each of which returns a *partial* fault vector (PFV), must agree the fault is present in the row into which each hash is mapped. False positives are dramatically reduced, although not entirely eliminated, using this method.

A two-dimensional example of this mapping to faults is shown in Figure 37. Row 1024 has faulty bits at columns 1 and 5 and row 16880 has faulty bits at columns 1, 3, and 4. Row 1024 is translated to rows 417 and 780 by hash functions one and two, respectively.

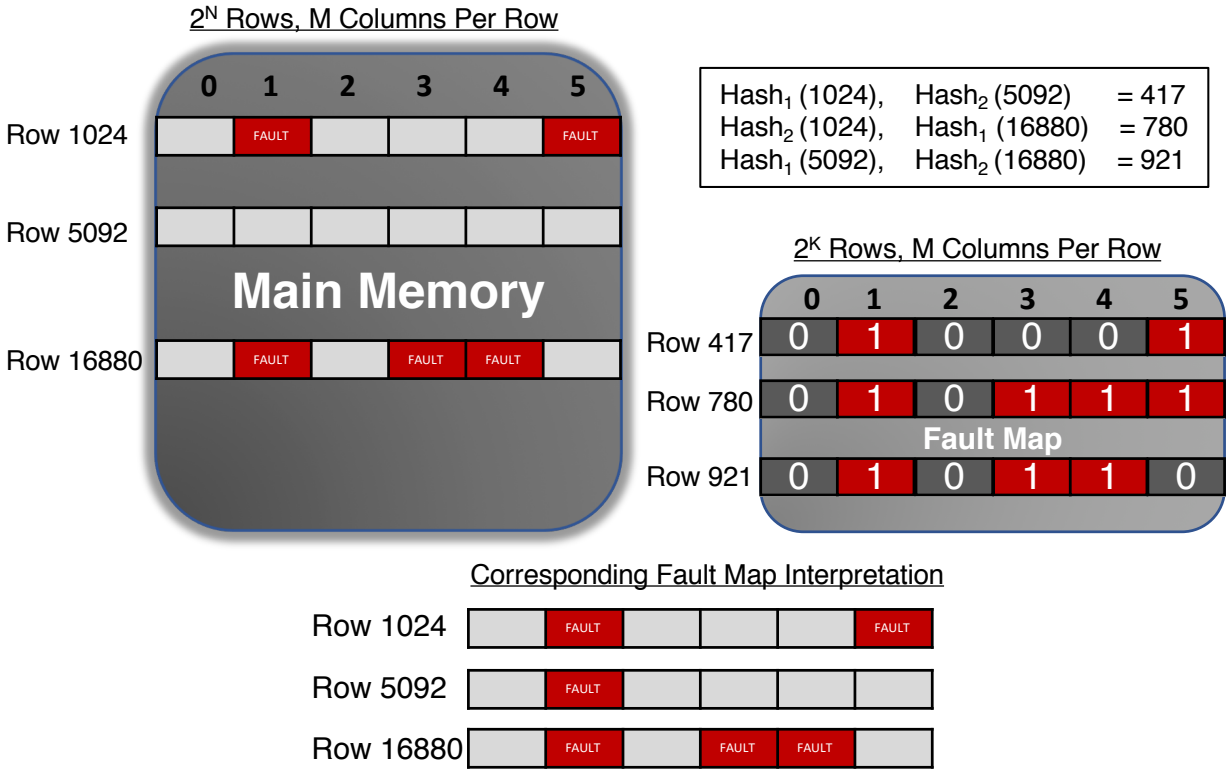


Figure 37: Two-Dimensional, unified-array FLOWER example. Red represents faulty cells. M is the length of each row, N are the bits in the address space for the memory, and k are the bits which result from the hash function

Thus, for these rows, bits 1 and 5 are set to '1.' Row 16880 maps to row 780 and 921 for hashes one and two, respectively. In these rows, bits 1, 3, and 4 are set to '1.' In general, for a d-dimensional Bloom filter, each faulty cell will result in d cells set to one in the Bloom filter, one for each hash.

Returning to the example, row 5092 in the memory has no faults, but maps to rows 417 and 921 in the Bloom filter. Because bit position 1 is set in both of these rows in the filter, when accessing this row in the fault map, it reports a false positive: that location 1 is faulty due to the collision of data from rows 1024 and 16880. For a d-dimensional fault map, fault vectors for a given row are constructed through the `AND` function of the PFVs accessed from each of the d hash functions in the filter. Note, the faults for rows 1024 and 16880 are still reported correctly. The prevalence of false positives directly correlates to the scalability and effectiveness of FLOWER; thus we will discuss techniques to minimize them in the next section followed by a discussion of the FLOWER architecture along with an in-memory processing extension in Section 7.1.2.

7.1.1 MinCI: A Tuned Hash for FLOWER

The application of a Bloom Filter for constructing a fault map that reaches high fault rates can create more dense structures than commonly employed in other applications. Thus, it is important to develop tuned hash functions that maximize the amount of information retained in the hashes to minimize collisions in the filter. Moreover, there are several opportunities to tune the hash function to support architecture implementation. First, dividing the d-dimensional Bloom filter storage into d smaller, independent subarrays, one for each of the d hash functions, has some advantages for access latency through parallelism. Second, employing simpler hashing functions can further streamline the implementation of FLOWER. In this section, we discuss these techniques to adjust our Bloom filter implementation to be more efficient.

In order to motivate our approach we simulated several fault maps with different storage capacities to study the use a unified array versus multiple arrays, one for each dimension.

Maps of faulty cells were created using normal distributions to mimic the impact of process variation and include spatial correlation of faults [85] (more details in Section 11.4).

First we consider the impact of using a Bloom filter using Murmur hashes with separate subarrays. Using a block-based partitioning where for a d-dimensional fault map, each dimension was allotted $1d$ the storage space as the unified array. The comparison of unified vs separate subarrays for the Murmur hash in a FLOWER fault map are show in the blue and orange bars, respectively, of Figure 39. The results show that multiple independent arrays performed at least as well as the unified array often reducing false positive rates. Moreover, this allows partial fault vectors for different hashes to be stored in separate sub-arrays in-memory which can support parallel combination (*e.g.,* using in-memory processing).

Murmur hashes, essentially pseudorandom hash functions, were designed to minimize collisions in the unified array Bloom filter. In separate arrays, the d hashes no longer have to consider collisions/overlap with one another to minimize false positives. They can instead focus on collectively retaining the most information possible. Moreover, while simple for software, most efficient Murmur hash implementations still require approximately 50 instructions, which is a significant complexity overhead for a hardware implementation. Thus, to simultaneously minimize the complexity of the hashes while maximizing the information retained by the filter we propose a *minimum cumulative intersection* (MinCI) of the address space.

Our MinCI hashes are designed offline for a particular memory size and fault map overhead and are implemented at runtime by simple address bit masking. We define MinCI for a given number of dimensions (d), address range (N), and hash size ($h = k - log_2 d$) to obey the following properties. Each bit in the address space is used a (or $a-1$) times. Each hash shares b (or $b-1$) bits with each other hash. Each hash shares c (or $c-1$) bits total, across all hashes. a, b, and c are minimized given the particular d, N, and h. These conditions for the minimum cumulative intersection can be intuitively explained using an overlap matrix, which shows how many times each hash overlaps with each other hash. An example of an overlap matrix is shown Figure 38. Each cell contains the $b_{i,j}$ value corresponding to the intersection of two hashes specified by the matrix row and column, while c refers to the sum of overlap for a particular hash along each column (or row). When the conditions on a, b,

and c are met, the sum of the elements of the overlap matrix will be minimized and equal to the value given by Eq. 7.1:

$$overlap_{minci} = 2 * \sum_{i=1}^{d-1} H(h * d - i * N) * (h * d - i * N) \tag{7.1}$$

where H refers to the Heaviside function [86]. Once MinCI masks are computed for a particular N, d, and h, they can be used for any system which uses the same N, d, and h.

A simple MinCI example is also shown in Figure 38. In this example, $d = 4$, $N = 9$, and $h = 4$. Each of the N bits is used by either one or two hashes, thus $a = 2$. Hash_1 shares two bits with Hash_2, and one bit with both Hash_3 and Hash_4, for a total of four bits shared. Hash_{2-4} have similar overlaps such that b=2 and c=4. The gray and yellow bars in Figure 39 show the false positive rate for the MinCI for both single (unified) and multiple arrays, which indicate the MinCI approach considerably reduces false positives over Murmur hashes. The multiple arrays version of MinCI degrades slightly over the unified array; however, the difference is small compared to the improvement of MinCI over Murmur. Thus, using the multiple array version of MinCI allows for several efficiency optimizations in the memory architecture while providing effective false positive minimization for FLOWER.

7.1.2 FLOWER Architecture

FLOWER uses the memory it is protecting to store the fault-map. An example of this is illustrated in Figure 40, where FLOWER is stored in one bank of a DRAM chip, protecting the other banks in the chip. Efficiently storing and accessing FLOWER directly within the memory also requires several adjustments to the memory system architecture. A traditional solution for implementing d dimensions requires d memory reads, each retrieving a PFV corresponding to each hash function. These PFVs can be combined with an `AND` operation directly at the memory controller. These fault-map memory accesses are a considerable overhead for each memory access; therefore we propose an in-memory approach to reduce this overhead where possible for both DRAM [87, 88, 89] and PCM [90]. In our experimental

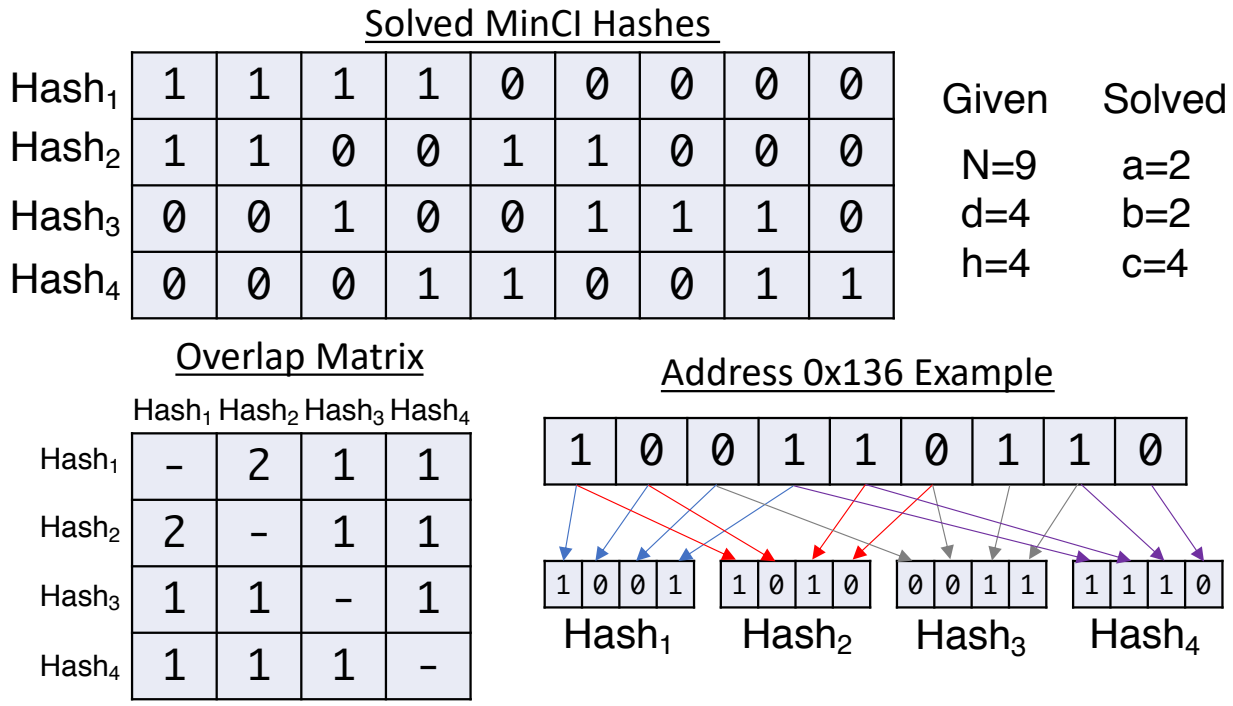


Figure 38: 4 masks for MinCi hash functions assuming $N = 9$ address bits, $d = 4$ dimensions, and $h = 4$ hash bits.

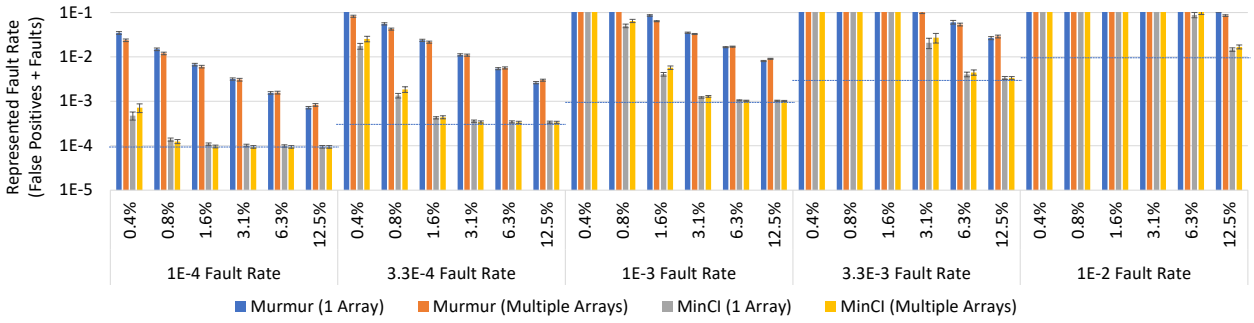


Figure 39: False positive rates for 4D fault maps with different storage overheads and initial fault rates.

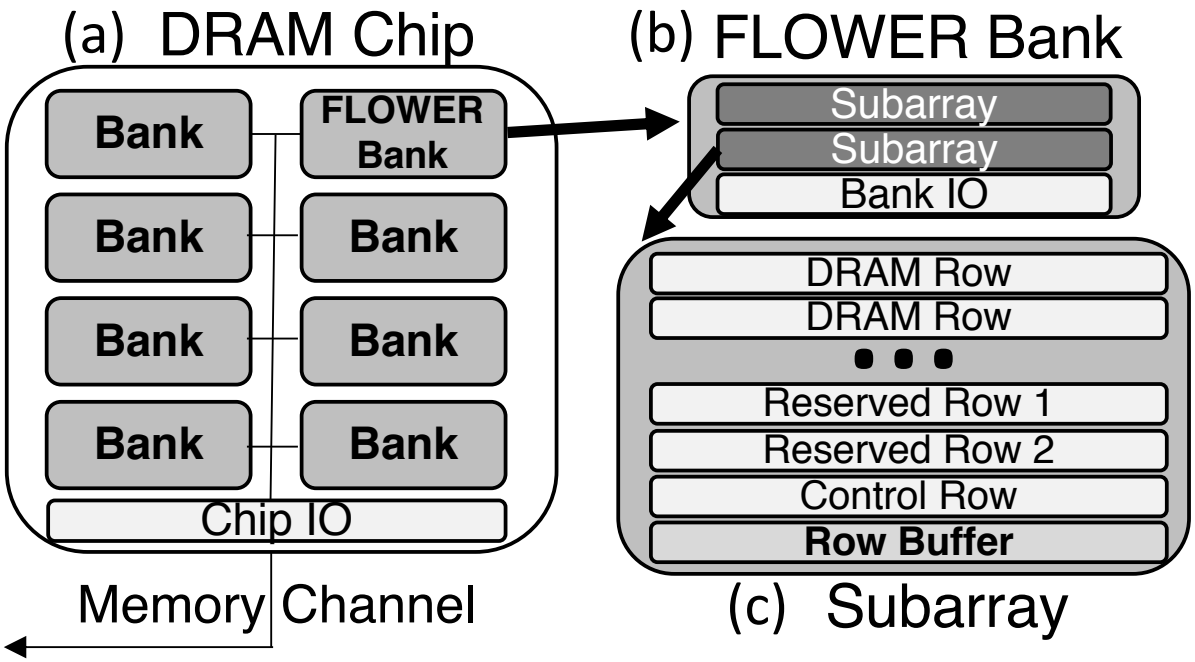


Figure 40: DRAM Organization for FLOWER using in-memory logical operations.

evaluation we compare the performance of both memory controller and in-memory based combination of FLOWER PFVs.

7.1.2.1 In-Memory FLOWER Access for DRAM To avoid the latency and energy of multiple DRAM accesses on the memory bus we leverage in-memory cloning and logical operations, proposed by [89]. A row can be cloned if the row is read into the row buffer and then a second row is activated as the row buffer will override what is stored in the row [87]. Logical operations can be conducted by activating two rows simultaneously, along with a third *control* row [88]. Along each bit line the voltages are combined such that if $\geq$2 of the three rows contains a V_{DD} the voltage will be driven to V_{DD} and if $\leq$1, the value is driven to V_{SS}.

Reading from the in-DRAM FLOWER fault-map is shown in Figure 41. First in steps 1 and 2, the two DRAM rows corresponding to PFVs from the first two hash functions, H_1, H_2–in the example rows `A1`, `A2`–are cloned to reserved rows `R1` and `R2`. In step 3 the control row `Ctrl` is initialized to '0's. In step 4, rows `R1`, `R2`, `Ctrl` are opened simultaneously to

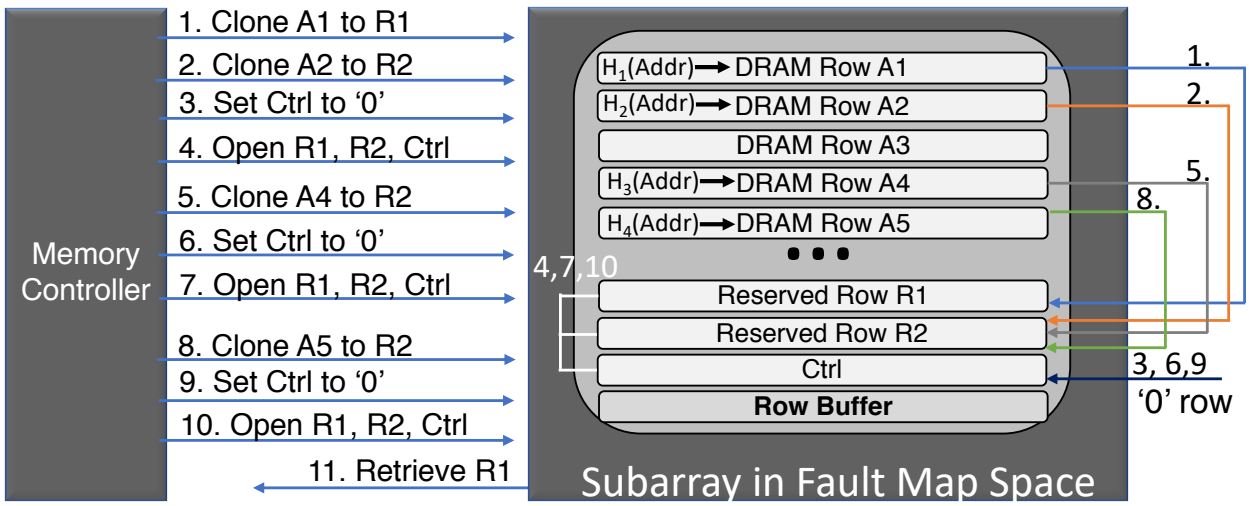


Figure 41: In-memory 4D FLOWER reading example.

conduct the `AND` operation. Note the result of the `AND` operation is left in all three rows and the row buffer [88]. Subsequent `AND` operations can be accomplished by cloning the next dimension's (*e.g.,* H_3) PFV, row `A4` in the example, into one of the two reserved rows (*e.g.,* `R2`) [Step 5], resetting the '0' row [Step 6], and completing the `AND` operation [Step 7]. In the example, after Step 10 the complete fault vector from a 4D map is available in the row buffer and can be returned to the memory controller [Step 11]. For d dimensions, this method requires d cloning, $d-1$ `AND`, and $d-1$ `zeroing` operations.

When applying MinCI in separate arrays to FLOWER with in-memory operations for the example in Figure 41, steps 1-4 for dimensions 1 and 2 can be completed in parallel to similar steps for dimensions 3 and 4 in independent subarrays. Then using similar steps to 5-7 the partial fault vector of dimensions 1 and 2 can be combined with the partial fault vector for dimensions 3 and 4 and steps 8-10 can be eliminated. For d dimensions this reduces the depth of computation from d-1 `AND` operations to $\log_2 d$ `AND` operations.

7.1.2.2 In-Memory FLOWER for PCM In-memory logical operations in PCM can be completed by simultaneously opening multiple rows, as in the DRAM case, using the technique developed in [90]. However, rather than using a control row, the result of the `AND`

function can be obtained by shifting the sensing threshold. For example, standard PCM sensing tests between resistance (Ω) Ω_{high} and Ω_{low}. By opening two rows simultaneously each cell resistance is combined along the bitline and can be tested in parallel. By adjusting the threshold to be halfway between $\Omega_{low+low}$ and $\Omega_{low+high}$ allows computation of the `AND` operation [90]. Theoretically, more than two `AND` operations can be completed in a single operation [90]. One key advantage of PCM in-memory logical operations over DRAM is that no initial copying is needed prior to activating two rows directly. Thus, in-PCM memory access of FLOWER functions in a similar fashion to the in-DRAM access from Figure 41 with some simplifications due to the non-destructive read operation. Consequently, the process of receiving a FLOWER result from PCM simply requires $d - 1$ comparisons. Similarly to DRAM, in-memory PCM bitwise operations can function across rows in different sub-arrays or banks on the same memory chip with a longer latency.

7.1.2.3 Updating the Fault Map While many faulty cells can be determined at the time of manufacturing and testing memory devices, endurance-based faults typically occur throughout the lifetime of the memory. PCM, Flash, and even DRAM can experience cells that fail due to use. FLOWER naturally supports adding new faults into the fault map. Once a new fault is detected, the previous fault vector (fv) for that row is updated with the new fault (fv'). The Bloom filter is updated by replacing each hash function's partial fault vector (pfv_i) with $pfv_i' = fv'$ `OR` pfv_i. Similarly to Sections 7.1.2.1 and 7.1.2.2, this can be accomplished using in-memory `OR` operations, possible in both DRAM [88] and PCM [90]. However, when using FLOWER with volatile main memory such as DRAM, upon system startup, the DRAM map must be populated from secondary, non-volatile storage. Thus, to protect against resident data loss, any new faults added to FLOWER have a write-through policy, and must immediately write updated FLOWER addresses to secondary storage. While this might seem expensive, for all relevant fault modes new failures occur very infrequently with respect to the lifetime of the device in question. Thus, fault map updates have only a minor, negligible impact on performance.

7.1.3 Implementation Details

The in-memory architecture, including combination of multiple partial fault vectors using in-memory logical operations is about 20% faster and upwards of 40× more energy efficient for DRAM [89], with potentially even higher speedups for PCM, compared to combining partial fault vectors at the memory controller. However, FLOWER access is still a significant overhead. Moreover, using a potentially faulty memory to store the fault map creates a challenge to ensure that the fault map itself does not accrue faults.

7.1.3.1 Improving Performance Whether performing the join of FLOWER dimensions in-memory or at the memory controller, the cost of accessing the fault map for every address would be prohibitive, and unreasonable at low error rates. Thus, FLOWER implementations for low-fault rates and for faults which accrue over time should have an *address filter* at the memory controller. This address filter is a simple, small, and fast first-level Bloom filter that tracks which rows have at least one fault [32]. If a row does not hit in the address filter, the row is fault free and accessing the fault map can be skipped. This filter allows the fault map to avoid significant performance degradation through a 10^{-4} fault incidence rate. The 10^{-4} fault rate is an important threshold, as it is the tolerable limit for many existing schemes. Beyond this limit, the FLOWER fault map is still capable of effectively representing fault rates of as much as 10^{-2} where the first level filter is less effective. Detailed performance results demonstrating the fault rate and performance tradeoff are presented in Section 7.3.2.

7.1.3.2 Storing FLOWER Reliably Storing FLOWER within the memory device it is protecting raises the question of reliable operation in a potentially faulty memory. However, for most cases, the special operation of the fault map makes it unlikely to experience these faults. Endurance-based faults and faults dependent on excessive write operations (such as wordline crosstalk in DRAM) can be safely ignored due to insufficient write activity in the fault map segment of the memory. Each row of the fault map would be written to *at most* M times (where M is the number of bits in the row) over the course of the memory's lifetime.

Moreover, each cell is only ever initialized to '0' and written one time to '1' to represent a faulty cell. Memories with high vulnerability to read disturbance can use probabilistic refresh [14] or counter-based trees [16] to protect FLOWER without additional overhead.

Furthermore, a FLOWER implementation that combines partial fault vectors at the memory controller can use ECC for protection. However, when electing to use in-memory logical operations for FLOWER, these operations may introduce transient faults and cannot be directly protected by ECC [89], because the ECC correction is not preserved over the `AND` operation required to combine the Bloom filter dimensions together. Our FLOWER-specific solution to this problem will be discussed in the context of FaME in Section 11.3.2. The next section describes how FLOWER can be used to support existing fault tolerance techniques for various types of technology specific faults (*e.g.,* endurance faults) or those due to deep technology scaling (*e.g.,* crosstalk).

7.2 FLOWER for Fault Tolerance

There are several fault tolerance techniques [12, 91, 92, 93, 94, 95, 11] that assume *a priori* knowledge of fault locations in order to provide fault tolerance. Most of these techniques assume a runtime method to detect the faults (such as read after write) [96] and possibly employ a fault cache to avoid adding these expensive tests [93, 17, 94, 95]. ArchShield uses a *word*-level fault map [11]. Unfortunately, for high fault rates, fault caches and word-level fault maps are ineffective. Additionally, read after write tests are often unreliable as faults can be probabilistic or instigated by unrelated accesses. Moreover, they can accelerate further cell failures in the case of endurance failures. Thus, FLOWER provides an alternative which we discuss in the context of two examples, bitline crosstalk in DRAM and stuck-at faults in PCM.

7.2.1 DRAM Bitline Crosstalk

Recall from the background, bitline crosstalk in DRAM occurs probabilistically when a "weak" cell and its surrounding cells are written with the same charge ("111" or "000"), which we refer to as "bad patterns." These weak cells are discoverable through system testing [17] and can be used to populate the FLOWER fault map. Periodic Flip Encoding (PFE) [12], is a technique to mitigate these bad patterns and significantly benefits from knowledge of weak cell locations. As shown in Figure 42, PFE either flips the first, second, or third bit (or no bits) of each 3-bit grouping to break up bad patterns. Red cells show a bad pattern overlapping with a weak cell (high probability for a fault) and gold show a bad pattern without a weak cell (no fault). PFE successfully avoids the fault in the (c) "10" encoding even though some bad patterns exist in the encoding, due to the knowledge of the location weak cells from the fault map. The authors of PFE specifically discuss the need for a bit-level fault map for fault-aware protection [12], which provides an $100,000\times$ improvement over fault-oblivious protection at a 10^{-4} fault incidence rate.

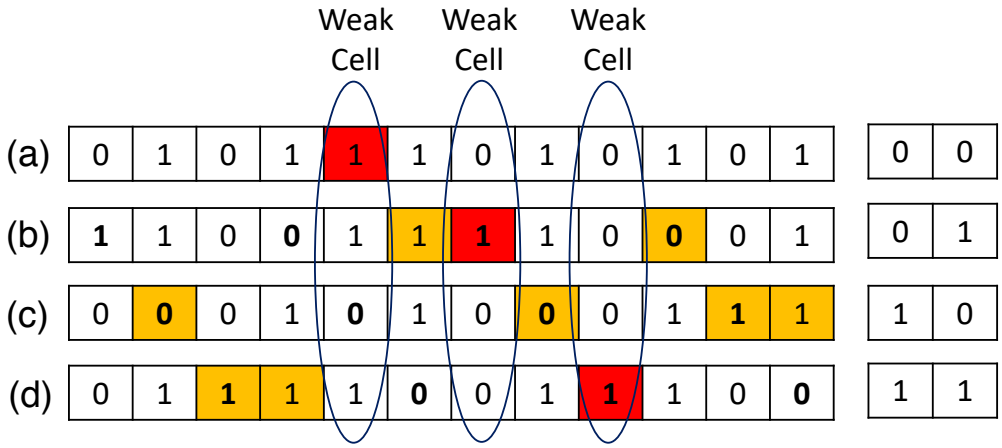


Figure 42: Encoding "0x5D5" into a 12-bit (11...0) with weak cells as positions 7, 5, and 3, using PFE.

7.2.2 PCM Stuck-at Faults

Yielding optimized dependability assurance (YODA) [91] is a technique to use stuck-at fault information to extend ECP from correcting p to $2p+1$ faults[1], where p is the number of pointers, by adding one bit for inversion. YODA matches data to faults to determine stuck-at right (SA-R) or wrong (SA-W) and points only to the SA-W values. YODA dynamically writes rows of '0's and '1's to discover the faults, significantly degrading lifetime (between 2×–3×).

FLOWER can keep separate maps for different errors, such as stuck-at '1' and stuck-at '0' faults, which can avoid lifetime reduction. However, in this instance, a single fault vector of generic stuck-at locations along with the data currently stored is a sufficient, lower overhead method to identify stuck-at '1' and stuck-at '0' faults. Nonetheless, the ability to segregate types of faults that must be handled differently, such as stuck-at faults and cells susceptible to write disturbance in PCM [97], is a useful capability of FLOWER.

7.3 Results

To evaluate the effectiveness of FLOWER and its capabilities in context of existingfault tolerance schemes we conducted simulations of error rate analysis, lifetime improvement in endurance limited memories (*e.g.,* PCM), and performance impact. The lifetime results will be left for the discussion of FaME (Chapter 11).

We first conducted a sensitivity study on the false positive rates when using one, two, four, and eight fault map dimensions for both Murmur and MinCI hash functions. In general we found that eight dimensions best minimized false positives while one and two dimensions performed significantly less well. However, four dimensions performed nearly as well as eight, while reducing the number of in-memory logical operations making four the best compromise. A comparison of Murmur and MinCI hashing for four dimensions is shown in Figure 39. The figure also shows at which area overhead point the adopted MinCI hash achieves fault

[1]This is classified as YODA_1 [91]

reporting rate with minimal false positives—*i.e.,* approaching the stated incidence fault rate indicated by a dashed line. Thus, unless otherwise specified, further results are reported for 4D MinCI hashes targeting separate arrays.

7.3.1 FLOWER for Enhanced Fault Tolerance

To begin, we analyze the improvement FLOWER-augmented fault tolerance provides to uncorrectable bit error rate (UBER). We focus on the two cases from Section 7.2, Bitline Crosstalk Correction in DRAM, and Stuck-at Fault Correction in PCM.

7.3.1.1 Bitline Crosstalk Correction (DRAM) Figure 43 provides an iso-area study of the PFE algorithm [12] with and without FLOWER for a faulty cell rate of 10^{-3}. It also provides a comparison with a perfect "ideal" fault map for the same instantiations of the FLOWER fault map. The PFE labels indicate the encoded block size. For example, PFE16 adds two auxiliary bits for each 16-bit block resulting in a 12.5% overhead, with PFE32, PFE64, and PFE128 requiring 6.25%, 3.13%, and 1.56% overhead, respectively. When utilizing FLOWER, half of the auxiliary bits are repurposed for storing a fault map of the same size. For example, the 4D fault map for PFE32 uses 6.25% overhead to encode faults and 6.25% to store a fault map, making it comparable to PFE16 without a fault map.

For 12.5% total area overhead, fault-aware PFE32 with FLOWER outperforms fault-oblivious PFE16 implementation by >125×. Comparing the achievable correction capability of FLOWER to an perfect (100% overhead) bit-level fault map, PFE32+FLOWER is within 50% of the ideal fault map with over 8x less area. At 6.25% total area overhead, PFE+FLOWER has a similar advantage over no fault map and is close to the ideal map. At the more coarse granularity dictated by 3.13% total overhead, PFE+FLOWER provides less than an order of magnitude advantage. In this case, the loss of fault tolerance seems to originate from significant reported false positives from FLOWER, as using an ideal fault map would improve this result by two orders of magnitude.

7.3.1.2 Stuck-at Fault Correction (PCM)

Figure 44 demonstrates the benefits of adding FLOWER to fault pointers for permanent faults for a 10^{-3} fault incidence rate. The areas are set approximately equal by replacing some pointers with two fault maps (SA-'1's and SA-'0's) and an additional "flip" auxiliary bit (see Section 7.2.2). For example, ECP4 dedicates four pointers and spares (8% overhead) but can only handle four total faults. YODA uses one, two, and three pointers with two 6.25%, 3.13%, and 1.56% overhead FLOWER maps, respectively, while having the ability to handle three, five, and seven reported faults, respectively. Total fault tolerance overheads are varied from 8% to 14%. At 8% area overhead, YODA-2 and YODA-3 with FLOWER are superior to ECP4, with the latter nearing two orders of magnitude improvement. By 10% area overhead, YODA-4 with FLOWER was sufficient to correct all possible faults in our SPEC [84] benchmark simulations, which without FLOWER requires ECP9. Thus, at this fault rate YODA (unlike PFE) is tolerant of FLOWER's false positives, making lower investments in fault map size and higher investments in auxiliary bits appropriate for this scenario.

7.3.2 Performance Impact (DRAM)

The fault tolerance enabled by FLOWER does come at a cost. In addition to the area overhead, there is a performance overhead from accessing the fault map. While somewhat mitigated by filtering rows at low fault rates and leveraging in-memory processing, FLOWER access latencies are non-negligible and block subsequent fault map accesses. To quantify the FLOWER performance impact we studied different overheads at different initial fault rates. Performance was simulated with the SNIPER full system simulator [65] using the architecture parameters from Table 12. Each memory access includes the delay of the first-level filter and upon a hit indicating at least one fault, the detailed fault-map is accessed as described in Section 7.1.2.1. In these results we assume a 4D FLOWER implementation. The results shown are for an average of 10 different fault maps[3] using MinCI with independent arrays, and are reported for an average nine SPEC CPU benchmarks. Results are conservatively reported for DRAM due to the longer relative latency of in-memory `AND` operations compared to PCM.

Figure 45 shows detailed performance results, in terms of instructions per cycle (IPC), for MinCI with multiple arrays at 6.25% fault map area overhead at different represented fault rates and compared to turning off the first-level fault filter, which has the same performance independent of error rate. The striped bars indicate the additional IPC achieved by using in-memory operations instead of bringing the four dimensions individually into the memory controller. Certain benchmarks, such as `h264ref` and `libquantum`, are not significantly impacted by the FLOWER detailed fault map access, while others, such as `perlbench`, `gcc`, and `gromacs`, can have over 25% reductions in performance at 10^{-2}. However, a general trend across all of the benchmarks is that $\leq 10^{-4}$ fault rates, the performance impact of FLOWER is within 2% of an unprotected system, while still benefiting from the fault map, due to the first-level filter. At 10^{-6}, this 2% degradation comes almost entirely from the access overhead of the address filter. Thus, FLOWER is competitive with previous techniques that attempt to reach 10^{-4} fault rates. Moreover, FLOWER can operate successfully at fault rates up to 10^{-2}, which for compute oriented benchmarks is still competitive with a fault free system. The significant performance impacts only occur for some applications, and only once the concentration of faults reaches the 10^{-4} threshold where other correction schemes no longer operate. Compared to the performance of ArchShield, which was reported as a 1% performance degradation and handled up to 10^{-4} incidence fault rates, FLOWER maintains within 2% performance degradation through 10^{-4}. In this way, FLOWER allows the system to perform correctly in a gracefully degraded mode after a 10^{-4} fault rate threshold.

Figure 46 shows the average normalized performance in IPC as fault rates increase from 10^{-6} to 10^{-2}. Note, smaller area budgets become ineffective when a sufficient fault rate is reached. Missing bars are excluded intentionally as they represent fault maps that report fault rates higher than 10^{-1}, thus providing insufficient accuracy. Depending on the desired level of fault tolerance, the figure can help guide what fault map area budget is sufficient. There is $<1\%$ performance degradation for $\leq 10^{-5}$ fault rate for each of the fault map storage overheads. At 10^{-4}, the viable limit of many other proposed correction schemes such as ArchShield [11], MinCI still maintains $<2\%$ performance overhead for fault map storage overheads of $\geq 1.56\%$. For fault rates above 10^{-4}, the performance degrades, as expected, due to increased hits in the first-level filter requiring more FLOWER accesses. At 10^{-3}, the

performance of MinCI with multiple arrays sees a 7% to 10% degradation, depending on the storage budget with the smallest fault map overheads now insufficient to provide sufficient fault resolution. By 10^{-2}, the first level filter no longer provides any benefit (all rows appear faulty) and at least a 6.25% area overhead is required.

7.4 Conclusion

We present FLOWER, a low-overhead bit-level fault map that facilitates unprecedented levels of fault detection while enabling previously infeasible correction schemes. Our novel hashing technique tuned for FLOWER, MinCI, allows improved false positives and performance over traditional (*e.g.,* disk-level) hashes such as Murmur. Our in-memory strategy for multi-dimensional fault maps allows faster and more efficient fault map accesses than traditional techniques conducted at the memory controller. FLOWER greatly enhances the correction capability of a leading DRAM bitline crosstalk correction scheme, PFE, providing an UBER improvement of over 125×. Similarly, FLOWER enables a pointer-based PCM endurance fault tolerance scheme, YODA, to operate with enhanced lifetimes. While fault map accesses do have performance overheads, in the typical range of operation for existing systems (fault rates reaching 10^{-4}), FLOWER has an average IPC degradation of $< 2\%$ with a 1.6% area overhead across SPEC benchmarks. In gracefully degraded DRAM operation, FLOWER can tolerate up to 10^{-2} fault rates with approximately 80% and 83% of original average performance, for memory controller and in-memory processing, respectively, at 6-12.5% area overheads.

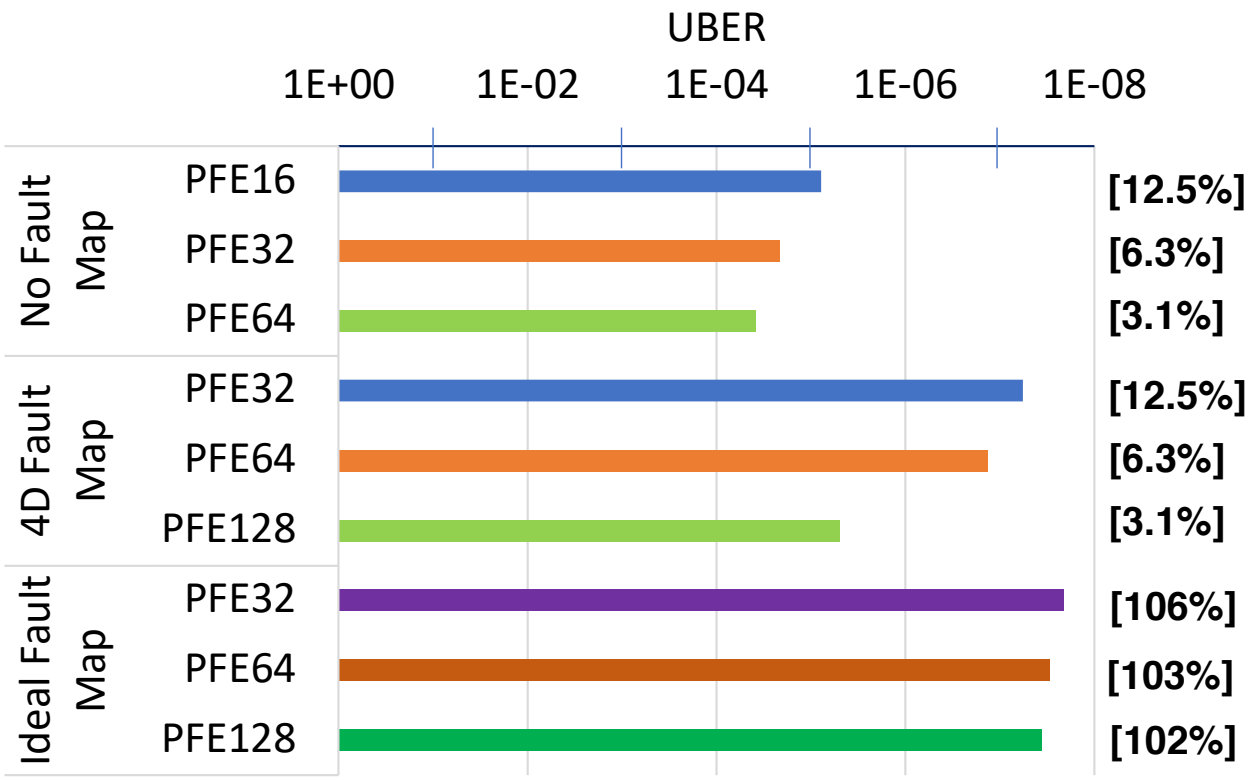


Figure 43: Iso-area PFE correction capability at 10^{-3} incidence weak-cell rate.Total overhead including fault map and encoding bits shown in "[]."

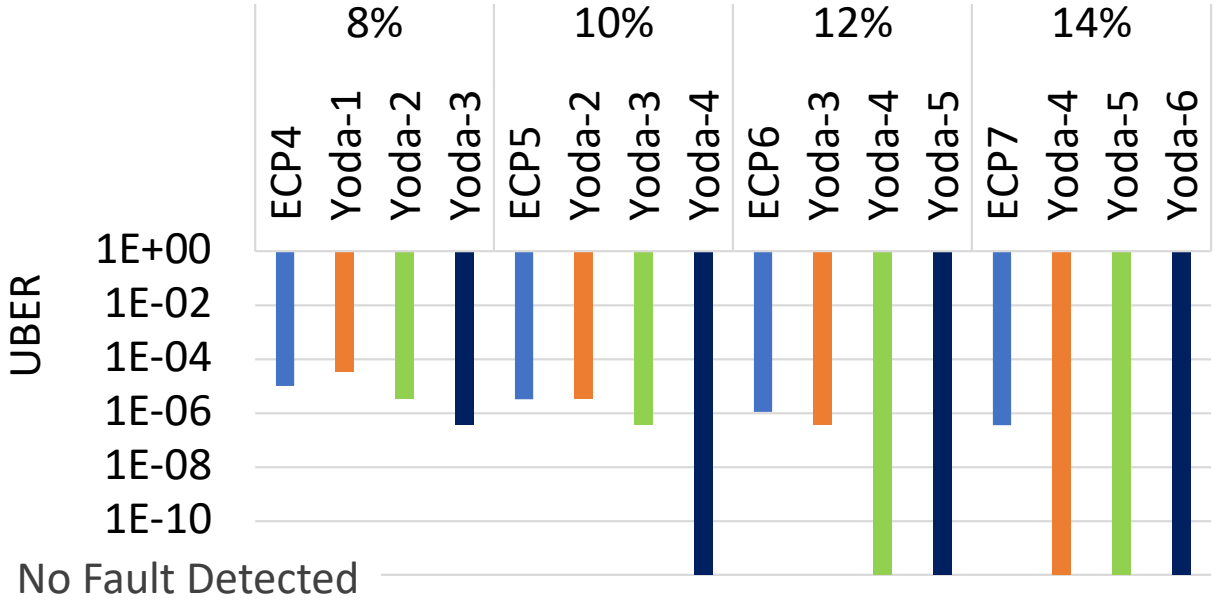


Figure 44: Iso-area analysis of UBER for ECP and ECP's extension, Yoda with FLOWER at 10^{-3} fault incidence rate. Colors indicate different fault map allocations: Orange: 6.25%, Green: 3.13%, Purple: 1.56%, Blue: 0%.

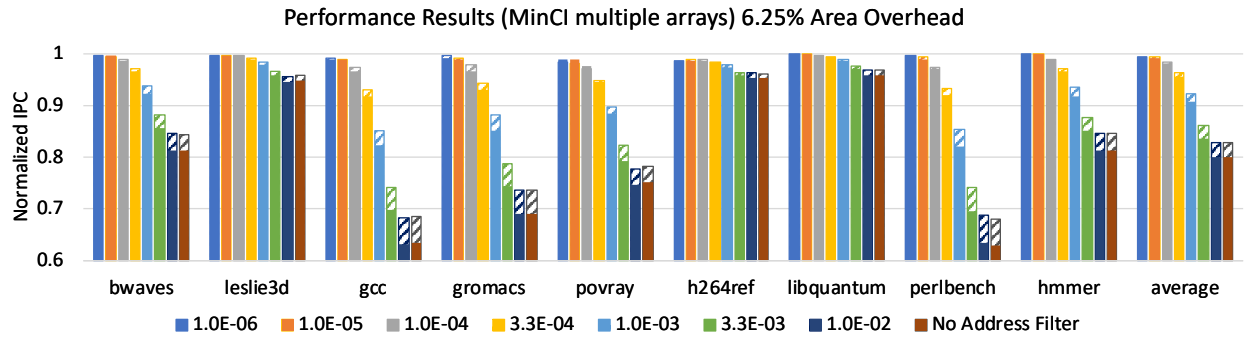


Figure 45: IPC over 9 SPEC Benchmarks at 6.25% area overhead at different error rates (10^{-6} to 10^{-2}) for FLOWER. Striped bars indicate IPC improvement from using in-memory operations.

Table 12: Architecture Parameters

CPU	Cache
4 out-of-order cores	Private L1 32KB Inst, 32KB Data
4 issue width, 4GHz clk	Private L2 256KB/Core
45 nm Technology	Associativity: 8 (L1 data and L2)
1GHz Frequency	Block Size: 64B
Memory: DDR3-1066 (8-8-8)[98]	
2 channels, 1 rank per channel, 8 banks per rank	
4KB address filter, 3ns latency [32]	
50ns (Subarray), 155ns (Inter-Bank) RowClone [87, 88]	

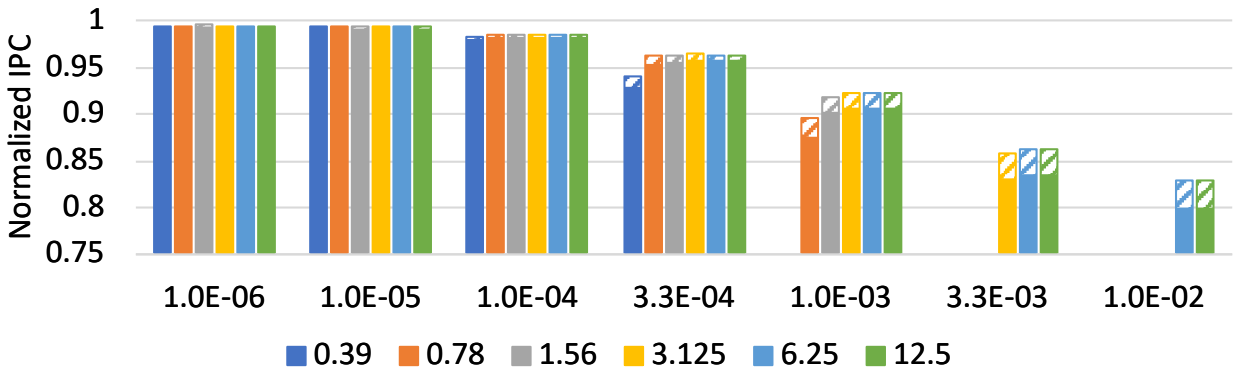


Figure 46: Average IPC over 9 SPEC Benchmarks for different fault map area overheads (0.39% to 12.5%) and error rates (10^{-6} to 10^{-2}). Striped bars indicate IPC improvement from using in-memory operations.

8.0 HOTH and Neutron Radiation Experiments

8.1 The HOTH Faultmap

To address the aforementioned issues with overhead at scale and fault tolerance, we propose a fault map with a more efficient mapping and built-in protection. Our HOTH fault map is similar in structure to a set-associative cache, where each way contains an identifying tag, a pointer to the weak cell location within the row, and a replacement bit. To access a way, a hash function on part of the physical address of the row maps it to an entry in a dedicated section of memory. The remaining address bits are used for the tag to identify existing weak cell entries. Figure 47 demonstrates the process of mapping an address to an entry. The structure of HOTH is similar to Archshield, but an important distinction is that the table entries are bit-level and implement TMR in HOTH, as opposed to being word-level replicas in Archshield. Each entry is reproduced in triplicate, in-place, such that the fault map itself is protected from faults.

As with any hash function, or set associative structure, there is potential for collisions that exceed the number of ways. In caches, this requires eviction. However, in this context, if the number of collisions exceeds the number of ways, the fault map will fail. This can be detected and resolved by rebuilding the fault map with a new hash function; this is an expensive operation, but as we will show, it has an extremely low probability and would require a one time cost to rehash. To estimate the probability of this scenario, we can apply the binomial probability of having k entries in any one of m buckets over n trials:

$$m\binom{n}{k}\left(\frac{1}{m}\right)^{k}\left(1-\frac{1}{m}\right)^{n-k} \tag{8.1}$$

The expression in Eq. 8.1 counts any situation where one or more buckets have k entries separately, but in this context, any such instance is a failure. To collapse these duplicates, all duplicate entries can be captured in a series, and then removed via the inclusion-exclusion principle as follows:

$$\sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} (-1)^{i+1} \binom{m}{i} \left(\prod_{j=0}^{i-1} \binom{n-jk}{k} \right) \left(\frac{1}{m} \right)^{ik} \left(\frac{m-i}{m} \right)^{n-ik} \tag{8.2}$$

To quantify the probability of failure and overhead for HOTH on our specific system, the memory size, row size, fault map size, and associativity are relevant. For a 16 GB memory with 64 byte (512 bit) rows, we have 34 bits of address, 5 of which are used for byte addressing. If 15 bits are hashed to determine the fault map row index, then the remaining 14 bits are used as tag bits. Thus, each entry in the fault map needs to store a 14 bit tag, a bit to indicate it is in use, and 9 bits for a pointer and 1 spare bit. After applying TMR, each entry requires 75 bits, thus, 6 entries will fit in each memory row. Setting the overall fault map size to 128k entries (<0.05% overhead), we then have $\lceil \frac{128k}{6} \rceil = 21846$ rows in the fault map. Based on how many weak and failed cells we anticipate, we can determine the probability of failure using Eq. 8.2. If we expect to see 200 weak or failed bits, the failure condition is computed with $m = 21846$, $k = 7$, and $n = 200$, resulting in a probability of $2.1 \cdot 10^{-14}$ of requiring a different hash function. Extending this to 1000 and 10k vulnerable bits, we expect failure with probability of $1.7 \cdot 10^{-9}$ and 0.011 respectively. Empirically, we observed 239 weak cells, composing a fault incidence rate of $2.5 \cdot 10^{-13}$. Given that the table failure is correctable, and the probability is only computed once for a given memory and set of hash bits, we conclude that this is a reasonable set of parameters for HOTH in our system architecture.

Access to the HOTH fault map will be a significant overhead, as it requires a read from memory followed by a search for the entry that must occur after the read is completed. Other systems that rely on a fault map have reported significant performance degradation and have developed solutions to address this performance bottleneck, such as address Bloom filters that filter access to the fault map when no faults appear in the fault map [32], which consume cache resources and are not performance and area overhead free. Given that HOTH leverages a form of ECC in Chipkill, this replaces the need for an address filter. The HOTH faultmap will only be accessed in the infrequent case of a reported ECC failure making the aggregate performance overhead extremely low while improving system reliability.

8.2 Procedure

To test our hypotheses we created a test rig consisting of the computation equipment described in Table 13. We arranged the rig to allow multiple DRAM DIMMs and the chips that comprise the DIMM to be placed into a linear path in a radiation beam at the Los Alamos Neutron Science Center (LANSCE) as shown in Figure 48. The figure actually shows the equipment from multiple concurrent experiments in the path of the beam, including the exposure of Intel Optane (XPoint) Solid State Memory (in vertically oriented motherboard), which we comment on further in Section 8.3.4. The motherboards with the DRAM under test were at the front of the beam path arranged horizontally. The DIMMs are shown in detail in Figure 49 and were oriented so that the path of the neutron beam was parallel to the DRAM and the center of the beam was aligned through the middle of the center DIMMs, which were arranged in rows of four. The objective of this organization was to minimize the scattering and maximize radiation exposure of the motherboards and CPU while focusing the beam on the DDR3 under test. A 1-inch collimator was used, and the average total fluence of the neutron radiation per bin was $3.74 \cdot 10^{10}\ J/cm^2$.

In order to test the memory function during the radiation exposure in as lightweight a manner as possible, MemTest86 Pro version 7.5 [99] was used. This software runs at the BIOS level before the full OS is loaded, minimizing the memory footprint required while testing. The procedure was to run four sequential instances of MemTest86 Test 5 (moving inversions, random pattern). During the test, any bitflips detected would be automatically saved as part of the testing report to an external flashdrive connected via a 12 foot USB extension cable (to be completely out of the path of the beam). At the conclusion of these four sequential tests, the system would automatically reboot and begin the testing again, to save into a separate log. On the very few occasions where the system crashed during testing, it would be automatically rebooted without saving the failure log. This testing procedure was also conducted before turning the beam on, to verify the memory we were using had no observable faults and was fully functional in a standard operating environment.

Through the testing period, we observed two distinct phases: testing in the direct path of the neutron radiation beam for three days, and testing the rigs outside the path of the

radiation beam afterwords for four days. With this second test, we observed errors caused by the decay of irradiated material without direct neutron bombardment in order to observe if there were any noticeable differences in our hypothesis of weak cell existence and their prevalence.

8.3 Results

In Section 8.3.1 we first discuss the observed results from our neutron radiation tests, and then in Section 8.3.2 we use our observed results to approximate fault rate of HOTH compared to standard ECC-based fault mitigation strategies.

8.3.1 Experimental Results

The results from the beam testing experiment are shown in Table 14. Byte-level errors were only observed when the rig was placed in the beam (phase 1), despite the slightly longer testing period out of the beam (phase 2). We can see in both phases that a very small percentage of the total cells, weak cells, make up a majority of the errors observed. The definition of a weak cell for a given threshold T is that the bit experiences a number of faults F such that $F > T$. Some cells entirely failed, becoming stuck at a particular value, which are reported separately from weak cells.

In the beam, 82 weak cells were responsible for 67.3% of the total (not including stuck-at) errors observed, and all experienced at least 50 bitflips each (T=50). During the experiment 17 weak cells became effectively stuck. When combining weak and stuck-at cells, the total percentage of single bit faults covered was $>95\%$. "Random" bit flips are the total bit flips minus the stuck-at and weak cell bit flips. Thus, as more cells are classified as weak, fewer cells are random, and vice versa. For the total number of non-stuck errors experienced, the expected probabilistic value of bit flips per bit is less than one, so T=3 (four or more flips) is already a significantly higher percentage than random chance, making it a sufficient threshold to weakness classification.

One aim of the experiment was to test for an architectural relationship between the locations of the weak and stuck cells. Figure 50 depicts a heat map in terms of number of bit flips for T=3, where stuck-at cells were considered saturated at the highest number of weak cell faults. Among these cells, there was no noticeable architectural trend (columns/chip correlations), and each memory row had at most one weak cell. Accordingly, this heat map and other heat maps of weak cells with other data groupings do not appear to have any distinguishable patterns (such as repeated offsets) that might not be obvious from statistical analysis. Thus, there is no evidence of any architectural relationship between the locations of the weak cells or permanently failed cells.

Figure 51 shows the heat map of bit flips when the weak and stuck cells are excluded, showing that byte-level errors dominate this distribution. We hypothesize that each byte of every word belongs to the same chip, which experiences latch-up, which is why a noticeable pattern of chip failures across rows is repeated every 64 bits. Using Chipkill, the codewords would be interleaved across the chips, allowing it to tolerate byte-level errors from a single chip. In this case, the repeating pattern of byte errors every 64 bits would not be observed on the row level, but would be observed in adjacent rows.

In the next subsection, we use these frequencies of byte-level, weak/stuck cell, and random faults in Table 14 to evaluate the impact of HOTH on expected UBER.

8.3.2 HOTH Fault Tolerance Results

To evaluate the correction capability of HOTH compared with traditional error correction methodologies of SECDED and Chipkill, we probabilistically generated a larger fault stimuli set using the frequencies of byte-level faults, weak-cell faults, and random faults reported in Section 8.3.1. For modeling, we assumed byte-level, weak-cell, and random faults occurred with independent probability and scaled linearly with the radiation flux.

Throughout the various UBER calculations, we use the same three thresholds for weak cells (T=3,10,50) presented in Table 14. Naturally, as T increases, more faults are moved into the random category, and the frequency of random faults increase, while the frequency of a weak cell flipping also increases (though the total number of weak cells decreases).

Figure 52 demonstrates the UBER per radiation flux for storing data in a row with exactly one weak cell. The reported flux $\mathbf{x}$ is defined as a scaled factor from the radiation in the beam experiment with an approximate time of 10s between accesses: $\mathbf{x} = 3.74 \cdot 10^{11}\ J/cm^2/s$. We report results scaled from 0.1$\mathbf{x}$ to 10$\mathbf{x}$. Results are reported for a row with a "typically" weak cell (circa T=10) with up error bars showing only extremely weak cells (T >50) and down error bars showing borderline weak cells (circa T=3). While undergoing neutron bombardment (phase one) for radiation rate $\mathbf{x}$, the weak cells in the experiment on average have a 1%-2% flip. This is considerably higher than observed random byte-level faults and individual flips and why "no correction" is so poor in these rows. Using HOTH's fault map alone, the UBER can be improved to close to 10^{-12}. ECC1 slightly improves upon this by also covering the random bit flips, though its correction capability is hindered by its inability to protect the frequent byte-level faults that occur in the same chip yielding a <1% improvement. Chipkill (CK) outperforms ECC1 with 38-87× improvement, depending on cell weakness, due to its ability to tolerate byte-level latchup faults in addition to single bit flips. Augmenting Chipkill with HOTH provides a substantial $7 * 10^8$ to $9 * 10^8$ × UBER improvement, depending on cell weakness, over Chipkill alone. Essentially, CK+HOTH provides an extremely low-overhead "poor man's" ECC2 where at least one fault is predictable. This highlights the value of weak cell knowledge in high radiation environments. Figure 52 shows that as flux is scaled, the UBER magnitude follows the flux trend, but the seven orders of magnitude in improvement of adding HOTH over CK scales.

Using TMR for the HOTH faultmap at x flux results in a HOTH access UBER of $5.3 \cdot 10^{-26}$, which is significantly better than the best case UBER of weak cell DRAM data protected HOTH+Chipkill (10^{-22}).

The error rates observed after the devices were removed from the beam and subject to secondary radiation (phase 2) are shown in Figure 53. Scaling $\mathbf{x}$ assumes a scaled dose of that magnitude to generate the secondary radiation. HOTH alone achieves an UBER between $4 \cdot 10^{-13}$ and $8 \cdot 10^{-14}$, depending on cell weakness. ECC1 only protects slightly better than HOTH alone. ECC1 and Chipkill are indistinguishable, as there are no longer multi-bit latchup faults. However, the correction capability of HOTH combined with Chipkill (similar to HOTH+ECC1) yields a $7 \cdot 10^8\times$ to $2 \cdot 10^8\times$ improvement, depending on cell weakness,

reducing the probability of error to the probability that two or more random errors occur in the same word. The out-of-beam UBER with TMR and Chipkill to protect the HOTH fault map is $3.3 \cdot 10^{-30}$, which, again, is many orders of magnitude better than the UBER of HOTH+Chipkill, and thus can be safely used to protect it.

8.3.3 Correlation Non-Radiation Reliability Concerns

To test if the weak cells due to radiation were correlated to other sources of reliability concerns, we attempted to determine cells due to cross talk discovered through row hammering. We relaxed the refresh rate of the DRAM and performed the rowhammer memory test to observe the locations of these weak cells from crosstalk. Using a threshold of T=3 over 28 tests, we observed 2,639 cells weak to rowhammering, and for a threshold of T=10, we observed 324 cells. None of these weak cells overlapped with any of the hundreds of cells weak or stuck due to radiation, indicating no apparent correlation between these forms of weakness. This result is not entirely unexpected, as prior work showed there was no correlation between other forms of weakness, such as cells with lower retention time and cells weak to rowhammering [14].

8.3.4 Secondary Qualitative Results

We observed BRAM (SRAM-based) memory in four Xilinx Zynq FPGAs. In total, there were fewer faults observed over six days in the beam (double the DRAM time) with 4,605 total bit flips. There was no apparent architectural pattern associated with bit flips, and no bit flipped more than twice. We also tested an Intel Optane Memory [29]. Over a 2.5 day period of consecutive reads and writes, we observed no faults. Unfortunately, the rig failed due to radiation scattering. In post-experiment testing, the Optane was still fault free.

8.4 Conclusions

As process scaling continues, resulting in increased process variation, it has been observed that DRAM and other memories have "weak cells," which are significantly more vulnerable than other cells to a particular type of fault. While this has been demonstrated for retention and cross talk, in this work, we provide evidence that radiation is another fault mode that has an imbalance of vulnerability in DRAM cells leading to "radiation weak cells." We propose a new, low-overhead fault map, HOTH, to redundantly represent these cells, thus providing substantial benefits to UBER both during radiation exposure and due to secondary radiation after exposure ($\geq 10^8\times$). This demonstrates the potential to leverage COTS memory in radiation compromised environments. Our future experiments will explore more types of radiation exposures and additional memory technologies and architectures.

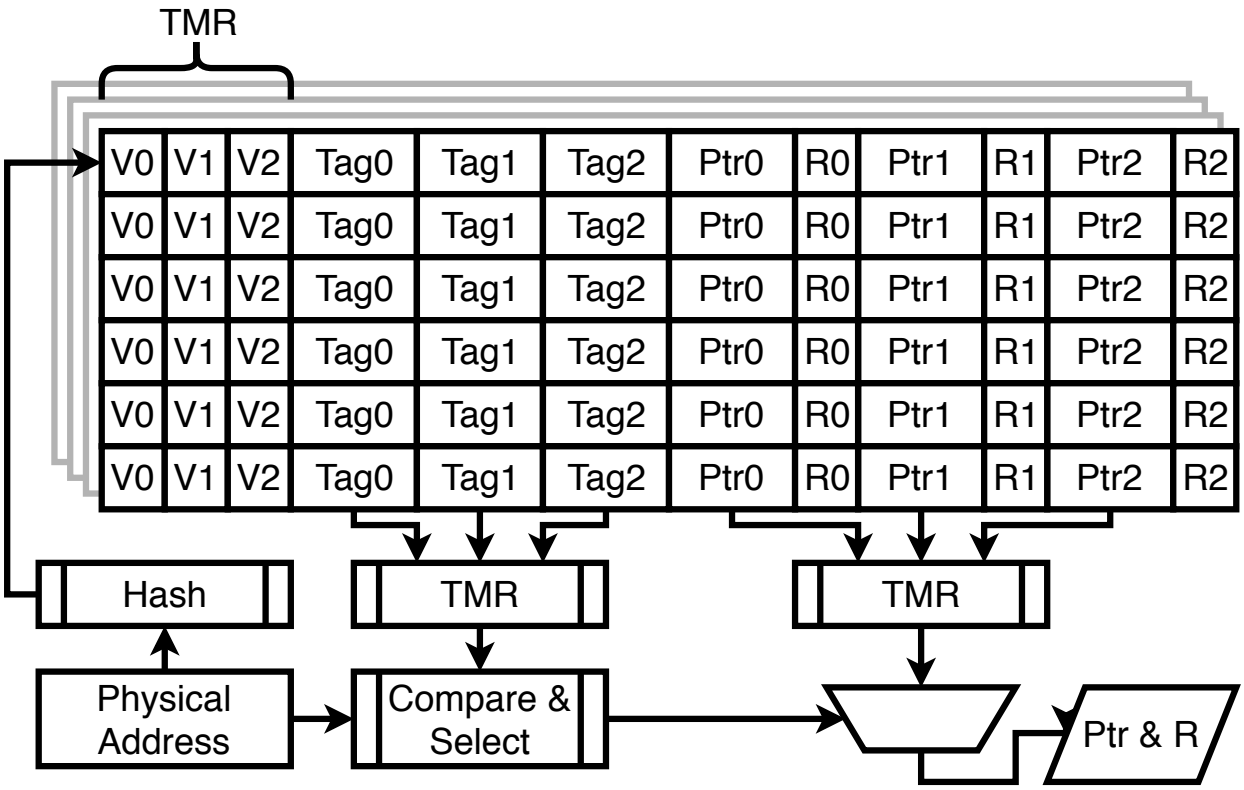


Figure 47: Architecture for the HOTH fault map. A masked physical address is hashed to table rows, each bin has 6 "ways." Each way uses remaining physical address to test "tag" bits for valid "V" entries. The payload is the appropriate pointer and spare bit. Entries are protected with TMR as indicated.

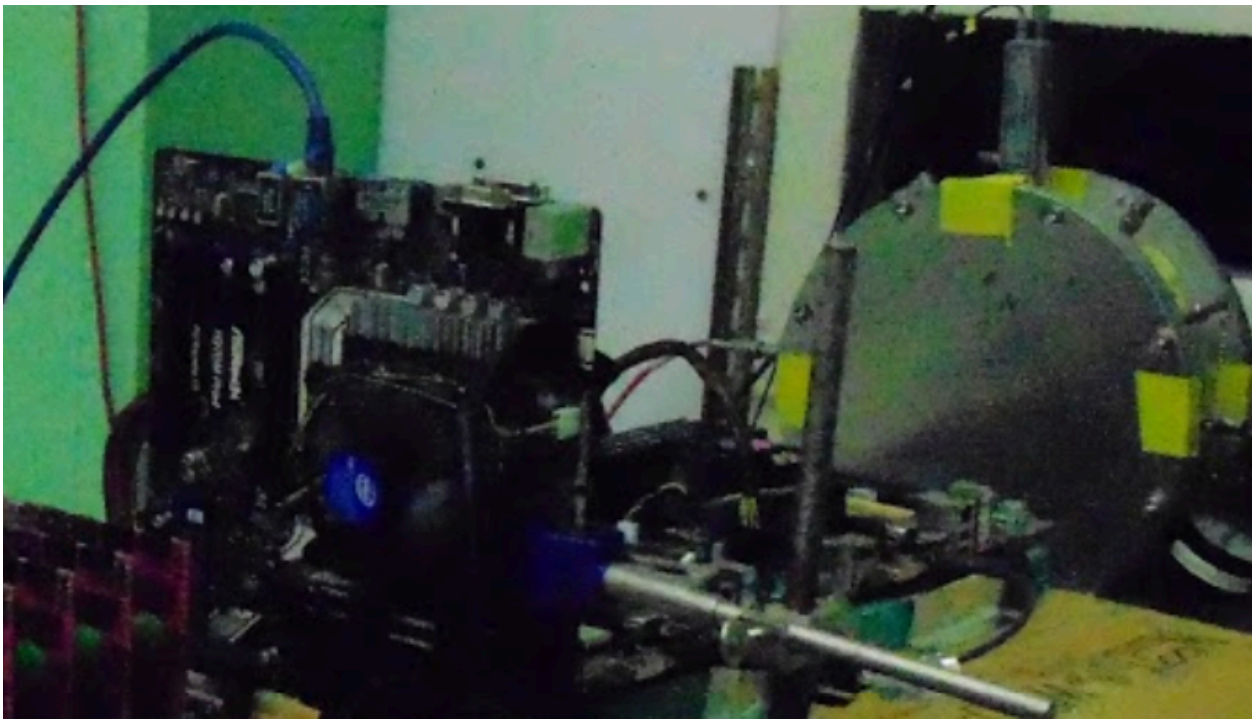


Figure 48: Neutron beam test at Los Alamos Neutron Science Center (LANSCE).

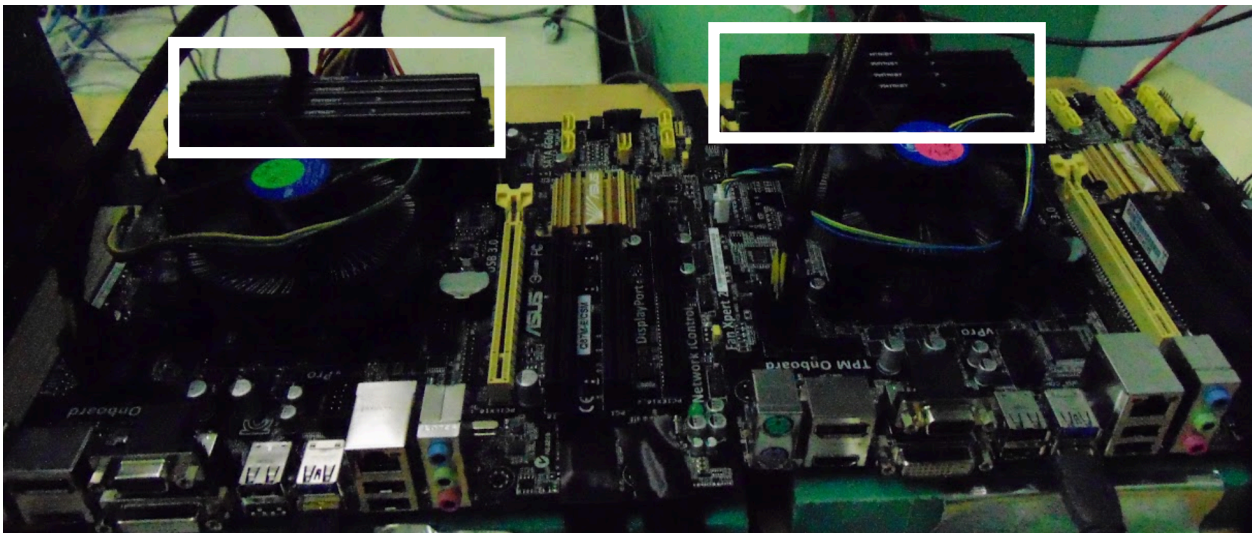


Figure 49: Detailed rig with DIMMs parallel to beam path highlighted in white.

Table 13: Devices under test

Motherboard	2 ASUS Q87M-E/CSM, DDR3 1600MHz, LGA 1150
DRAM	8 Patriot Viper 3 Series, Black Mamba, 4GB (Micron)
Processor	2 Intel Core I3-4160, 3.60 GHz, 2-Core Hyper-Threading

Table 14: Results from neutron radiation experiment

	In Beam	**Out of Beam**
Total Bits Written	9.42374E+14	2.61051E+15
Byte-level Errors	754	0
Number of Weak Cells, T=3	239	174
Number of Weak Cells, T=10	155	141
Number of Weak Cells, T=50	82	101
Weak Cells T=3 Total Flips	31948	122689
Weak Cells T=10 Total Flips	21128	111873
Weak Cells T=50 Total Flips	19394	98352
Permanent Failed Bits	30	
Random (T=3) Flips	1027	90
Random (T=10) Flips	1453	297
Random (T=50) Flips	3187	1376

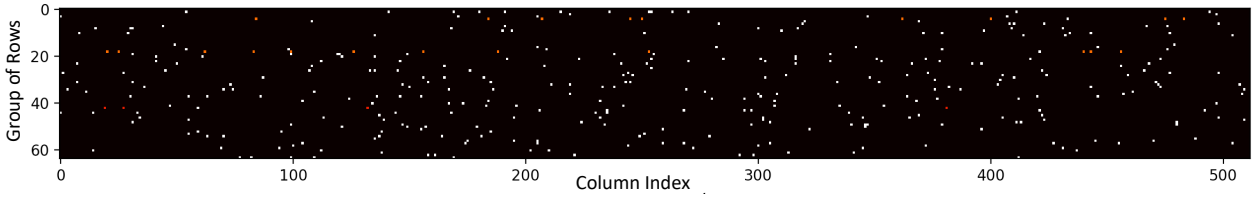


Figure 50: Weak and stuck-at cell heatmap (T=10) over groups of rows for 16 adjacent 32 bit word accesses. Rows split evenly according to address space.

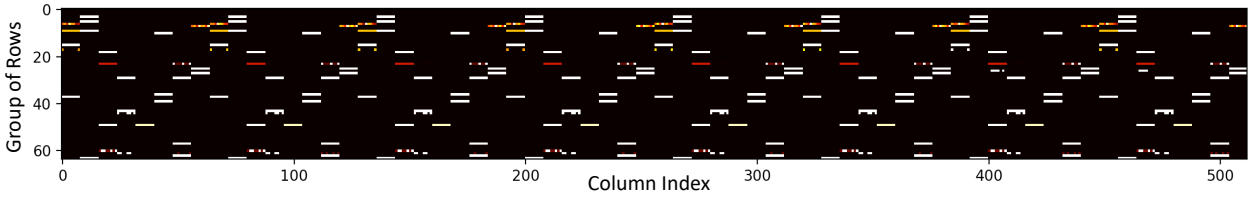


Figure 51: Heatmap of all bitflips excluding weak cells and stuck-at cells over groups of rows for 16 adjacent 32 bit word accesses.

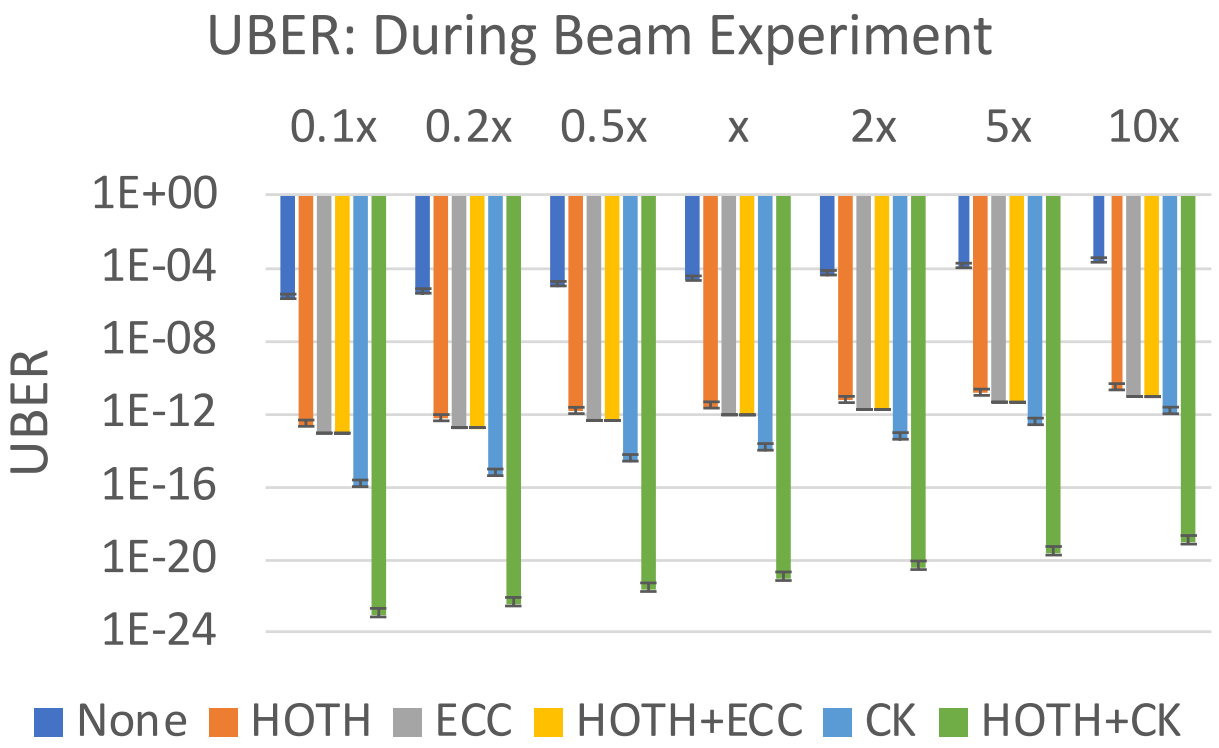


Figure 52: UBER for rows with one weak cell based on frequency of failures in the neutron radiation beam assuming different weak cell thresholds. Error bars show range based on cell weakness.

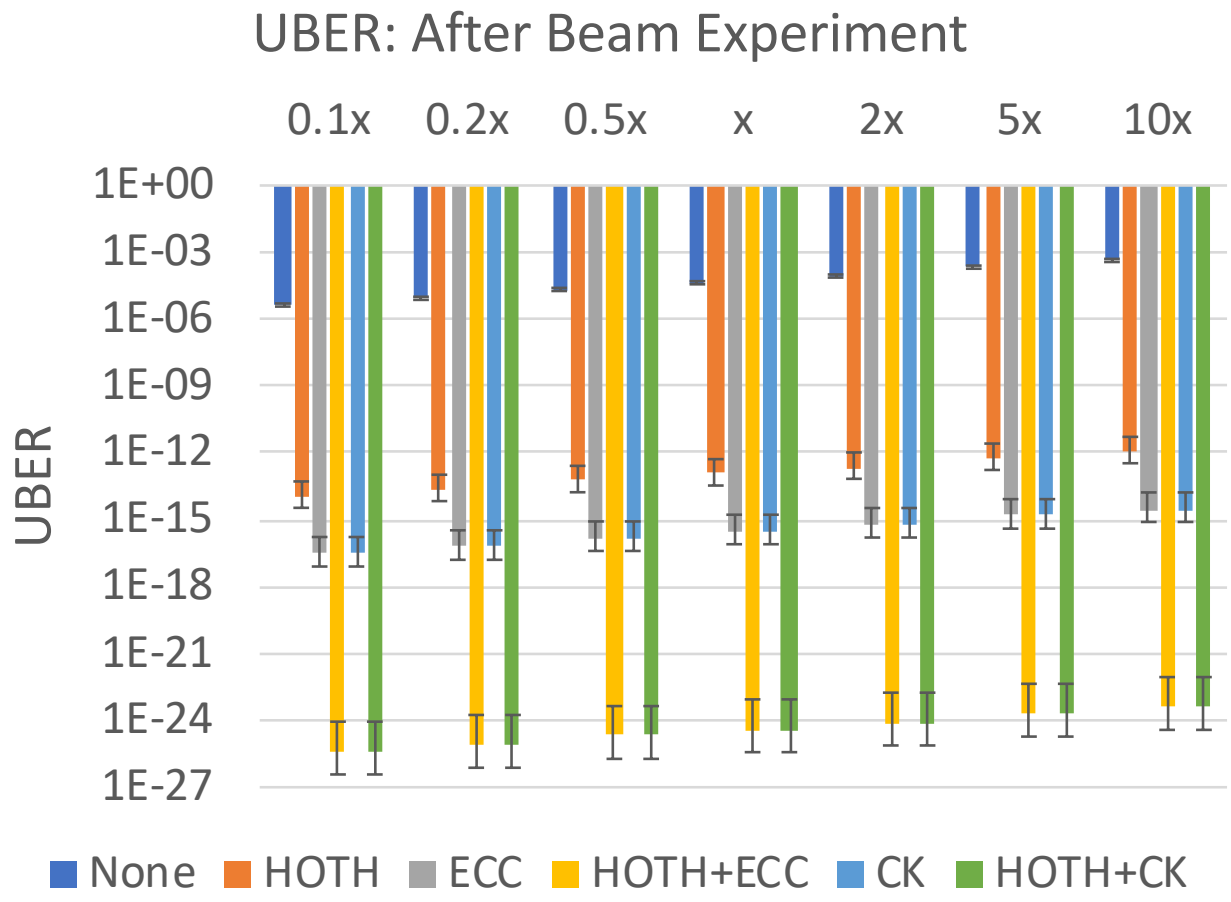


Figure 53: UBER for rows with one weak cell based on frequency of failures after the neutron radiation beam assuming different weak cell thresholds. Error bars show range of cell weakness.

9.0 PFE: Periodic Flip Encoding

9.1 Design

PFE is a simple and effective solution to reduce the number of '000' and '111' sequences in data to be written, which is based on flipping '0' and '1' of sub blocks interleaved throughout the data. Specifically, to write an N-bit data word, $W = x_0, ..., x_{N-1}$, we propose a low overhead PFE technique that reduces the occurrence of bad patterns by partitioning W into 3-bit groups. We either keep the original data unchanged or flip a specific bit (1st bit, 2nd bit, or 3rd bit) in all groups to obtain three possible code words, $W1$, $W2$, and $W3$. The code word that minimizes the number of bad patterns is selected. To decode the generated code word, the encoder should supplement the data word with two auxiliary bits to record which of the four code words were used and enable the decoder to retrieve the original data word through simple logic operations.

We use the idea of PFE to tolerate crosstalk errors via two different modes: fault-oblivious PFE and fault-aware PFE. In fault-oblivious PFE, all four code word candidates are generated and the code word containing the minimum number of bad patterns is written in memory without knowing the location of weak cells. In fault aware PFE, a generated off-line map of weak cells maintains information about the location of weak cells and is used to minimize or avoid the overlap of bad patterns and weak cells. We describe the two modes of PFE in the following sections.

9.1.1 Fault Oblivious PFE ($\mathbf{PFE_{FO}}$)

Figure 54 depicts fault-oblivious PFE. Figure 54(a) shows the original data word appended by two auxiliary bits set to '00' to indicate that no bits are flipped in the encoded data word. In addition to the bad patterns in the original data word, the last bit of the data word and the two auxiliary bits are concatenated, and thus may introduce a new bad pattern. For example, concatenating '000' and '00' in Figure 54(a) increases the number of

'000' sequences from two to four. Note that if k consecutive zero (one) bits are concatenated with '0' ('1'), the number of '000' ('111') sequences is k+1. Figure 54(b) shows the code word when flipping the first bit of each group. While flipping the first bit of each group breaks up the first '000' sequence, it introduces other '111' sequences that are equally problematic. Figure 54(c) flips the second bit of each group and removes all bad patterns from the data word. Finally, Figure 54(d) toggles the third bit of each group which results in four bad patterns. Hence, PFE examines four possible code words and selects the one with the minimum number of bad patterns to be written into memory. Note that this type of PFE does not require any information about the location of weak cells.

9.1.2 Fault-Aware PFE ($\mathbf{PFE_{FA}}$)

In contrast to fault-oblivious PFE where the encoded candidate with the minimum number of bad patterns is chosen, fault-aware PFE utilizes weak cell information to select the code word that minimizes (or eliminates) the occurrence of crosstalk errors. As will be shown in Section 9.3, PFE_{FA} has the additional advantage of being able to guarantee the mitigation of at least three weak cells in a block. *When writing an N-bit data word, $W = x_0, ..., x_{N-1}$ onto N DRAM cells $c_0, ..., c_{N-1}$, PFE_{FA} uses information about the weakness of cells to select the code word that avoids the overlap of the center of a 3-bit bad pattern with a weak cell.*

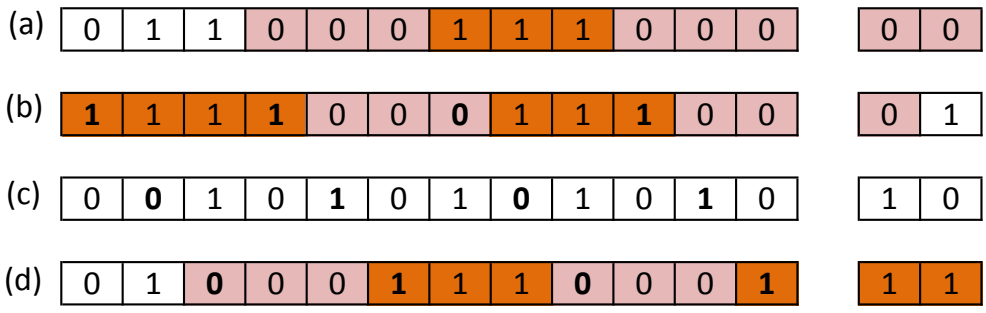


Figure 54: Encoding W = 0x638 using PFE.

A map of weak cells can be discovered as part of memory regression tests during the memory testing phase. For example, this testing can write bad patterns (all zeros, all ones, alternating 01's, etc.) many times to map out the locations of weak cells or by using well-known testing algorithms [17, 25, 100]. The weak cell information can be stored in resident memory and cached in the on-chip cache on-demand, as was proposed in [11]. Alternatively, the weak cell information can be stored in a ROM that can be accessed by the hardware.

9.2 Memory Controller Implementation

Figure 55 shows the implementation flow of fault-aware PFE in a memory controller. The memory controller first computes the address of the fault map entry and then checks ① whether the entry exists in the last level cache or not. If not, a request is issued to the main memory ② in order to bring and store the fault map entry of the corresponding address in the last level cache [11]. The location of weak cell(s) is temporarily recorded in a weak cell Map (WCM). The 512-bit original data and WCM ③ are then divided into 32-bit blocks and sent to 16 encoding modules that leverage the weak cell information to map each 32-bit block to 34-bit code words. The encoding module utilizes a priority encoder ④ to select the code word which avoids the overlap of bad patterns with the location of weak cells. The 16 encodings are done in parallel and the final 16 generated code words are concatenated and sent ⑤, along with the memory address, to be written in memory ②.

The PFE_{FO} encoder implementation is similar to the PFE_{FA} encoder implementation except that it does not include the fault map generator. Furthermore, it requires 5-bit counters in the encoding modules to count the number of bad patterns in each of the four generated code words to select the code word with the minimum number of bad patterns. For a 512-bit cache line, PFE_{FO} encoder requires 64 5-bit counters that produce an extra cost area overhead in comparison to PFE_{FA} encoder.

Note that an additional step may be required to unscramble the logical bit locations in order to determine the logical locations of physically neighboring bits, which are subjected

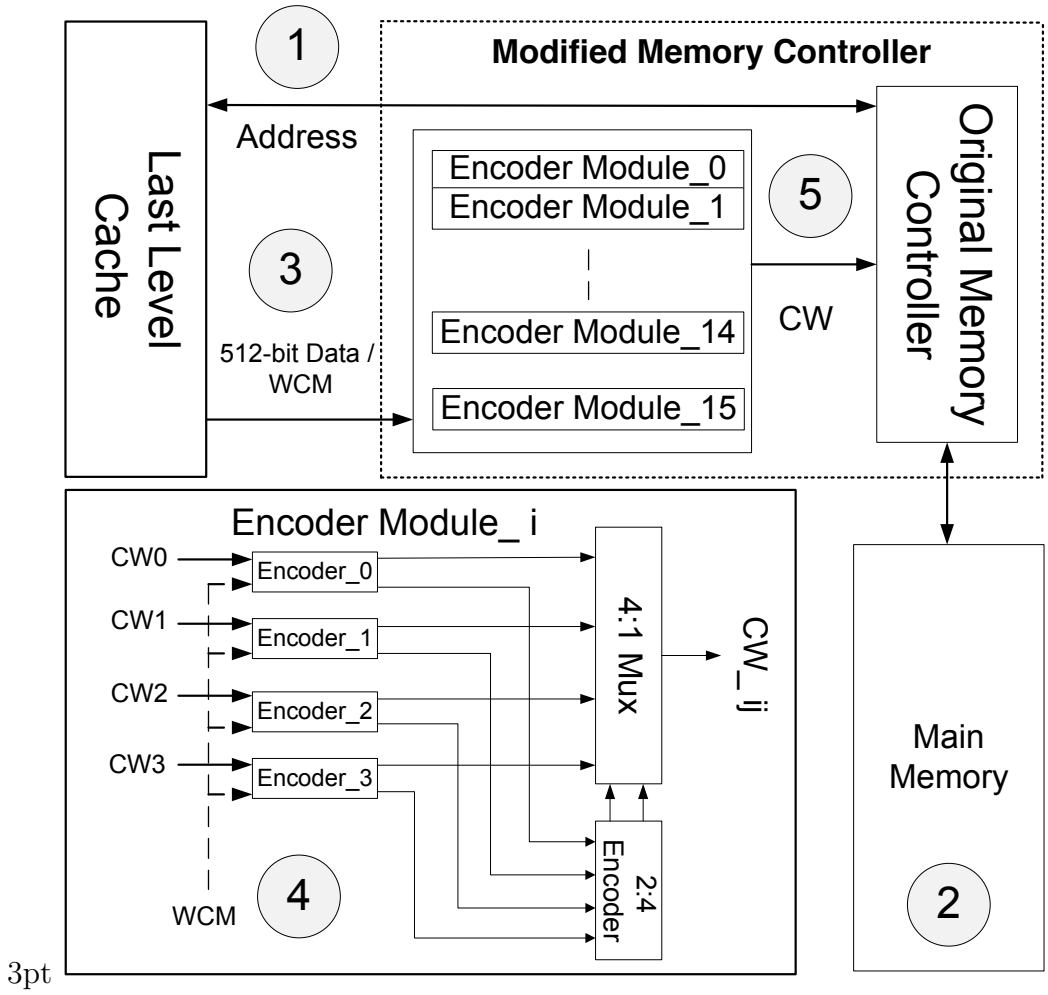


Figure 55: Memory controller implementation of fault aware PFE.

to the highest levels of crosstalk. This information, which is known to vendors, can also be discovered [17].

9.3 Tolerance Capability

Given the four code word candidates from PFE, in PFE_{FA} there are some guarantees that can be made about the protection capability. In what follows, we will first prove that

PFE_{FA} can tolerate atleast three of the weak cells storing the data bits, then we will present an extension which will enable PFE to tolerate at least three weak cells in either the data bits or the auxiliary bits.

Theorem 1. If an N-bit data word, $W = x_0, ..., x_{N-1}$, is to be written to N cells, $c_0, ..., c_{N-1}$, then using PFE_{FA} encoding will tolerate at least three weak cells assuming that the auxiliary bits used in the encoding are stored in a reliable memory.

Proof: Given any sequence of three consecutive bits $p =$ "$x_{i-1}\ x_i\ x_{i+1}$", applying the three PFE transformations will change this sequence to $p_1 =$ "$x'_{i-1}\ x_i\ x_{i+1}$", $p_2 =$ "$x_{i-1}\ x'_i\ x_{i+1}$" or $p_3 =$ "$x_{i-1}\ x_i\ x'_{i+1}$", where x' is the complement of x. Table 15 indicates that for any sequence, p, at most one of $p,\ p_1,\ p_2$ and p_3 will be equal to '000' or '111'. Hence, if c_i is a weak cell, then three of the four sequences $p,\ p_1,\ p_2$ or p_3 (and thus three of the four PFE code words) will tolerate the weakness of this cell. Consequently, if three cells $c_i,\ c_j$ and c_k are weak, $0 <\ i,\ j,\ k <\ N-1$, then at least one of the four PFE code words will tolerate the weakness of the three cells. ■

If two auxiliary bits x_N and x_{N+1} are used to indicate which of W, W_1, W_2, or W_3 is used in the encoding and these bits are written to the two cells $c_N,\ c_{N+1}$ following the data cells, then it is only possible to prove that PFE can tolerate the weakness of at least two of the cells $c_0, ..., c_{N+1}$. This is because the two auxiliary bits can generate a bad pattern in more than one of the four possible encodings. However, we will prove in the next theorem that by using three auxiliary bits, $x_N,\ x_{N+1}$ and x_{N+2} to record which of the code words, W, W_1, W_2, or W_3 is used, and storing these bits in the three cells $c_N,\ c_{N+1}$ and c_{N+2}, it is possible to tolerate the weakness of at least three of the $N+3$ cells storing the data and the auxiliary bits. Specifically, we define PFE+ as a PFE scheme which uses three auxiliary bits such that '000' indicates the original data word, '100' indicates the encoding in which bit x_{N-3} is flipped (call it W_1), '010' indicates the encoding in which bit x_{N-2} is flipped (call it W_2) and '001' indicates the encoding in which bit x_{N-1} is flipped (call it W_3). With this scheme, we prove the following:

Theorem 2. If an N-bit data word, $W = x_0,\ ...,\ x_{N-1}$, is to be written to N cells, $c_0,\ ...,\ c_{N-1}$, then using fault aware PFE+ encoding with the three auxiliary bits $x_N,\ x_{N+1}$

and x_{N+2} written into cells $c_N,\ c_{N+1}$ and c_{N+2}, will tolerate at least three weak cells $c_i,\ c_j$ and c_k, $0 < i, j, k < N + 2$.

Proof: It was shown in the proof of Theorem 1 that if c_i, $0 < \ i < \ N - 1$, is a weak cell, then three of the four code words, $W,\ W_1,\ W_2$ and W_3 will tolerate the weakness of this cell. Here, we will prove that the same applies to any weak cell, c_i, $N - 1 \leq i < N + 2$. For this, we observe that the five bits x_{N-2} ... x_{N+2} will have one of four possible forms: "$x_{N-2}\ x_{N-1}$ 000" (if W is used), "$x_{N-2}\ x_{N-1}$ 100" (if W_1 is used), "$x'_{N-2}\ x_{N-1}$ 010" (if W_2 is used), "$x_{N-2}\ x'_{N-1}$ 001" (if W_3 is used). We consider the following three cases:

1. Cell c_{N+1} is weak: the three bits $x_N,\ x_{N+1},\ x_{N+2}$ can produce a bad pattern (000) only if W is used in the encoding. Hence, the three other encodings can tolerate a weak c_{N+1}.

2. Cell c_N is weak: the 3-bit sequence $x_{N-1},\ x_N,\ x_{N+1}$ can equal "x_{N-1} 00", "x_{N-1} 10", "x_{N-1} 01" or "x'_{N-1} 00" if W, W_1, W_2, or W_3 are used for the encoding, respectively. Hence, "$x_{N-1}\ x_N\ x_{N+1}$" cannot be equal to 111 and can only be equal to 000 either if W is used (when $x_{N-1} = \ 0$) or if W_2 is used (when $x_{N-1} \ = 1$). In other words, for any value of x_{N-1}, three encodings can tolerate a weak c_N.

3. Cell c_{N-1} is weak: the 3-bits sequence x_{N-2}, x_{N-1},
x_N can equal: "$x_{N-2}\ x_{N-1}$ 0," "$x_{N-2}\ x_{N-1}$ 1," "x'_{N-2}
x_{N-1} 0," or "$x_{N-2}\ x'_{N-1}$ 0" if $W,\ W_1,\ W_2$, or W_3 are used for the encoding, respectively. It is straightforward to check that for any specific values of $x_{N-2}\ x_{N-1}$, only one of the encodings $W,\ W_1,\ W_2$, or W_3 will produce 000 or 111. Hence, the three other encodings can tolerate a weak c_{N-1}.

Consequently, if three cells $c_i,\ c_j$ and c_k are weak, $0 < i, j, k < N + 2$, then at least one of the four encodings will tolerate the weakness of the three cells. ■

Clearly PFE relies on the specific information about the nature of the faults to largely outperform more general error correcting codes. Specifically, at least $2logN$ and $3logN$ auxiliary bits have to be used when ECC-2 and ECC-3 are used to tolerate the errors resulting from two or three weak cells, respectively. In contrast, only 2 or 3 auxiliary bits (independent of N) are needed to tolerate two or three faults using PFE or PFE+, respectively. Moreover, ECC-k cannot tolerate more than k faults while PFE and PFE+ can tolerate more than two

Table 15: PFE transformations of 3-bit sequences.

p	000	001	010	011	100	101	110	111
p_1	100	101	110	111	000	001	010	011
p_2	010	011	000	001	110	111	100	101
p_3	001	000	011	010	101	100	111	110

or three faults, respectively, with some probability. This is because when q instances of the same 3-bit bad pattern overlap q weak cells, the same code word in PFE/PFE+ will tolerate all q weak cells. Note, however, that PFE/PFE+ are designed to tolerate errors induced by crosstalk faults, while ECC can tolerate errors due to any type of faults, including transient faults. To overcome this limitation, it is possible to combine PFE+ with ECC-k to tolerate k transient and three crosstalk faults. This will be more efficient than using ECC-$(k+3)$ to tolerate the same faults. Finally, PFE/PFE+ encoding and decoding are much simpler than ECC-2/ECC-3 and leads to a much simpler implementation as described next.

9.4 Experimental Methodology

To evaluate the fault tolerance capability of ECC, ECP, FFE, and PFE, we developed a PIN-based simulator [101] to model the cache hierarchy in order to determine the accesses to main memory. We also used the Gem5 full system simulator [66] to evaluate the performance overheads of the PFE fault aware scheme. The system parameters were designed to be similar in both simulation environments and a listing of the relevant parameters for Gem5 are shown in Table 16. The PIN simulator uses a similar L1/L2 cache configuration. To evaluate the fault-rate, the PIN simulator evaluates main memory writes by encoding the data and recording a fault if the center bit of a bad pattern in the encoded block is aligned with a weak cell. To model weak cells of the memory, maps of weak cells were created using Bayesian distributions to mimic the impact of process variation and include spatial

correlation of faults [21, 85]. We followed the model described in [85] to generate maps of weak cells.

Errors can be mitigated by (1) reducing the probability that bad patterns coincide with weak cells by reducing the number of bad patterns, as in FFE (Four-to-Five Encoding) and PFE_{FO}, (2) avoiding bad patterns that coincide with weak cells, as in PFE_{FA} or (3) correcting faults, as in ECC. ECP can be used to protect against potential faults by pointing to weak cells and providing reliable storage for their content (will be called ECP_{FO}). Alternatively, ECP can be used to correct faults by pointing to and storing the values of weak cells that overlap with the *center* of any bad pattern (will be called ECP_{FA}). We consider both approaches.

In our experimental tests, we measure UBER when applying FFE, PFE, ECP, or ECC. The amount of storage overhead for all of these schemes is summarized in Table 17 where k, in ECC-k and ECP-k, refers to the maximum number of errors that can be corrected by the scheme. We consider 512-bit cache lines that are divided into n-bit protected blocks. For FFE, n is always 4 and for PFE we use n=32. We use ECC-1_{32} and ECC-1_{128} to denote single error correcting ECC with $n = 32$ and $n = 128$, respectively. Finally, we use $n = 512$ for ECP because it results in 10-bit pointers, compared to 6-bits for 32-bit blocks. This makes the overhead of ECP-3 comparable to the overheads of PFE and ECC-1_{128}, and the overhead of ECP-12 comparable to the overhead of FFE.

We model our fault-aware implementations as described in Section 9.2, where the memory controller must obtain the fault map entry. If the fault map entry is not cached, this can increase traffic on the memory bus, impacting performance. To illustrate this performance overhead, we compare a conservative system without a fault map cache (MemCTRL) to an

Table 16: Simulator Parameters

CPU	4-core, 8-issue width per core, out of order
L1 Cache	16K private Inst. & Data, 8-way set-assoc.
L2 Cache	1MB shared 16-way set-assoc.
Cache Block	512-bits
Write Buffer	64-entries

Table 17: Bit overheads for fault tolerance schemes.

	ECC-k			**FFE**	**PFE**	**ECP-k**	
Overhead per n-bit block	$k(\lceil log(n) \rceil + 1)$			$\lceil \frac{n}{4} \rceil$	2	$k(\lceil log(n) \rceil + 1) + 1$	
	ECC-1$_{32}$	**ECC-2$_{32}$**	**ECC-1$_{128}$**	**FFE**	**PFE**	**ECP-3**	**ECP-12**
Block size	32		128	4	32	512	
Overhead bits per block	6	11	8	1	2	31	121
Overhead %	18.75%	34.37%	6.25%	25%	6.25%	6.05%	23.63%

ideal system (MemDIMM). The conservative system requires access to the memory bus for each encoding operation while the ideal system knows the fault map and can encode the data on the memory DIMMs directly, without incurring additional memory traffic due to querying the fault map.

We performed our evaluations for workloads from the PARSEC [70] and selected SPEC CPU2006 [71] benchmarks[1] for different maps of weak cells including the moderate and high weak cell rates of 0.01% and 1%, respectively. In this context, the weak cell rate is defined as the fraction of weak cells relative to the total number of DRAM cells.

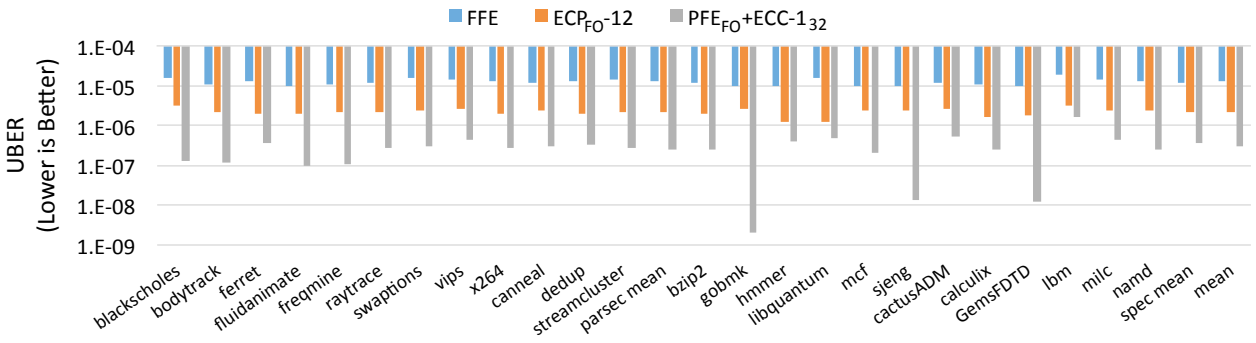
9.5 Results

9.5.1 Fault-Oblivious Effectiveness

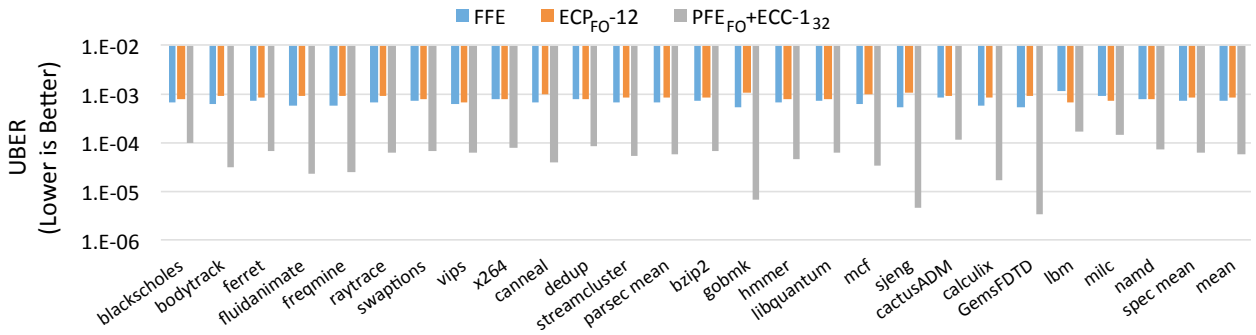
Knowledge of both the location of weak cells and bad patterns is necessary for a fault-aware scheme in the context of intra-row crosstalk. FFE and PFE$_{\text{FO}}$ attempt to minimize the number of bad patterns without regard to the location of weak cells. Thus, it selects the encoding with the fewest bad patterns and upon a tie selects a candidate arbitrarily.

FFE requires a 25% encoding overhead (see Table 17). For an iso-storage-overhead comparison, we combined PFE with ECC-1$_{32}$ (PFE$_{\text{FO}}$+ECC-1$_{32}$), which requires the same overhead of eight bits (2+6) required for FFE. We also compare with the ECP$_{\text{FO}}$-12 approach

[1]The benchmarks selected were those we were able to run successfully within gem5. All SPEC CPU2006 benchmarks were tested using the PIN tool for fault tolerance and are consistent with the results presented here.



(a) 0.01% incidence of weak cells



(b) 1% incidence of weak cells

Figure 56: Comparison of "moderate-overhead" (∼25%) fault-oblivious approaches of FFE with ECP$_{FO}$-12 and PFE$_{FO}$+ECC-1$_{32}$.

with similar overhead (24%). Figures 56(a) and 56(b) show this comparison for weak cell incidence rates of 0.01% and 1%, respectively. For the 0.01% rate of weak cells, the UBER of FFE is 1.2×10^{-5}. However, for the same overhead, ECP$_{FO}$-12 does a much better job achieving an UBER of approximately 2.3×10^{-6} while PFE$_{FO}$ with ECC-1$_{32}$ is even more effective with an UBER of approximately 2.5×10^{-7}. FFE demonstrates its scalablility to higher numbers of weak cells reaching an UBER of approximately 6×10^{-3} with a weak cell incidence rate of 1%. ECP$_{FO}$-12 degrades UBER to match FFE at the UBER of 8.3×10^{-3}. In contrast, PFE$_{FO}$+ECC-1$_{32}$ still provides an UBER of 5×10^{-5}.

Although the proposed techniques decrease the susceptibility to crosstalk through avoiding bad patterns, they cannot completely guarantee tolerance to crosstalk faults. Additionally, bad patterns may be formed when consecutive blocks are concatenated or when the

number of weak cells is larger than what can be tolerated. A single parity bit can also be added to provide a capability to detect an uncorrectable fault similar to the addition of a parity bit in ECC-k to detect a $(k+1)^{th}$ fault. However, the greatest value of using PFE in combination with an ECC-k is to increase the effectiveness of ECC with a much lower overhead than moving to ECC-$(k+\beta)$ for $\beta > 1$.

9.5.2 Fault-Aware Effectiveness

In this section, PFE_{FA} is compared to ECC-1_{128} and ECP_{FA}-3. Each scheme has a storage overhead of approximately 6% additional bits (see Table 17). ECP can be fault-aware in a similar fashion to PFE_{FA} by similarly retrieving the fault locations from the on-chip cache and using the pointers only to store the corrected values for locations at which a bad patterns and weak cells intersect.

Figures 57(a) and 57(b) show the comparison of PFE_{FA} with ECC-1_{128} and ECP_{FA}-3 for a weak cell incidence rate of 0.01% and 1%, respectively. For the 0.01% weak cell incidence rate, ECC-1 achieves an UBER of 6.8×10^{-6} while ECP_{FA} achieves an UBER of 1.4×10^{-6}. In contrast, PFE at 0.01% error produced no errors during our experiments. From the number of writes and repeated experiments, this guarantees that in the worst case it has an UBER of *at most* 3×10^{-12}. This actually exceeds the capability of PFE_{FO}+ECC-1_{32} with a 75% less storage overhead. For the 1% weak cell incidence rate, even with fault-awareness, ECP is only able to correct one-third of the faults dropping below ECC-1's roughly 50% correction rate at the UBER of 2.5×10^{-3}. In contrast, PFE still reaches an UBER of 1.3×10^{-5} and corrects 99.7% of the faults.

9.5.3 Impact on Performance

To enable a fault-aware implementation of PFE and ECP requires additional memory accesses to query the weak cell locations if they miss in the cache, which is an overhead. This is described in Section 9.2. The performance overhead only impacts writing, while read operations proceed normally with a small increase in delay due to decoding (see Table 18). The encoding overheads for fault-oblivious schemes of 2ns or less (see Table 18) combined

with the fact that this encoding could occur in the write-buffer, which masks write latency, resulted in a negligible impact on performance. Thus we only report the performance impact of the fault-aware schemes, which also include this encoding delay in addition to the access to the weak cell map. These performance overheads are reported as instructions per cycle (IPC) in Figure 58 and compared against the performance of a fault-oblivious scheme that does not query the fault map (Baseline). The write-buffer seems to still mask much of the additional latency from the fault-aware schemes. In general, a controller level (MemCTRL) implementation does have a slightly higher performance degradation than a ideal one implemented at the memory chip level (MemDIMM). However, in most benchmarks, the degradation from either scheme is not dramatic. Only in a handful of applications, vips, gobmk, sjeng, gemsfdtd, and milc, do we see a noticeable reduction of IPC for MemCTRL. Overall, the fault-aware MemCTRL and MemDIMM implementations see a 2.3% and 1.1% degradation, respectively, in IPC. However, the improvement in fault-tolerance with a fault-aware scheme makes it a good choice given a minimal performance overhead.

9.5.4 Comparison of different fault mitigation schemes

As the number of weak cells increases in newly fabricated memories as a result of increasingly smaller technology nodes, fault mitigation strategies will have to scale with the increased propensity for faults. In Figure 59, we compare fault-oblivious and fault-aware versions of ECC-1 and 2, FFE, PFE, ECP-3 and 12, and PFE+ECC-1 for weak cell rates ranging from 0.01% to 1%. The figure shows that PFE approaches the desired system UBER (10^{-14} [14]) at a weak cell rate of 0.01% while continuing to outperform other baseline correction schemes at higher weak cell rates (e.g., 1%). Hence, PFE shows propensity for being effective at moderate weak cell rates (e.g., 0.01%) while being extremely valuable for high weak cell rates (e.g., 1%).

For a 0.01% weak cell rate, ECC-1_{128} only achieves an UBER of 6.8×10^{-6} while ECC-2_{32} achieves an UBER of 1.3×10^{-6}. Moreover, the UBER drops dramatically as the number of errors increases, to 2.5×10^{-3} and 1.3×10^{-3}, respectively at a 1% weak cell incidence rate. For FFE, the UBER changes linearly with respect to the incidence rate. In other words, the percent of faults corrected by FFE remains invariant as the weak cell rate increases. As a

result, while FFE has a worse UBER at lower error rates than traditional correction schemes such as ECC, it becomes the more effective correction scheme at high error rates.

ECP_{FO}-3 and ECP_{FA}-3 achieve relatively good UBERs of 2.3×10^{-6} and 1.4×10^{-6}, respectively, at a 0.01% weak cell incidence rate. However, like ECC, ECP's fault tolerance drops off sharply as weak cell incidence rate increases. With a larger overhead, ECP_{FO}-12 and ECP_{FA}-12 are very effective for low weak cell incidence rates. With this higher overhead, ECP has a similar overhead to FFE and is slightly less effective at lower weak cell incidence rates (e.g., 0.01%) for similar reasons.

While for a 0.01% weak cell incidence rate, PFE_{FO} achieves an UBER close to that of ECP_{FA}-12, the effectiveness of PFE_{FO} increases at higher weak cell incidence rates. Moreover, the results show that adding ECC-1 to PFE_{FO} improves the UBER by an order of magnitude versus PFE_{FO} for different ranges of weak cell incidence rates. PFE_{FA} reaches an UBER that is less than 3×10^{-12} for a 0.01% weak cell incidence rate and is about five orders of magnitude more effective than PFE_{FO}+ECC-1. When adding ECC-1 on top of PFE_{FA}, the PFE scheme reaches the best fault mitigation efficiency (at least one order of magnitude) for 1% weak cell incidence rate against all fault mitigation schemes shown in the figure. PFE clearly provides the best fundamental protection against intra-row cross talk and when coupled with ECC-1 can achieve reasonable error rates even with extremely large numbers of weak cells.

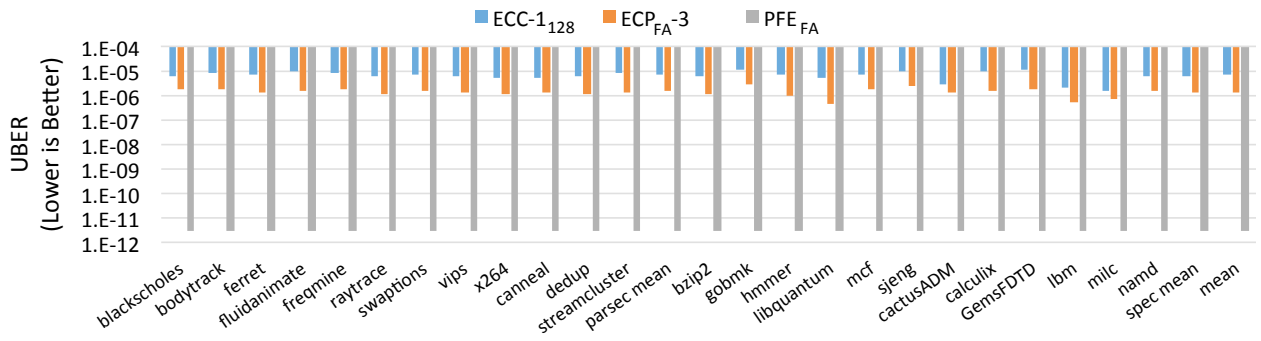
9.5.5 Sensitivity to block size

PFE is also effective for a variety of block sizes. Figure 60 shows the impact of varying the block size for PFE. For the fault-oblivious case, as the size of the block covered by PFE increases, the ability of bad pattern reduction decreases linearly. However, even for a 512-bit block with 1% weak cells, PFE reaches an UBER of 4.8×10^{-4}.

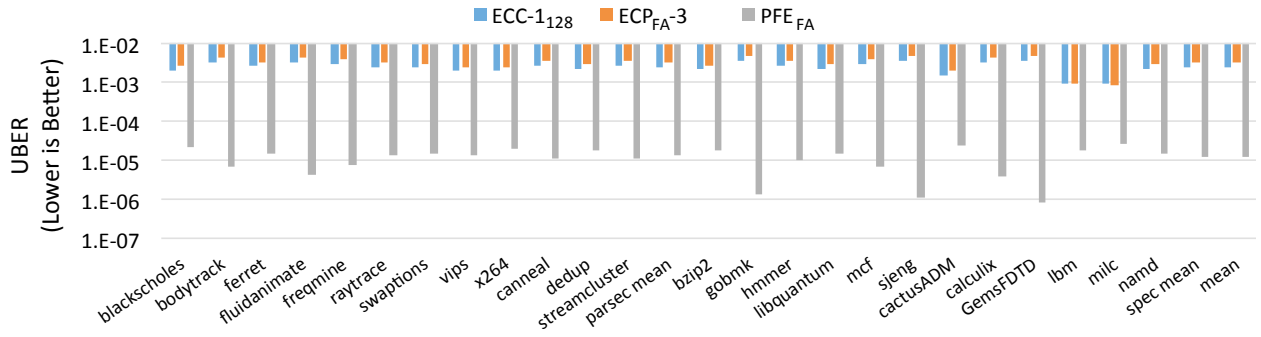
For the fault-aware approach, the UBER of PFE is at most 3×10^{-12} for low weak cell rates and 512-bit blocks. For a 1% weak cell rate, a 32-bit block size results in an UBER of 1.3×10^{-5}, but as the block size increases, the effectiveness drops, achieving only an UBER of 8.2×10^{-5} for 512-bit blocks.

9.6 Conclusion

We discussed how the presence of specific patterns of stored data exacerbates the likelihood of crosstalk occurrence in DRAM cells and triggers crosstalk faults. Reducing the number of bad patterns can decrease the occurrence of crosstalk incurred by process variation. We studied two orthogonal crosstalk mitigating techniques for DRAM cells. Experimental results conducted on PARSEC and SPEC benchmarks showed that the effects of crosstalk can be destructive, especially as the percentage of weak cells increases. The results, however, showed that the proposed PFE scheme is effective at avoiding the occurrence of crosstalk faults and, if combined with single error correcting ECC, may eliminate crosstalk faults completely when fewer than 0.01% of the cells are weak. Furthermore, combining PFE with ECC provides tolerance to other types of faults, such as transient faults, that are traditionally tolerated through ECC.



(a) 0.01% incidence of weak cells



(b) 1% incidence of weak cells

Figure 57: Comparison of PFE to other "low-overhead" (6.25%) fault-aware approaches.

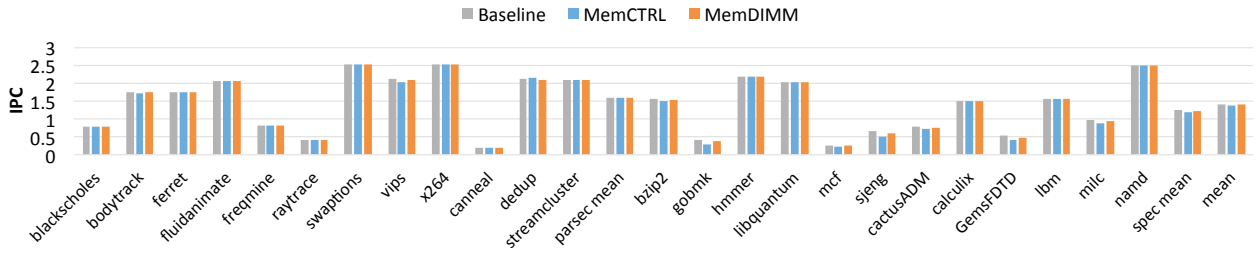


Figure 58: Performance impact of run-time determination of weak cells at the memory controller (MemCtrl) or within the memory DIMM (MemDIMM) compared to a fault-oblivious baseline that does not query the weak cell map.

Table 18: The overhead of different schemes with latency optimization and power optimization.

Scheme		512-bit block size					
		Latency Optimization			Power Optimization		
		Latency (ns)	Power (mW)	Area (μm^2)	Latency (ns)	Power (mW)	Area (μm^2)
Enc.	PFE_{FA}	0.66	9.90	32348.38	0.84	7.50	30223.86
	PFE_{FO}	1.65	20.46	84268.44	2.02	8.25	76139.23
Dec.	PFE_{FA}	0.17	2.57	4835.67	0.40	1.40	2905.91
	PFE_{FO}	0.17	2.57	4835.67	0.40	1.40	2905.91

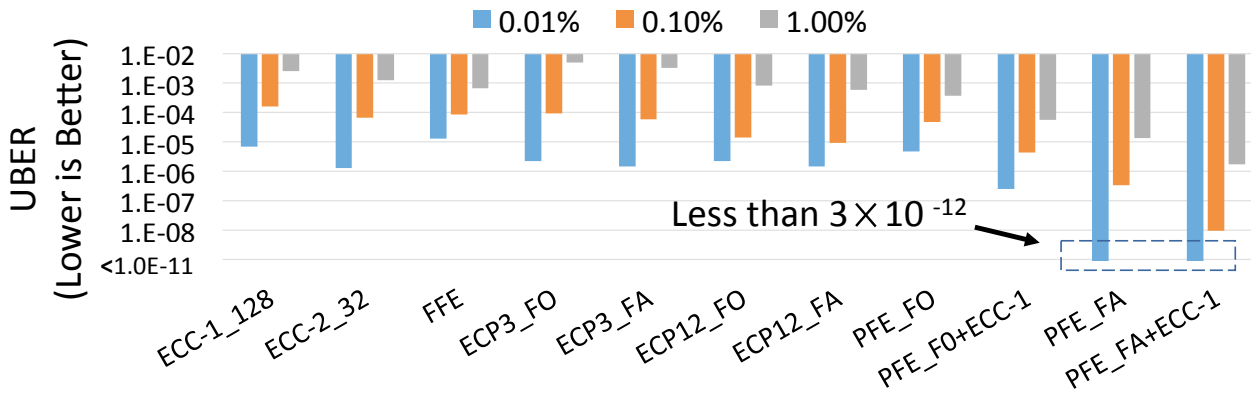


Figure 59: UBER for different fault mitigation schemes as weak cell incidence rate varies from 0.01% to 1%.

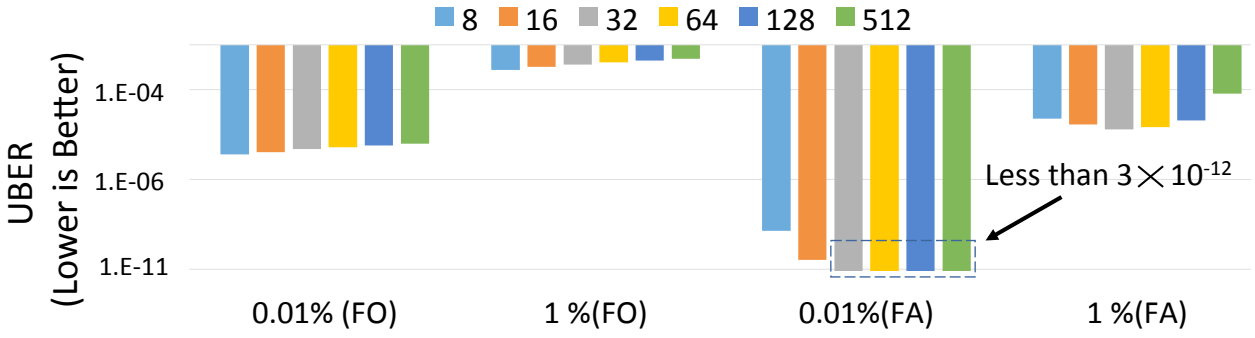


Figure 60: PFE for different block sizes, n.

10.0 Counter Advance

10.1 Design

Counter advance leverages the property that encryption of plaintext with a new counter generates a new random ciphertext. Recall that data that matches the stuck-at value is SA-R and data opposed to the stuck-at value is SA-W. When writing a ciphertext candidate in the presence of stuck-at faults, each faulty cell has a 50% probability of being SA-R, and similar for SA-W. Thus, by incrementing the counter, it is possible to improve fault tolerance by finding a ciphertext candidate that maximizes SA-Rs.

Consider the example in Figure 61 for a row with two stuck-at faults such that, given the ciphertext, the first is SA-R and the second is SA-W. Advancing the counter (`Counter+1`) resulted in the first fault becoming SA-W and the second becoming SA-R. This is due to the property that each fault in each ciphertext candidate has a 50% chance to be SA-R, but is equally likely to be SA-W. Advancing the counter again (`Counter+2`) was unlucky, resulting in two SA-Ws. The probability of finding an error free candidate with f faults is 2^{-f}, or 25% for f=2, which required multiple advancements in the example. Both word-level encryption and error correction can dramatically reduce the number of advancements to find an error free candidate. For example with single bit error correction (e.g., ECP-1), the example of Figure 61 would have been successful without counter advancement. If SECRET is used with independent sub-counters per block and assuming all blocks were dirty, blocks zero to two would have been written with `Counter` and block three would have used `Counter+1`.

We explore counter advance with block level encryption and error correction in Figure 62. Figure 62(a) expands on Figure 3 with sub-counters per 128-bit block in the style of SECRET [52]. Counter advance applied in this context examines the stuck-at bits independently between blocks and only advances the counter when SA-W bits appear. Stuck-at faults can be determined by storing and reading patterns of all '1's and '0's or using a fault cache [93]. It is straightforward to extend block-level encryption with ECC as the parity bits (e.g., SECDED (64,72) ECC) would not cross blocks. In this case, counter advance could

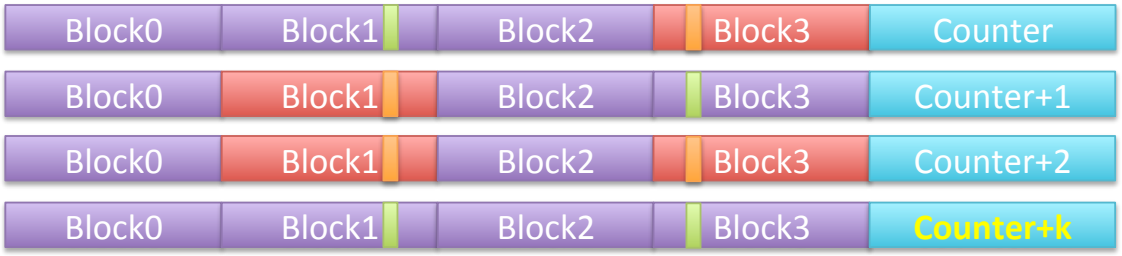


Figure 61: Counter advance example. Green indicates a SA-R fault and orange a SA-W fault. Purple blocks are error free and red blocks contain an error.

protect fewer SA-W errors and allow ECC protection to correct others at the cost of reduced transient error protection.

ECP, in contrast, uses pointers that are shared by the blocks of a given row. Block-level counter advance faces the trade off of using a pointer or advancing the counter to mitigate a fault. Using a pointer will reduce the availability of pointers to tolerate faults in other blocks. The algorithm for selecting the appropriate write candidate for ECP with counter advance is shown in Figure 62(b).

Assuming a counter advancement epoch window w where $w = 2^b$ and b is the number of bits for each sub-counter, w serves as a threshold of how many counter values will be explored to accomplish a particular write successfully. If the encrypted data experiences E errors (*i.e.,* SA-Ws) but E is less than a threshold T, the write proceeds with the current c value. Otherwise, if this is the "best" candidate (*i.e.,* fewest SA-Ws) so far it is retained. If c is still within the epoch w, c is incremented and the next candidate is evaluated. If c reaches the limit of the epoch without finding a candidate within the error threshold, the best candidate is written if sufficient ECP pointers are available, otherwise the write fails. In our evaluation we consider two schemes: the *counter minimization* (CM) approach sets T to the number of available ECP pointers, allowing a write to proceed with the minimum counter value that discovers a possible solution, while the *pointer minimization* (PM) approach sets T=1 requiring a fault free solution to write immediately.

10.2 Evaluation

To evaluate the effectiveness of counter advance we studied a 4GB main memory, with 64-bit words, and 512-bit rows organized in 4KB pages using eleven SPEC CPU 2006 benchmarks [71]: bzip2, cactus, gamess, gcc, gobmk, gromacs, leslie3d, mcf, namd, pearl, and zeusmp. Each workload was executed for at least 1 billion write accesses. Three bits were allocated as a sub-counter for each block, setting $w = 8$. High cell failure rates (10^{-3}, and 10^{-2}) representing different points during the memory lifetime, as shown in Figure 63, were used as stimuli for counter advance. To model the stuck-at faults, we created fault maps, including fault bits stuck at '0' or '1,' at these cell failure rates using Bayesian distribution to mimic the impact of process variation with spatial correlation of faults [21]. As process variation increases with scaling, the memory will incur cell faults more quickly and better error correction will be necessary to maintain effective lifetimes. For example, at a coefficient of variation (CoV) of 0.2, increasing fault tolerance to handle cell failures of 10^{-2} instead of 10^{-4} will extend the lifetime by 1.1×. For CoV of 0.25 and 0.3 effective lifetime can be extended by 3.8× and 16.4×, respectively, making operation in this failure range critical to reasonable memory lifetimes as scaling increases.

Figure 64 shows a summary of the uncorrectable bit error rate (UBER), defined as the number of bit errors that occur per bit written, for both row and block-level encryption with different strengths of error correction used to protect the data and counter bits. A "word-level" 64-bit block size was selected to match the word size in modern architectures. With no error correction, word-level counter advance (word CA) provides two orders of magnitude improvement in UBER compared to word-level encryption alone, such as SECRET [52], for as high as a 10^{-3} cell failure rate. As error correction is employed the improvement is amplified, providing 3–5 orders of magnitude improvement by introducing one ECP pointer (ECP1). At a cell failure rate of 10^{-3}, an UBER of $\leq 10^{-10}$ required only ECP4 for word and row CA. Employing PM reduced the requirement to ECP3. In contrast, SECRET with ECP6 can only achieve a 10^{-7} UBER.

At a cell failure rate of 10^{-2}, unsurprisingly, UBER is drastically reduced. For ECP6, the protection proposed by SECRET [52], counter advance achieves a 10^{-7} UBER versus 10^{-4}

for ECP6 alone. Relaxing counter advance to explore eight epochs allowed word CA with PM to function at a $<10^{-10}$ UBER with ECP6. This indicates that while a device might operate using the CM approach initially to minimize counter advancements, it could switch into PM and expand the searching window for *gracefully degraded* operation mode when the cell failure rate became sufficiently high.

Counter advance is sensitive to block size. Small block sizes will increase the flexibility to eliminate faults. Our block-size sensitivity study indicates counter advance is nearly as effective for 64-bit blocks as 32-bit blocks. 128-bit blocks have a noticeable degradation (0.5-1 orders of magnitude UBER), particularly with ECC and ECP, however, the counter advance improvements are still dramatic.

A logical concern about counter advance is the impact to performance from evaluating multiple ciphertext candidates and the potential to saturate the encryption counter more quickly. Figure 65 shows the number of counter increments per write operation. Word-level encryption naturally reduces the average counter advancements per write (A) to A=0.98 compared to the row-level baseline of A=1, as each write only advances the dirty words' sub-counters. This provides sufficient "room" for word-level fault-induced counter advancements for lower fault rates (*e.g.,* $\leq 10^{-4}$) without exceeding the row-level counter lifetime. At 10^{-3}, for ECP $\geq$3, A<1. At 10^{-2} there are significant fault-induced counter advancements, owing to gracefully degraded operation. However, increasing the number of epochs searched for larger numbers of ECP pointers, especially ECP6, provides significant improvements in protection, with only slight increases to A. To achieve an UBER of 10^{-10} only requires A=1.2 with ECP6 after a cell failure rate of 10^{-2}.

As this gracefully degraded mode would only occur very late in the memory lifetime, these counter advancements would only saturate the counter nominally sooner while extending the usable life dramatically. If the system is reset with a new encryption key or the data is moved for another reason (e.g., wear-leveling [102]), the counter can also be reset. Moreover, given that writing is not typically on the performance critical path, our experiments indicate that these additional encryptions (A=1.2) do not significantly degrade performance.

10.3 Alternative Implementations

Counter-mode encryption has a downside that it requires the storage of a counter for each row. Unfortunately, this overhead cannot be eliminated for counter-mode encryption. However, the storage dedicated to the per-word sub-counters, initially proposed by SECRET [52] to reduce energy and improve endurance, could be retargeted to improve fault tolerance. This storage would be insufficient to add additional ECC, but could add two additional ECP pointers. This comparison is shown in Figures 64 and 65 by comparing word CA with ECP-N to row CA with ECP-N+2. The results indicate that for lower fault rates, row CA would provide an advantage in fault tolerance at the cost of increased energy and reduced endurance. As the fault rate increases, word CA in PM mode is more fault tolerant while maintaining energy and endurance benefits over row CA.

NIST has proposed to use AES XTS as a standard for disk encryption [103]. While there are feasibility challenges to applying XTS in memory while guaranteeing protection, XTS would eliminate the need for counter storage in memory. XTS is based on XOR-encrypt-XOR (XEX) [104]. In disks, XEX/XTS encrypts twice, utilizing one key/encryption based on the sector and a second for each block within the sector. By extending the second encryption with an additional parameter supplied by the sub-counter, multiple candidates can be generated per block to improve fault tolerance. Because the ciphertext candidates for both XTS and counter-modes of AES have the same random properties, the results we obtained from XTS encryption are the same as those reported in Figures 64 and 65.

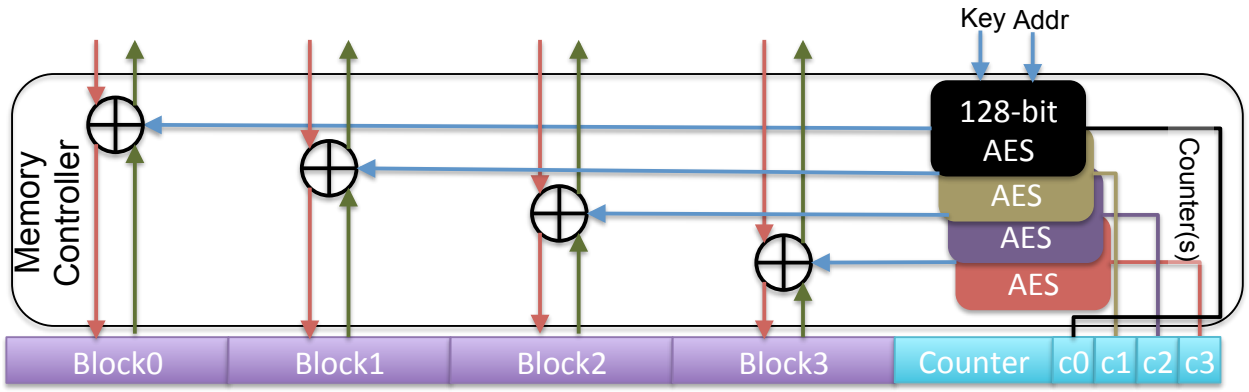
10.4 Related Work

SECRET is the current state of the art in energy reduction and fault tolerance for encrypted PCM memory. However, SECRET improves on prior proposals such as DEUCE, or Dual Counter Encryption, which saves energy by distinguishing between dirty and clean words and encrypting on the dirty words within the epoch. DEUCE does not consider reliability and SECRET provides significant savings over DEUCE at the cost of additional

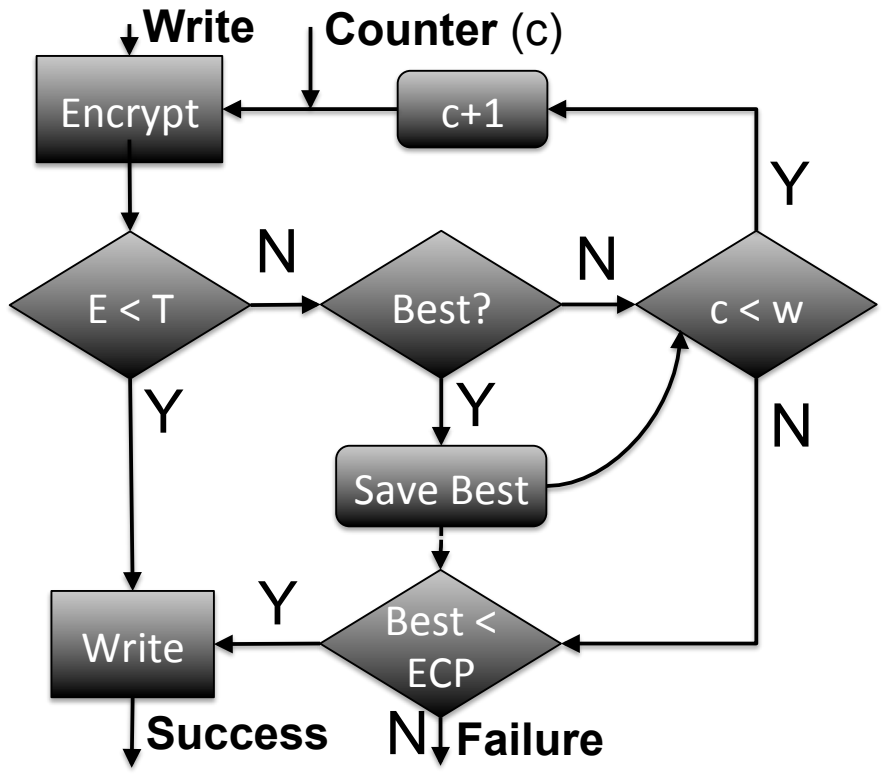
sub-counter bits [52]. A specifically fault-tolerant proposal is to use the increasing counter value to serve as an indicator of PCM memory cell age and using this information to adopt increasingly capable ECC to combat memory faults [105]. Counter advance is complementary to this approach allowing use of lower overhead ECC for a longer duration, or achieving a particular UBER with a lower overall ECC storage. There has also been recent work to collaboratively design wearleveling with counter-mode encryption. The counter storage size is reduced by resetting the counter when the data is moved to a new location from the wear-leveling resulting in improved overhead and latency [102]. Again counter advance is complementary as the number of counter advancements to maintain reliability would require minimal impact to the total counter advancement, which can retain the storage savings of this wear-leveling approach.

10.5 Conclusion

Counter advance leverages the nature of block cipher encryption to improve reliability of systems that use in memory encryption for memory with endurance faults that manifest as stuck-at values. Counter advance in the presence of SA-Ws generates additional write candidates to maximize SA-Rs just by advancing the counter. Counter advance is compatible with row or word-level writes and provides multiplicative improvements in UBER compared to error correction alone. Counter advance can achieve the same protection as strong error correction (e.g., ECC or ECP5) with far fewer pointers (e.g., ECP1) at moderate error rates. It can also maintain an UBER of 10^{-10} with the same error correction as the leading related work [52] for extremely high fault rates of 10^{-2}. This can lead to lifetimes being extended by 2-10$\times$ or more depending on severity of process variation. We plan to explore the performance impact, lifetime improvement, and examine other modes of encryption in detail in our future work.



(a) Encrypting individual blocks with sub-counters.



(b) ECP encoding flow.

Figure 62: Block level counter advance architecture.

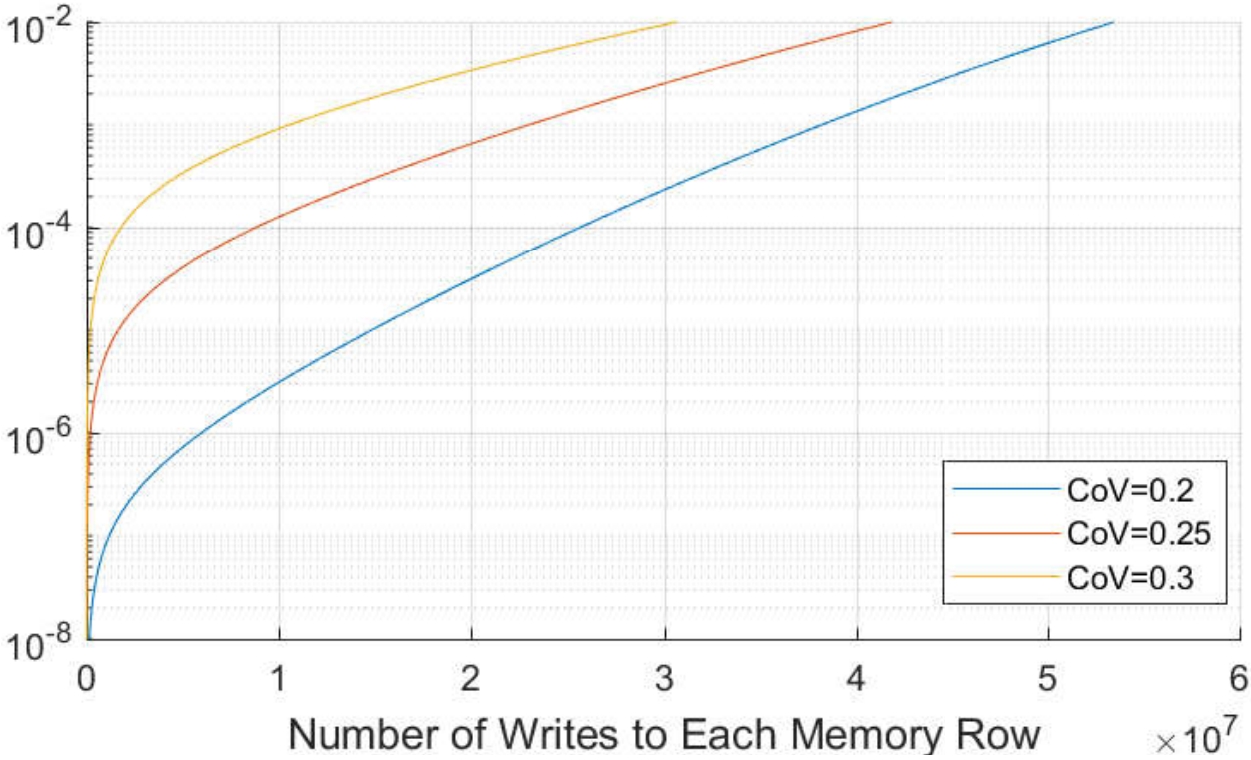


Figure 63: Cell fault rate for different coefficients of variation.

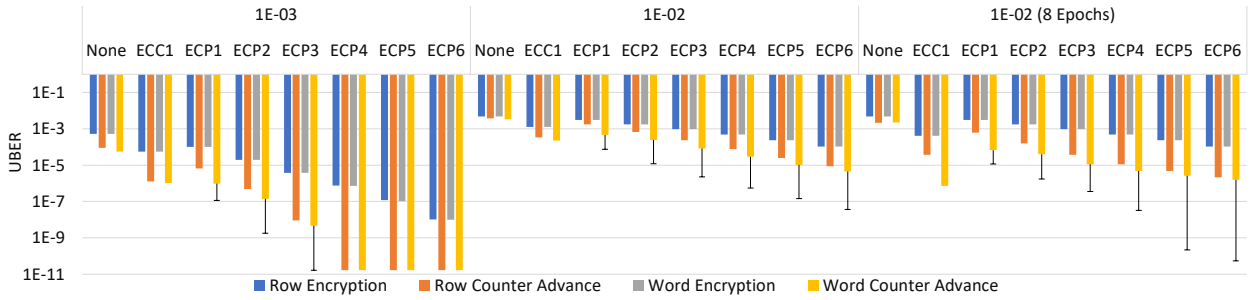


Figure 64: UBER for various error rates. Counter advance explores one epoch (w=8), except where noted. Word CA+ECP is reported for CM with an error bar indicating PM.

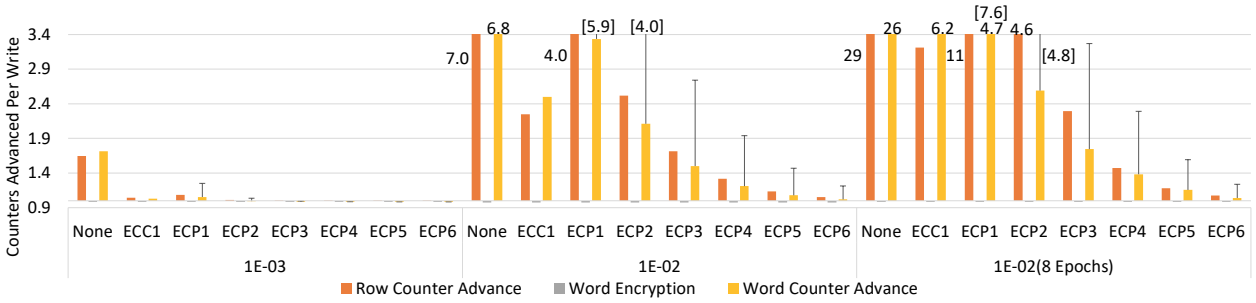


Figure 65: Counters advanced per write at various error rates. Counter advance explores one epoch (w=8) except where noted. Word CA+ECP is reported for CM with an error bar indicating PM. Row Encryption is always unity.

11.0 FaME

11.1 FaME Error correction

In Chapter 7.2, we discussed how FLOWER can enable and enhance previously developed fault-tolerance techniques. Most importantly, however, FLOWER enables *new* fault-tolerance techniques which otherwise would not be possible. In this section, we propose a new technique for correcting stuck-at faults in PCM: **Fa**ult **M**ap **E**nabled Error Correction.

11.2 FaME Design

In FaME, f bits are added to each memory row to provide protection against f faults. Instead of requiring f pointers, FaME utilizes the FLOWER fault vector, utilizing spare bits for each reported fault location, *in order*. A comparison of the auxiliary bit overheads to protect three faults for ECP, YODA, and FaME are shown in Figure 66. Traditional ECP requires 31 ECP bits per 512-bit row to protect three faults: 3×9 pointer bits, three spare bits, and one flag bit signifying all pointers are used. ECP does not require knowledge of SA-R or SA-W. YODA requires only 11-bits to correct three faults: one ECP pointer, or nine pointer bits, one flag bit, and one inversion bit (see Section 7.2.2). YODA also requires knowledge of SA-'1' and SA-'0' fault locations. FaME requires only three auxiliary bits to protect against three faults[1]. FaME requires knowledge of fault *locations*, but does not distinguish between SA-'1' or SA-'0.'

Because FaME only requires f bits per row to correct f faults per row, there are several substantial advantages over previous fault tolerance implementations, such as ECP. The primary advantage is that FaME combined with FLOWER can achieve a substantial area savings over ECP. For example, if a 3.13% area overhead FLOWER fault map is used, six

[1]FaME works properly if FLOWER reports false positives, but each false positive will consume a spare bit and limit the fault tolerance capability of FaME.

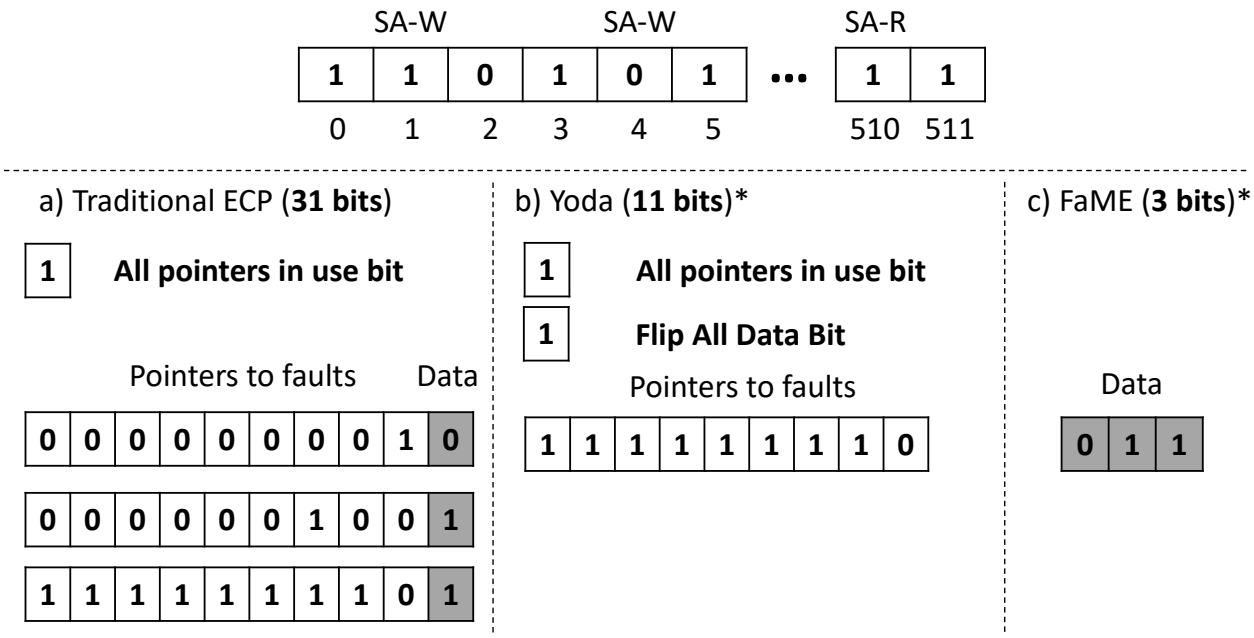


Figure 66: Auxiliary bit example for a 512-bit PCM row for a) traditional ECP, b) Yoda, and c) FaME. SA-W and SA-R refer to stuck-at-wrong and stuck-at right cells. FaME and Yoda overheads do not include the fault map.

fault protection per row requires an additional area overhead of 6512 or 1.2% for a total overhead of 4.3%. In contrast ECP requires 61-bits per row for a 61512 or a nearly 12% overhead. Even if the nominal lost fault tolerance from false positives is accounted for, FaME still provides substantial benefits in area with iso-tolerance, fault tolerance with iso-area, or both improved fault tolerance and area.

Faults in the auxiliary bits are protected in same way data faults are represented, by extending the fault vectors with auxiliary bits in the fault map. Thus, FaME can discover if an auxiliary bit has a stuck-at fault. In this case, the auxiliary bit identified as faulty will be skipped, such that FaME with f auxiliary bits protects against f faults reported by FLOWER across both the data and auxiliary bits.

11.3 Accumulated Faults

FaME's dependence on fault vectors for its encoding and decoding operations allows it to use extremely small auxiliary bit overheads. This dependence causes complications in systems where faults accrue over time. Before updating a row's partial fault vectors, as discussed in Section 7.1.2.3, data rows whose fault vectors will be affected by adding the new fault must have their auxiliary bits updated, even if the new fault is not a true fault (*e.g.,* a false positive).

MinCI with multiple arrays allows some simplifications, not possible with Murmur hashes, to allow discovery of all the rows which might require updates. For example, consider an 8-bit address (n=8), with a d=2 (2D) fault map that requires a 25% overhead (k=5). Assume address A=[a_0,a_1, ...a_7] has a hash function $H_j(A) \rightarrow$ [h_{j0},h_{j1},...h_{j4}]. Due to the properties of MinCI, $H_j(A)$ essentially strips $n-k$ bits—in this case arbitrarily bits at positions 1, 4, and 6—to form a vector [a_0,a_2,a_3,a_5,a_7]. Assuming a vector U that contains all possible permutations of $n-k$ bits, vector B_{Aj} can be created from $H_j(A)$ and U, such that B_{Aj}= [a_0,u_0,a_2, a_3,u_1,a_5,u_2,a_7]. Rows other than $u_0 = a_1$, $u_1 = a_4$, and $u_2 = a_6$ are the "conflicting rows" from the hash. Determining these conflicting rows for all d hashes produces the pool of memory rows that may require updates to their FaME bits. The size of the pool is $d2^{n-k}$.

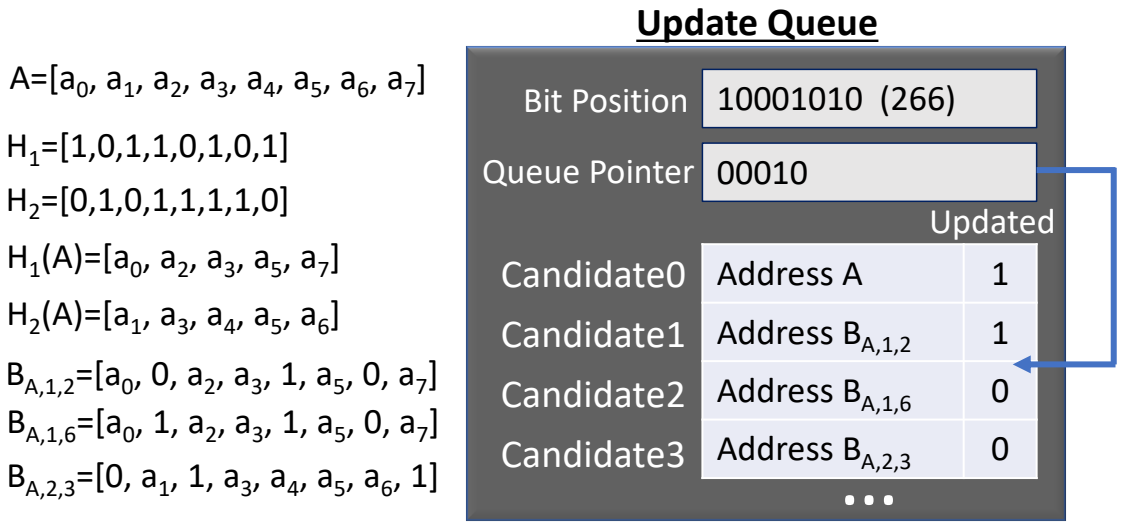


Figure 67: FaME update cache.

However, the number of writes can be substantially reduced after checking the current partial fault vectors for a potential candidate. Address B_{Aj} only requires updates if, for the new fault position p and for all j, bit p of the partial fault vector addressed at $H_j(B_{Aj})$ is '1' or $H_j(B_{Aj})$=H_j(A).

11.3.1 Memory Update Queue

To avoid memory stalls, any rows which satisfy the aforementioned conditions are added to an update queue shown in Figure 67, while memory accesses continue. The queue stores the bit position to update in the fault map and a list of update candidates. Each update candidate entry stores the address and a flag bit to indicate whether the address is updated (re-written with the corrected auxiliary bits) in memory. If an address in the update queue is accessed and the auxiliary bits have been updated in memory, the fault vector from the fault map is updated with the stored bit position. Otherwise, the access proceeds using the unmodified fault vector. Once all the candidates in the update queue have been written, the fault vector for row A is updated with the procedure in Section 7.1.2.3 and the queue is flushed. In the rare case a second fault is discovered, for example in row C, prior to

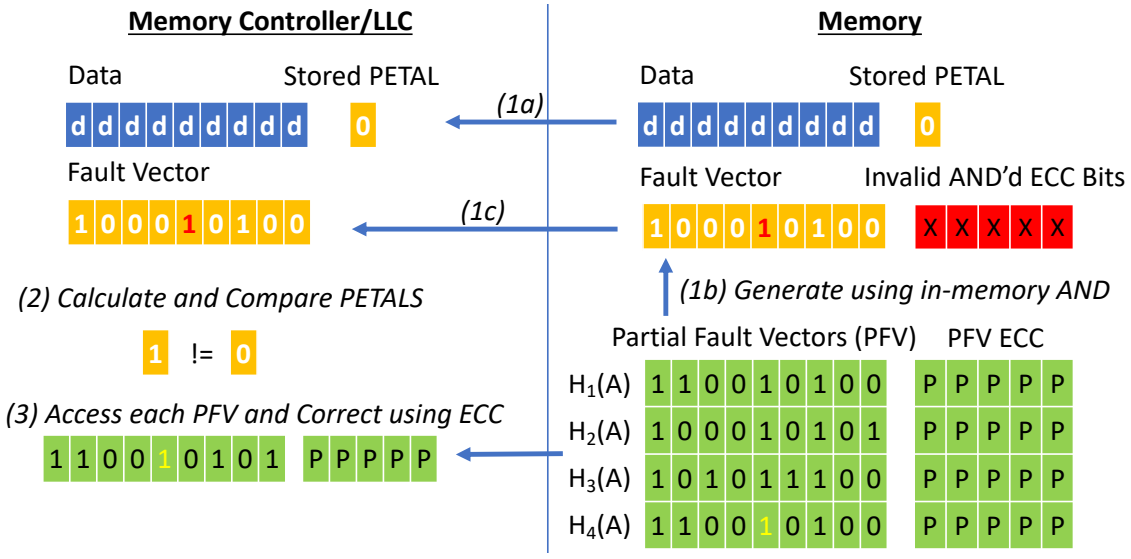


Figure 68: (1) Data/FLOWER access, (2) PETAL comparison, and (3) Error correction on petal detection.

completing the first fault update, the write to row C stalls until the update to row A is complete.

11.3.2 PETALs: In-Memory FLOWER Correction

Revisiting error detection and correction for FLOWER in the context of protecting the fault map against transient faults (such as errors during in-memory `AND` operations), the FaME update queue can also be used to support a fault tolerance structure for the in-memory version of FLOWER. We add a FLOWER PETAL (Parity Enabling Tolerance for Accelerating Logic-in-memory) bit to each memory row (not just memory rows storing FLOWER) requiring a 1512 or 0.2% overhead, as shown in Figure 68. The PETAL bit stores the parity of the number of faults for the *corresponding FLOWER fault vector* to that row. If there is a fault accessing a *partial* fault vector, it *may* appear as a false positive ('0' read as '1' where all other hashes are '1') or a false negative ('1' read as '0' and all other hashes are '1'). The PETAL bit will detect one fewer or one additional reported fault, which triggers individual access of the partial fault vectors. These vectors can be corrected using their ECC

Table 19: UBER for in-memory combination protected with PETALs versus memory controller combination protected with ECC-1 for a 4D FLOWER fault map.

	10^{-10}	10^{-8}	10^{-6}	10^{-4}
PETAL	$2{\cdot}10^{-19}$	$4{\cdot}10^{-16}$	$4{\cdot}10^{-12}$	$4{\cdot}10^{-8}$
ECC-1	$6{\cdot}10^{-19}$	$6{\cdot}10^{-15}$	$6{\cdot}10^{-11}$	$6{\cdot}10^{-7}$

at the memory controller. Note that the in-memory `AND` result of the ECC bits (shown as 'X's in red) is unusable [89] and discarded. Note, multiple rows will require updates to the PETAL bit when a new fault is added to FLOWER. This update follows the same process used to identify the rows requiring FaME bit updates as described above.

The UBER of using PETALs to protect in-memory partial fault vector combination is shown in Table 19 and compared with using SECDED ECC at the memory controller across different initial fault rates. Our goal was for PETALs to match ECC-1 when combining fault vectors at the memory controller combination. PETALs provide slightly higher error protection making the in-memory access approximately as safe as ECC at the memory controller.

11.4 FaME Lifetime Improvement for PCM

To determine the impact of YODA+FLOWER on the lifetime of PCM memory we conducted experiments on memory traces to a 1MB memory segment (16K 64-byte rows). Our custom simulator determines the expected lifetimes of each cell in the memory based on a probabilistically determined failure map. The failure map is computed from the mean lifetime (10^8 writes) and coefficient of variance (CoV) according to a normal probability distribution with spatial correlation of faults [21, 82]. The memory traces averaged over nine SPEC benchmarks (SUITE), synthetic traces simulating thrashing between two memory rows (THRASH), and perfectly uniform wear-leveling (LEVEL) were run in the simulator

for 20 different failure maps[2] at CoV=0.2,0.3. During each experiment, once the number of writes to a cell exceeds its lifetime, dictated in that particular failure map, it is considered stuck-at its last recorded value. Once two rows in the 1MB memory segment cannot be corrected, we consider the memory failed at its end of life.

Figure 69 shows the lifetime results for iso-area (12.5%) fault tolerance area overhead for different correction schemes with and without FLOWER.As expected, because of the high CoV, the system fails much earlier than the mean cell failure rate of 10^8 writes. For the SPEC benchmarks, on average, "No Correction" fails before the 10^6th write to the 1MB memory segment for a 0.3 CoV. ECC1 and ECP6 improve this correction substantially, to close to 10^8 writes to the memory segment before failure.

Assuming a CoV of 0.2 and dedicating 1.56% area overhead for the fault map(s), YODA-5 extends the lifetime compared to ECP6 by 4×. With the same encoding overhead, FaME can provide 51 bit spares and extends the lifetime by 17× and 14× over ECP6 for CoV of 0.2 and 0.3, respectively. Comparing to larger fault maps (6.25%) at the expense of fewer auxiliary bits, we see an interesting behavior. For SUITE, YODA achieves a longer lifetime with more pointers and smaller fault maps (1.56%), while FaME achieves a longer lifetime with larger maps (6.25%) and fewer auxiliary bits. This is because FaME benefits from the drastically reduced false positives, which more than make up for the 20 fewer spare bits. For LEVEL at 0.2 CoV, FaME31 with the larger FLOWER fault map improves lifetime versus no correction by 2,895× and 28× compared to ECP6. This improvement can be treated as an upper bound of the lifetime improvement for the system. For THRASH, the benefit is, unsurprisingly, tempered—YODA and FaME improve over ECP6 by only up to 1.8× and 3.4×, respectively at CoV=0.3. However, as many systems aim to implement improved wear leveling strategies, this THRASH (worst-case) improvement can typically be avoided, and a behavior between SUITE and possibly close to LEVEL is the more likely scenario. It is also important to note that the trends bear out that more overhead dedicated to the fault map has a better impact on maintaining fault rates with higher CoV values. For both YODA and

[2]20 maps were deemed to have sufficient coverage as the difference in results after including more than 10 maps was insignificant.

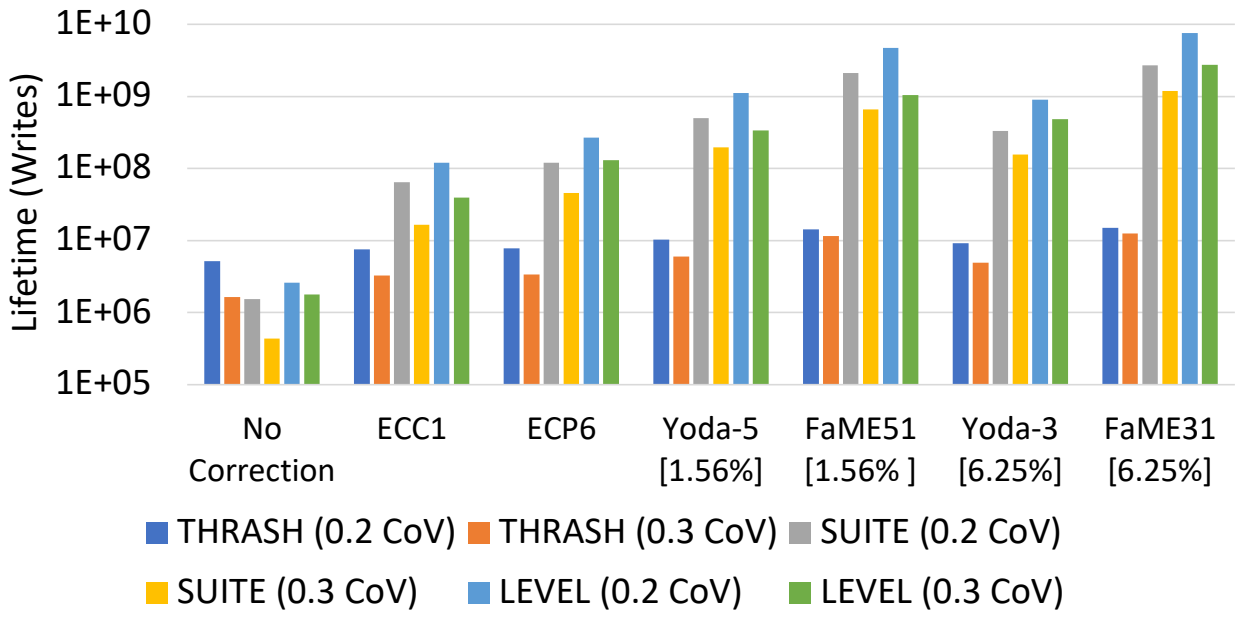


Figure 69: Lifetime to 1MB memory for 12.5% area budget (iso-area). FLOWER overheads shown in "[]."

FaME, the difference between CoV of 0.2 and 0.3 is tempered when more space is dedicated to the map.

11.5 Conclusion

We present FaME, a new fault tolerance scheme for PCM endurance fault mitigation. The PETAL bits scheme provides novel protection for in-memory operations to enhance their reliability in this context. FLOWER combined with FaME provides a 17× improvement in fault tolerance compared to ECP6 at the same area overhead for SPEC workloads. Compared to Archshield at 10^{-4}, a fault rate supported by physical studies of crosstalk in 5-year-old DRAM [17], FLOWER+FaME can provide equal memory protection while reducing the area overhead by 20% and when scaling to higher (*e.g.,* 10^{-2}) fault rates, this overhead advantage

exceeds 7×. Further, we extend PCM lifetime using FLOWER+FaME fault tolerance, which will reach 10^{-4} to 10^{-2} faults after continued use.

12.0 Overall Conclusions

In this dissertation, I provided a variety of potential solutions to issues arising from the scaling of memory technology. First, I demonstrated new techniques I developed for holistic sustainability analysis, necessary at shrinking dimensions due to the non-negligible impact of manufacturing. This including an architecture-level flow (GreenChip) and a lower-level circuit/CAD tool (GreenASIC). I also developed a technique to utilize these tools while also considering reliability, with LARS.

Secondly, in order to enable low-overhead extremely effective correction schemes, I created several bit-level fault maps. SFaultMap is a low-overhead solution for static faults or weaknesses in a system. FLOWER greatly improved on this with a structure which can easily add faults over time, while also having better performance and built-in fault map reliability through its PETAL bits. Separately, HOTH was developed specifically for harsh environments, for tracking both permanently failed cells and cells weak to radiation.

Finally, I developed extremely low-overhead encoding techniques for fault avoidance, several of which were directly enabled by the development of the bit-level fault maps. I developed PFE, a technique for avoiding bitline crosstalk faults which provides substantial reliability improvements when the fault locations are known. I also developed FaME, which can correct n faults with n bits and is directly enabled by bit-level maps. Lastly, counter advance requires no area overhead, and leverages the inherent randomness of encryption to enhance the reliability of encrypted Phase Change Memories.

Bibliography

[1] Apple Inc., "Environmental Report." [Available Online]: `http://www.apple.com/environment/reports/`, 2015.

[2] S. B. Boyd, *Life-Cycle Assessment of Semiconductors*. Springer, 2012.

[3] C. F. Murphy, G. A. Kenig, D. T. Allen, J.-P. Laurent, and D. E. Dyer, "Development of Parametric Material, Energy, and Emission Inventories for Wafer Fabrication in the Semiconductor Industry," *Env. Sci. & Tech.*, Vol. 37, No. 23, No. 23, 2003.

[4] D. Kline, N. Parshook, X. Ge, E. Brunvand, R. Melhem, P. K. Chrysanthis, and A. K. Jones, "Holistically evaluating the environmental impacts in modern computing systems," *Green and Sustainable Computing Conference (IGSC), 2016 Seventh International*, pp. 1–8, IEEE, 2016. Best Paper Award Winner.

[5] D. Kline Jr, N. Parshook, X. Ge, E. Brunvand, R. Melhem, P. K. Chrysanthis, and A. K. Jones, "GreenChip: A tool for evaluating holistic sustainability of modern computing systems," *Sustainable Computing: Informatics and Systems*, 2017.

[6] M. J. Scott and E. K. Antonsson, "Using indifference points in engineering decisions," *Proc. of ASME Des. Eng. Tech. Confs.*, 2000.

[7] D. Kline, N. Parshook, A. Johnson, J. E. Stine, W. Stanchina, E. Brunvand, and A. K. Jones, "Sustainable IC design and fabrication," *Green and Sustainable Computing Conference (IGSC), 2017 Eighth International*, pp. 1–8, IEEE, 2017.

[8] E. Brunvand, D. Kline Jr, and A. K. Jones, "Dark Silicon Considered Harmful: A Case for Truly Green Computing," *Green and Sustainable Computing Conference (IGSC), 2018 Seventh International*, IEEE, 2018. Best Paper Award Winner.

[9] D. Kline, R. Melhem, and A. K. Jones, "Sustainable fault management and error correction for next-generation main memories," *Green and Sustainable Computing Conference (IGSC), 2017 Eighth International*, pp. 1–6, IEEE, 2017.

[10] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *ISCA*, pp. 141–152, 2010.

[11] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates," *ISCA*, 2013.

[12] S. M. Seyedzadeh, D. Kline Jr, A. K. Jones, and R. Melhem, "Mitigating bitline crosstalk noise in dram memories," *Proceedings of the International Symposium on Memory Systems*, pp. 205–216, ACM, 2017.

[13] D. Kline Jr, R. G. Melhem, and A. K. Jones, "Counter Advance for Reliable Encryption in Phase Change Memory," *IEEE Computer Architecture Letters*, 2018.

[14] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *ISCA*, 2014.

[15] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *Black Hat*, Vol. 15, 2015.

[16] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *IEEE Computer Architecture Letters*.

[17] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," *DSN*, 2016.

[18] Y. Konishi, M. Kumanoya, H. Yamasaki, K. Dosaka, and T. Yoshihara, "Analysis of coupling noise between adjacent bit lines in megabit DRAMs," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 1, No. 1, pp. 35–42, 1989.

[19] Z. Yang and S. Mourad, "Crosstalk induced fault analysis and test in DRAMs," *Journal of Electronic Testing*, Vol. 22, pp. 173–187, 2006.

[20] T. Yoshihara, "A twisted bit line technique for multi-Mb DRAM's," *ISSCC Dig. Tech. Papers*, pp. 238–239, 1988.

[21] Z. Al Ars, *DRAM fault analysis and test generation*. TU Delft, Delft University of Technology, 2005.

[22] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM design and organization for energy-constrained multi-cores," *SIGARCH Comput. Archit. News 2010*.

[23] S. Khan, C. Wilkerson, D. Lee, A. R. Alameldeen, and O. Mutlu, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *in CAL 2016*.

[24] J. Kim and M. C. Papaefthymiou, "Block-based multiperiod dynamic memory design for low data-retention power," *TVLSI*, Vol. 11, No. 6, No. 6, pp. 1006–1018, 2003.

[25] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," *ACM SIGARCH Computer Architecture News*, Vol. 40, pp. 1–12, IEEE Computer Society, 2012.

[26] 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," *ACM SIGARCH Computer Architecture News*, Vol. 38, pp. 83–93, ACM, 2010.

[27] P. G. Emma, W. R. Reohr, and M. Meterelliyoz, "Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications," *IEEE micro*, Vol. 28, No. 6, No. 6, 2008.

[28] C.-H. Lin, D.-Y. Shen, Y.-J. Chen, C.-L. Yang, and M. Wang, "SECRET: Selective error correction for refresh energy reduction in DRAMs," *ICCD*, pp. 67–74, IEEE, 2012.

[29] Intel and Micron, "3D XPoint Technology," [Available Online]: https://www.micron.com/about/our-innovation/3d-xpoint-technology, 2015.

[30] C. J. Xue, G. Sun, Y. Zhang, J. J. Yang, Y. Chen, and H. Li, "Emerging non-volatile memories: opportunities and challenges," *CODES+ISSS*, pp. 325–334, IEEE, 2011.

[31] R. Wang, L. Jiang, Y. Zhang, and J. Yang, "SD-PCM: Constructing reliable super dense phase change memory under write disturbance," *ACM SIGPLAN Notices*, Vol. 50, No. 4, pp. 19–31, 2015.

[32] Y. H. Son, S. Lee, O. Seongil, S. Kwon, N. S. Kim, and J. H. Ahn, "CiDRA: A cache-inspired DRAM resilience architecture," *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 502–513, IEEE, 2015.

[33] ISO, "Environmental management – Life cycle assessment – Requirements and guidelines," Tech. Rep. 14044, 2006.

[34] UNEP/SETAC, "Life Cycle Approaches: The road from analysis to practice," Tech. Rep., 2005.

[35] "U.S. Life Cycle Inventory Database," 2012.

[36] J. Bare, "TRACI 2.0: the tool for the reduction and assessment of chemical and other environmental impacts 2.0," *Clean Technologies and Environmental Policy*, Vol. 13, No. 5, No. 5, 2011.

[37] A. Jones, L. Liao, W. Collinge, H. Xu, L. Schaefer, A. Landis, and M. Bilec, "Green computing: A life cycle perspective," *IGCC*, 2013.

[38] A. Jones, Y. Chen, W. Collinge, H. Xu, L. Schaefer, A. Landis, and M. Bilec, "Considering fabrication in sustainable computing," *ICCAD*, 2013.

[39] M. A. Yao, T. G. Higgs, M. J. Cullen, S. Stewart, and T. A. Brady, "Comparative assessment of life cycle assessment methods used for personal computers.," *Env. Sci. & Tech.*, Vol. 44, No. 19, No. 19, 2010.

[40] P. Teehan and M. Kandlikar, "Comparing Embodied Greenhouse Gas Emissions of Modern Computing and Electronics Products," *Environmental Science & Technology*, Vol. 47, No. 9, No. 9, 2013.

[41] SIA, "Model for Assessment of CMOS Technologies and Roadmaps," 2007.

[42] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," *MICRO*, 2009.

[43] M. Neisser and S. Wurm, "ITRS lithography roadmap: 2015 challenges," *Advanced Optical Technologies*, Vol. 4, No. 4, No. 4, 2015.

[44] R. I. Bahar, A. K. Jones, S. Katkoori, P. H. Madden, D. Marculescu, and I. L. Markov, "Workshops on Extreme Scale Design Automation," Tech. Rep., CCC, 2014.

[45] P. Teehan, "LCA Studies of Tablets; Embodied CO2 of Tablets; Comparison with Similar Products," *Green Electronics Council Slates/Tablets Workshop*, 2013.

[46] A. K. Jones, "Design for Sustainability," *Emerging Green*, Green Electronics Council, 2015.

[47] E. D. Williams, R. U. Ayres, and M. Heller, "The 1.7 Kilogram Microchip: Energy and Material Use in the Production of Semiconductor Devices," *Environmental Science & Technology*, Vol. 36, No. 24, No. 24, 2002. PMID: 12521182.

[48] M. A. Yao, A. R. Wilson, T. J. McManus, and F. Shadman, "Comparative analysis of the manufacturing and consumer use phases of two generations of semiconductors [microprocessors]," *Electronics and the Environment, 2004. Conference Record. 2004 IEEE International Symposium on*, 2004.

[49] J. von Geibler, M. Ritthoff, and M. Kuhndt, "The environmental impacts of mobile computing: A case study with HP," Tech. Rep. IST-2000-28606, Digital Europe, Wuppertal Institute, 2003.

[50] W. Diffie and M. E. Hellman, "Privacy and authentication: An introduction to cryptography," *Proceedings of the IEEE*, Vol. 67, No. 3, pp. 397–427, March 1979.

[51] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," *ISCA*, pp. 179–190, 2006.

[52] S. Swami, J. Rakshit, and K. Mohanram, "SECRET: Smartly EnCRypted Energy EfficienT Non-Volatile Memories," *DAC*, 2016.

[53] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, Vol. 13, No. 7, No. 7, pp. 422–426, 1970.

[54] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He, "BitFunnel: Revisiting signatures for search," *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 605–614, ACM, 2017.

[55] Y. Heo, X.-L. Wu, D. Chen, J. Ma, and W.-M. Hwu, "BLESS: bloom filter-based error correction solution for high-throughput sequencing reads," *Bioinformatics*, Vol. 30, No. 10, No. 10, pp. 1354–1362, 2014.

[56] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *ACM SIGCOMM Computer Communication Review*, Vol. 35, No. 4, No. 4, pp. 181–192, 2005.

[57] F. Putze, P. Sanders, and J. Singler, "Cache-, hash-and space-efficient bloom filters," *Experimental Algorithms*, pp. 108–121, 2007.

[58] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON)*, Vol. 8, No. 3, No. 3, pp. 281–293, 2000.

[59] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 25–36, ACM, 2006.

[60] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok, "Don't thrash: how to cache your hash on flash," *Proceedings of the VLDB Endowment*, Vol. 5, No. 11, No. 11, pp. 1627–1637, 2012.

[61] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75–88, ACM, 2014.

[62] P. K. Pearson, "Fast hashing of variable-length text strings," *Communications of the ACM*, Vol. 33, No. 6, No. 6, pp. 677–680, 1990.

[63] A. Appleby, "Murmurhash 2.0," 2008.

[64] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, Vol. 11, No. 2007, No. 2007, p. 21, 2007.

[65] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 52, ACM, 2011.

[66] N. Binkert, B. Beckmann, *et al.*, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, Vol. 39, No. 2, No. 2, 2011.

[67] N. P. Jouppi and S. J. Wilton, "An enhanced access and cycle time model for on-chip caches," Tech. Rep. TR-93-5, Compaq, 1994.

[68] U.S. Energy Information Administration (EIA), "International Energy Statistics," [Accessed May 2016].

[69] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Comp. Arch. Let.*, Vol. 10, No. 1, No. 1, 2011.

[70] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," *PACT*, 2008.

[71] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, Vol. 34, No. 4, No. 4, 2006.

[72] J. L. Henning, "SPEC CPU2006 Memory Footprint," *SIGARCH Comput. Archit. News*, Vol. 35, No. 1, No. 1, 2007.

[73] S. M. Pieper, J. M. Paul, and M. J. Schulte, "A New Era of Performance Evaluation," *IEEE Computer*, Vol. 40, No. 9, No. 9, pp. 23–30, 2007.

[74] J. Suckling and J. Lee, "Redefining scope: the true environmental impact of smartphones?," *Intl. Jour. of Life Cycle Assess.*, Vol. 20, No. 8, No. 8, 2015.

[75] M. S. Müller *et al.*, "SPEC OMP2012 — An Application Benchmark Suite for Parallel Systems Using OpenMP," *IWOMP*, 2012.

[76] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, *et al.*, "The NAS parallel benchmarks," *The International Journal of Supercomputing Applications*, Vol. 5, No. 3, No. 3, pp. 63–73, 1991.

[77] J. Stevens, P. Tschirhart, M.-T. Chang, I. Bhati, P. Enns, J. Greensky, Z. Chisti, S.-L. Lu, and B. Jacob, "An integrated simulation infrastructure for the entire memory hierarchy: Cache, dram, nonvolatile memory, and disk," *Intel Technology Journal*, Vol. 17, No. 1, No. 1, pp. 184–200, 2013.

[78] S. W. Jones, "Understanding the costs of MEMS products," IC Knowledge LLC, 2009.

[79] "International Technology Roadmap for Semiconductors," Tech. Rep. [Available online] http://www.itrs.net/reports.html.

[80] C. Albrecht, "IWLS 2005 Benchmarks," Tech. Rep., June 2005.

[81] Y. Li, H. Xu, R. Melhem, and A. K. Jones, "Space Oblivious Compression: Power Reduction for Non-Volatile Main Memories," *Proceedings of the GLSVLSI Symposium*, 2015.

[82] T. Yuan, S. Z. Ramadan, and S. J. Bae, "Yield prediction for integrated circuits manufacturing through hierarchical Bayesian modeling of spatial defects," *Transactions on Reliability 2011.*

[83] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An Open-Source Variation-Aware Design Kit," *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, 2007.

[84] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, Vol. 34, No. 4, No. 4, pp. 1–17, 2006.

[85] T. Yuan, S. Z. Ramadan, and S. J. Bae, "Yield prediction for integrated circuits manufacturing through hierarchical Bayesian modeling of spatial defects," *Transactions on Reliability 2011.*

[86] J. R. Carson, "The Heaviside operational calculus," *The Bell System Technical Journal*, Vol. 1, No. 2, No. 2, pp. 43–55, 1922.

[87] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization," *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 185–197, ACM, 2013.

[88] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise AND and OR in DRAM," *IEEE Computer Architecture Letters*, Vol. 14, No. 2, No. 2, pp. 127–131, 2015.

[89] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory Accelerator for Bulk

Bitwise Operations Using Commodity DRAM Technology," *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, (New York, NY, USA), pp. 273–287, ACM, 2017.

[90] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," *Proceedings of the 53rd Annual Design Automation Conference*, p. 173, ACM, 2016.

[91] J. Zhang, D. Kline, L. Fang, R. Melhem, and A. K. Jones, "Yoda: Judge me by my size, do you?," *Computer Design (ICCD), 2017 IEEE International Conference on*, pp. 395–398, IEEE, 2017.

[92] R. Melhem, R. Maddah, and S. Cho, "RDIS: A Recursively Defined Invertible Set Scheme to Tolerate Multiple Stuck-at Faults in Resistive Memory," *DSN*, pp. 1–12, 2012.

[93] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," *MICRO*, pp. 115–124, 2010.

[94] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," *MICRO*, pp. 433–444, 2013.

[95] J. Zhang, D. Kline Jr, L. Fang, R. Melhem, and A. K. Jones, "Dynamic partitioning to mitigate stuck-at faults in emerging memories," *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 651–658, IEEE Press, 2017.

[96] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing PCM main memory lifetime," *Proceedings of the conference on design, automation and test in Europe*, pp. 914–919, European Design and Automation Association, 2010.

[97] L. Jiang, Y. Zhang, and J. Yang, "Mitigating write disturbance in super-dense phase change memories," *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 216–227, IEEE, 2014.

[98] A. JEDEC, "JESD79-3F: DDR3 SDRAM Specification," 2012.

[99] "Official Site of the x86 Memory Testing Tool," *MemTest86*, 2019.

[100] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM," *HPCA 2006*.

[101] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, (New York, NY, USA), pp. 190–200, ACM, 2005.

[102] F. Huang, D. Feng, Y. Hua, and W. Zhou, "A wear-leveling-aware counter mode for data encryption in non-volatile memories," *DATE*, 2017.

[103] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices," Tech. Rep. SP 800-38E, NIST, 2010.

[104] P. Rogaway, "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC," Tech. Rep., Dept. of Computer Science. University of California, Davis, 2004.

[105] J. Kong and H. Zhou, "Improving privacy and lifetime of PCM-based main memory," *DSN*, pp. 333–342, June 2010.