THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Advances on Adaptive Fault-Tolerant System Components: Micro-processors, NoCs, and DRAM

ALIRAD MALEK

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017

Advances on Adaptive Fault-Tolerant System Components:
Micro-processors, NoCs, and DRAM
Alirad Malek
Göteborg, Sweden, 2017
ISBN: 978-91-7597-666-2

Ioannis Sourdis	Advisor	Assoc. Professor at Chalmers University of Technology
Vassilis Papaefstathiou	Co-Advisor	Post Doctoral researcher at FORTH-ICS
Onur Mutlu	Thesis Opponent	Professor at ETH Zürich
Yiannakis Sazeides	Grading Committee	Assoc. Professor at University of Cyprus
Ramon Canal	Grading Committee	Assoc. Professor at Universitat Politecnica de Catalunya
Stefanos Kaxiras	Grading Committee	Professor at Uppsala University

Doktorsavhandlingar vid Chalmers Tekniska Högskola
Ny serie Nr 4347
ISSN 0346-718X

Technical Report No. 150D
Department of Computer Science and Engineering
Chalmers University of Technology

Contact Information:

Division of Computer Engineering
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Telephone: +46 (0)31-772 10 00
`http://www.chalmers.se/cse/`

Author's e-mail: `aliradm@chalmers.se`

Printed by Chalmers Reproservice
GÖTEBORG, Sweden 2017

To my family, Naghmeh, Alma and ...

Advances on Adaptive Fault-Tolerant System Components: Micro-processors, NoCs, and DRAM

Alirad Malek
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden

Abstract

The adverse effects of technology scaling on reliability of digital circuits have made the use of fault tolerance techniques more necessary in modern computing systems. Digital designers continuously search for efficient techniques to improve reliability, while keeping the imposed overheads low. However, unpredictable changes in the system conditions, e.g. available resources, working environment or reliability requirements, would have significant impact on the efficiency of a fault-handling mechanism. In the light of this problem, adaptive fault tolerance (AFT) techniques have emerged as a flexible and more efficient way to maintain the reliability level by adjusting to the new system conditions. Aside from this primary application of AFT techniques, this thesis suggests that adding adaptability to hardware component provides the means to have better trade-off between achieved reliability and incurred overheads. On this account, hardware adaptability is explored on three main components of a multi-core system, namely on micro-processors, Networks-on-Chip (NoC) and main memories. In the first part of this thesis, a reliable micro-processor array architecture is studied which can adapt to permanent faults. The architecture supports a mix of coarse and/or fine-grain reconfiguration. To this end, the micro-processor is divided into smaller substitutable units (SUs) which are connected to each other using reconfigurable interconnects. Then, a design-space exploration of such adaptive micro-processor array is presented to find the best trade-off between reliability and its overheads, considering different granularities of SUs and reconfiguration options. Briefly, the results reveal that the combination of fine and coarse-grain reconfiguration offers up to 3$\times$ more fault tolerance with the same overhead compared to simple processor level redundancy. The second part of this thesis, presents RQNoC, a service-oriented NoC that can adapt to permanent faults. Network resources are characterized based on the particular service they support and, when faulty, they can be bypassed through two options for redirection, i.e. service merging (SMerge) and/or service detouring (SDetour). While SDetour keeps lanes of different services isolated, suffering longer paths, SMerge trades service-isolation for shorter paths and higher connectivity. Different RQNoC configurations are implemented and evaluated in terms of network performance, implementation results and reliability. Concisely, the evaluation results show that compared to the baseline network, SMerge maintains at least 90% of the network connectivity even in the presence of 32 permanent network faults, which is more than double versus SDetour, but will impose 51% more area, 27% more power and has a 9% slower clock. Finally, the last part of this thesis presents a fault-tolerant scheme on the DRAM memories that enables the trade-off between DRAM capacity and fault tolerance. We introduce Odd-ECC DRAM mapping, a novel mechanism to dynamically select Error-Correcting-Codes (ECCs) of different strength and overheads for each allocated page of a program on main memories. Odd-ECC is applied to memory systems that use conventional 2D, as well as 3D-stacked DRAMs and is evaluated using various applications. Our experiments show that compared to flat memory protection schemes, Odd-ECC reduces ECCs capacity overheads by up to 39% while achieving the same Mean Time to Failure (MTTF).

Keywords: Adaptive fault tolerance, Micro-processors, Network-on-Chip, Main memory, Reliability analysis, Quality-of-Service, Error-Correcting Codes

List of Publications

Parts of the contributions presented in this thesis have previously been published in the following manuscripts

- **Alirad Malek**, Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, Ioannis Sourdis, "Odd-ECC: On-demand DRAM Error Correcting Codes", *International Symposium on Memory Systems (MEMSYS)*, October, 2017.

- **Alirad Malek**, Ioannis Sourdis, Stavros Tzilis, Yifan He, Gerard Rauwerda, "RQNoC: A Resilient Quality-of-Service Network-on-Chip with Service Redirection", *ACM Transactions on Embedded Computing Systems (TECS)*, 15, no. 2 (2016): 28.

- Ioannis Sourdis, Danish Anis Khan, **Alirad Malek**, Stavros Tzilis, Georgios Smaragdos, Christos Strydis, "Resilient chip multiprocessors with mixed-grained reconfigurability", IEEE Micro. 2016 Jan; 36(1):35-45.

- **Alirad Malek**, Stavros Tzilis, Danish Anis Khan, Ioannis Sourdis, Georgios Smaragdos, Christos Strydis, "Reducing the performance overhead of resilient CMPs with substitutable resources", *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, October, 2015, pp.191-196.

- **Alirad Malek**, Stavros Tzilis, Danish Anis Khan, Ioannis Sourdis, Georgios Smaragdos, Christos Strydis, "A probabilistic analysis of resilient reconfigurable designs", *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, October, 2014, pp.141-146.

- Georgios Smaragdos, Danish Anis Khan, Ioannis Sourdis, Christos Strydis, **Alirad Malek**, Stavros Tzilis, "A Dependable Coarse-grain Reconfigurable Multicore Array", in 21st Reconfigurable Architectures Workshop (RAW), 2014.

The following papers are related but not covered in this thesis.

- I. Sourdis, C. Strydis, A. Armato, C.S. Bouganis, B. Falsafi, G.N. Gaydadjiev, S. Isaza, **A. Malek**, R. Mariani, D.K. Pradhan, G. Rauwerda, R.M. Seepers, R.A. Shafik, G. Smaragdos, D. Theodoropoulos, S. Tzilis, M. Vavouras, S. Pagliarini, and D. Pnevmatikatos, "DeSyRe: On-Demand Adaptive and Reconfigurable Fault-Tolerant SoCs ", in 10th Int'l Symp. on Applied Reconfigurable Computing (ARC), pp. 312-317, 2014.

- I. Sourdis, C. Strydis, A. Armato, C.-S. Bouganis, B. Falsafi, G. Gaydadjiev, S. Isaza, **A. Malek**, R. Mariani, D.N. Pnevmatikatos, D.K Pradhan, G. Rauwerda, R. Seepers, R.K. Shafik, K. Sunesen, D. Theodoropoulos, S. Tzilis, M. Vavouras, "DeSyRe: on-Demand System Reliability", in Elsevier Microprocessors and Microsystems, Special Issue on European Projects in Embedded System Design, November, 2013.

Acknowledgments

No road is too long with good company. On the verge of completing this seemingly never-ending chapter of my life, I would like to take this opportunity to thank everyone who helped me reach this point.

First of all, I would like to express my deepest gratitude to my advisor Yiannis Sourdis for his invaluable guidance and inspiring patience. I admire his encouraging enthusiasm for our work and I appreciate all his efforts to make me a better researcher in the field.

My warmest thanks goes to my co-advisors Vassilis Papaefstathiou and Pedro Trancoso for all those great discussions and insightful comments. It was a great honor and pleasure to work with you.

I would like to sincerely appreciate my examiners during these years, Georgi Gaydadjiev, Per Stenström and Jan Jonsson for their constructive feedback and advice. Special thanks to Sally McKee and Lars Svensson for their kind support and guidance during different phases of my research.

Next, I like to thank all my friends and colleagues in the Department of Computer Science and Engineering: Risat, Miquel, Angelos, Prasanth, Petros, Albin, Prajith, Stefano, Negin, Bhavishya, Jacob, Madhavan, Behrooz, Fatemeh, Dmitry, Waqar and anyone who, one way or another, helped me during this time. Special thanks to Vaggelis for huge technical and moral supports during final stages of my work. I learned a lot from you guys.

During all these years, our office was my second home and that's all thanks to my great office-mates Stavros and Ahsen. Thank you for the pleasant working environment and your kind-hearted support during my research.

I would like to express my sincere gratitude to our kind administrative staff in the department, Eva Axelsson, Monica Månhammar, Marianne Pleen-Schreiber and Rune Ljungbjörn for their help and support.

This work has been partly funded by the European Union's Framework Programme 7 DeSyRe project (grant agreement 287611), EMC2 project (grant agreement 621429)

and Horizon 2020 Programme ECOSCALE project (grant agreement 671632).

Many thanks to my close friends, Hessam, Dorreh, Nojan, Behnaz, Homayoun, Mitra, Kamyar and all others with whom I've got a lot of great memories.

I am cordially grateful to my parents and my sisters for their unconditional support and chances they've given me over the years. Thanks Mom. Thanks Dad.

Finally, I would like to thank my wife, Naghmeh for her love and patience over all these years and the joy she brings to my life. None of this would have been possible without you. Thank you.

Alirad Malek
Göteborg, December 2017.

Contents

List of Figures

List of Tables

1 Introduction

Advances in semiconductor technology have resulted in the emergence of new challenges in the design of reliable computing systems. As transistors' feature sizes continue to shrink, they become more susceptible to fault-factors such as manufacturing defects, environmental disturbances and wear-out phenomena [1]. In a digital circuit, these fault-factors will manifest as transient, intermittent or in more severe cases as permanent faults [2]. Without any mechanism to tolerate faults, even a single faulty circuitry could threaten the correct functionality of the system or make an entire chip unusable. This issue is even more crucial for embedded systems in mission/safety-critical domains such as medical, aerospace and automotive, where, if not handled appropriately, faults could have catastrophic consequences.

There have been many efforts to improve the reliability and availability of systems in the past. In general, modern computing systems can be equipped with various fault tolerance techniques applied to their comprising components, e.g. processors, interconnection infrastructure and memory elements, which altogether aim to increase the overall reliability of the system. Compared to end-to-end holistic system level fault tolerance, dealing with each component separately gives the designer more flexibility in addressing component specific faults, i.e. faults that might occur only in particular components, or in adjusting the reliability level based on the criticality of the component. Regardless of the target component, these techniques always employ some form of redundancy, i.e. information, temporal or spatial redundancy, to provide fault resilience [3].

This introductory chapter is a short overview for the work presented in this thesis. The remainder of this chapter is organized as follows: Section 1.1 describes the main problem statement, and subsequently, section 1.2 presents the main statement of this research. Section 1.3 covers an overview of thesis objectives. Section 1.4 summarizes the main contributions and finally, section 1.5 provides an outline for the rest of this thesis.

1.1 The Problem: Overheads of Fault Tolerance

Adding the fault tolerance property to any system imposes undesirable overheads. Depending on the employed redundancy scheme, these overheads could be in terms of design complexity, performance, silicon area or consumed energy. In fact, in all fault tolerance approaches, there is a direct relation between at least one of these overheads and the achieved reliability level, where higher reliability comes at higher costs. Hence, the problem that dependable system designers face is how to have an *efficient* fault handling mechanism which offers sufficient reliability while the overheads are within acceptable thresholds.

On the other hand, due to inevitable dynamic changes in system conditions, sustaining the efficiency of a fault tolerance mechanism can be challenging. There are several factors that might change in a system during its lifetime, creating the need for more flexible fault tolerance techniques that can adapt to changes. The source of these changes could be: (i) changes in systems status, e.g. number of available resources after occurrence of permanent faults, (ii) changes in the working environment that might affect the fault-rates, (iii) or changes in the reliability requirements based on the user's application [4]. Failure to adapt to these changes could potentially reduce the efficiency of the employed fault tolerance technique. In this regard, *Adaptive Fault Tolerance* (AFT) techniques have been proposed as an approach for providing the flexibility needed for designing efficient fault-tolerant systems [4–6]. The main purpose of AFT techniques is to dynamically satisfy the reliability requirements by adapting to changes in the system conditions [7, 8].

1.2 Thesis Statement: Hardware Adaptivity Offers Better Reliability-Overheads Trade-Off

In this thesis we study *the design of fault-tolerant system components that provide hardware support for adaptive fault tolerance techniques*. We exploit hardware flexibility to improve the efficiency of fault handling schemes in different system components. Ultimately, in this thesis we seek out to show that:

> *Adding adaptability to the hardware, enables components to demonstrate better trade-offs between reliability and its overheads.*

In other words, although the main purpose of using adaptive fault tolerance is to maintain a fixed level of fault tolerance, while adapting to changes in the system, the flexibility of such designs opens up an opportunity to achieve better balance between reliability and its overheads.

Accordingly, our goal in this thesis is to investigate the validity of the aforementioned statement, focusing on three main components of a multi-core system: micro-processors, a Network-on-Chip (NoC) as the communication infrastructure, and off-chip Dynamic Random Access Memories (DRAMs).

1.3 Thesis Objectives: Adaptive Fault-Tolerant System Components

In the following, we present an overview of the work in this thesis for each of the components at hand: micro-processors, NoC and main memories.

1.3.1 Adaptive Fault-tolerant Micro-processors

In the first part of this thesis, we study a chip-multiprocessor (CMP) with substitutable units that can adapt to permanent fault. Our main objective in this part is:

to find the granularity of substitutable units as well as the reconfiguration mix that achieve, for a given fault-rate, the best trade-off between reliability and incurred overheads (i.e. area, performance and power).

As our use-case, we consider a CMP with multiple identical processors, i.e. simple MIPS-like in-order RISC. The architecture is modified to support either coarse or fine-grain reconfiguration, as well as combination of both, henceforth referred to as mixed-grain, for adapting to permanent faults. To this end, the processor is divided into smaller substitutable units (SUs) which are connected to each other using reconfigurable interconnects. In coarse-grain (CG) reconfiguration, when a permanent fault is detected on a processor, the affected unit is isolated and other fault-free units of the processor will be used as spares for the faulty processors on the CMP. When the option of fine-grain (FG) reconfiguration is available, it is possible to instantiate the affected SU on the FG fabric and use reconfigurable interconnects to replace it with the newly configured one. Figure 1.1 shows the two options of coarse-grain and fine-grain reconfiguration for an array of four adaptive processors.

Considering this reconfigurable architecture, whether or not fine- and/or coarse-grain reconfiguration is employed, is a design choice which depends on the desired level of fault tolerance and accepted overheads. Thus, two important issues that arise are how to measure and compare: achieved fault tolerance and incurred overheads for the two reconfiguration alternatives. Evidently, these issues need to be addressed considering different fault densities and also how they associate with the granularity, i.e size of SUs. Overcoming these issues will eventually help us to find which *reconfiguration mix* and *granularity of SUs* will result in having the best trade-off between fault tolerance and overheads in each case. Concisely, in the first part of this thesis we:

(i) analyze the reliability and the overheads of coarse and fine-grain reconfigurable CMPs with SUs;

(ii) perform a design space exploration of adaptive fault-tolerant CMPs to find the most efficient reconfiguration mix for a given fault rate.

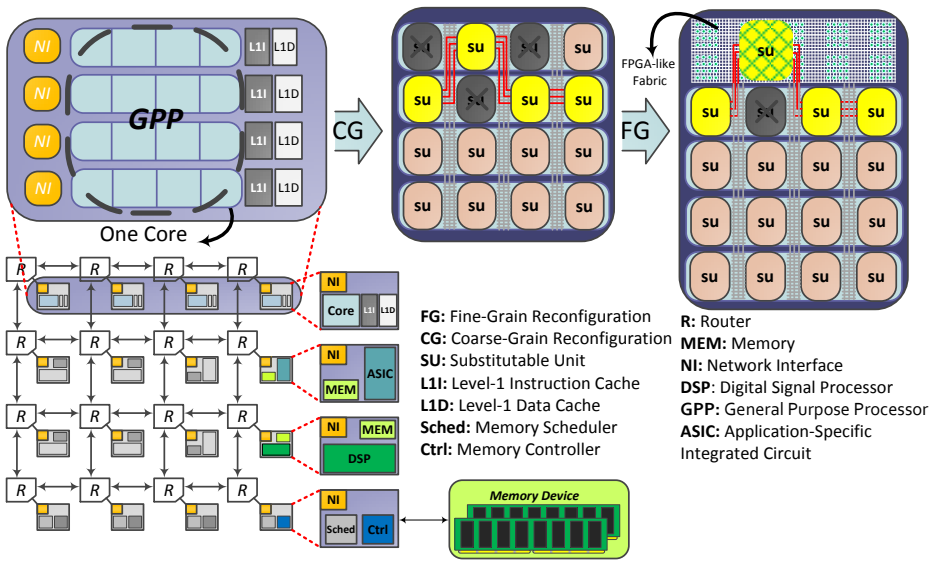


Figure 1.1: Adaptive processor in a chip-multiprocessor where permanent faults can be tolerated by using coarse-grain and fine-grain reconfiguration.

Related Work: Prior work have proposed several online-testing mechanism that can be used to detect and locate permanent hardware faults/bugs in microprocessors [9–12] or in other uncore components of a system-on-chip [13, 14]. A widely used approach for tolerating the detected faults is hardware adaptation in which a component is divided into smaller entities, i.e. SUs, that can be isolated and replaced by spares when required. The granularity of SUs can be from an entire core [15–17], down to gates/wires in a design [18]. Well-known examples of core-level redundancy are high-availability systems such a IBM-ZSeries [15] and HP-NonStop [16] servers, that employ triple-modular redundancy and dual-modular redundancy, respectively. For finer-than-core granularities, researchers have proposed reconfiguration at the level of functional units [19], and more commonly at processors pipeline stages, e.g. CCA [20], StageNet [21] and Viper [22]. At even finer granularities, there have been proposals for using spare bits in the processors data-path [18], or using field programmable gate array (FPGA) and programmable logic array (PLA) to instantiate permanently faulty parts of a design [23–25]. Contrary to the prior works, instead of selecting a particular SU granularity, the adaptive micro-processors proposed in this part of the thesis can employ a mix of both coarse- and fine-grained reconfiguration options, offering higher flexibility to combat permanent faults.

Generally, the mechanism to isolate and replace a faulty SU requires modifications to add adaptability to the hardware. Evidently, the chosen granularity of SUs has a significant effect on the level of reliability and its overheads. In the past, there have been some studies which explored the impact of SU granularity on the reliability and

overheads of reconfigurable architectures [26–28]. However, the scope of these studies are limited to FPGA based designs. In a more recent work, researchers analyzed how area and delay overheads of a reconfigurable design change with the chosen granularity [29]. Nevertheless, to the best of our knowledge, the work presented in this thesis is the first study that describes a methodology to perform a design space exploration of reconfigurable designs, considering different granularities of SUs, reconfiguration options and fault densities.

1.3.2 Adaptive Fault-tolerant Network-on-Chip (NoC)

In the second part of this thesis, we focus on a reliable NoC architecture that can adapt to changes in the available resources as a result of permanent faults. Our key objective is:

to design and evaluate a service-oriented NoC that enables us to trade service-isolation for reliability.

Reliable interconnects are essential for the correct functionality of system-on-chips (SoCs) with multiple communicating components. In modern SoC designs, NoCs have become the prominent choice for on-chip communication backbones due to their regular and robust structures. On the other hand, the diversity of transmitted messages between on-chip components has raised the demand for NoCs which can support different traffic classes with different attributes. Thus, service-oriented NoCs that support certain quality of service (QoS), e.g. latency and throughput, are gaining more and more attention. In a commonly practiced approach, the quality of service is guaranteed through utilizing isolated, physical-lanes per traffic class [30, 31].

In this thesis, we consider a service-oriented NoC architecture that supports different traffic classes, each corresponding to a specific service. Each router in the network has multiple data-paths, one per service, while packets of all services share the links connecting the routers in the network. The inherent intra-router and inter-router redundancies in this NoC architecture can be exploited to improve the fault tolerance of the network. However, the physical isolation of different service-lanes will be impacted by such fault tolerance schemes, which opens up a trade-off between reliability and performance of different traffic classes. Figure 1.2 shows an example of tolerating permanent faults, leveraging the redundancies inside a service-oriented NoC[1]. Based on these descriptions, in the second part of this thesis we:

(i) design a fault-tolerant service-oriented NoC that is able to adapt to permanent faults through breaching the isolation of its physical service-lanes;

(ii) evaluate and compare our proposed techniques in terms of offered reliability and performance costs.

[1] More implementation details can be found in Chapter 3.

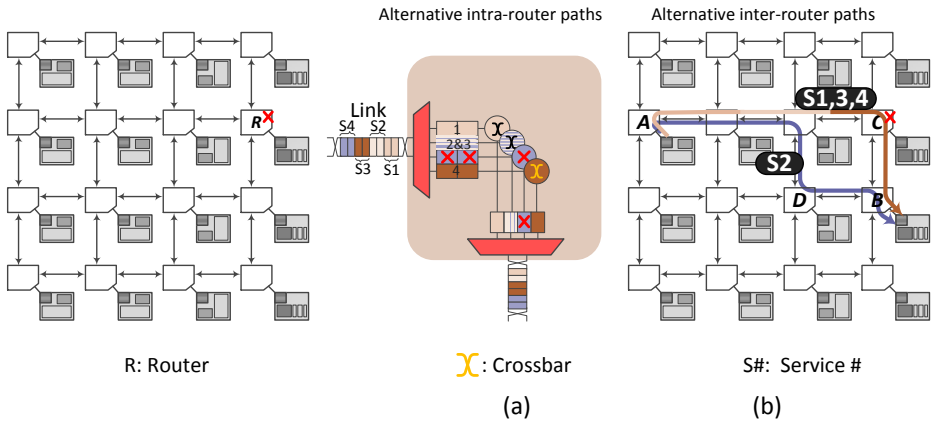


Figure 1.2: Tolerating Permanent faults in a service-oriented NoC using: intra-router (a) and, inter-router (b) redundancies.

Related Work: In the past a plethora of techniques have been suggested to combat permanent faults in NoCs. Many approaches modify the routing algorithm of a NoC to be dynamically adaptive and avoid faulty links or routers [32–38]. Other techniques re-route packets without modifying the actual routing algorithm. Instead, they perform a detour, selecting an intermediate destination for packets that would normally need to pass through some faulty links or routers [39–42]. In addition to rerouting of packets, faulty network parts can be bypassed in hardware [1, 43–48]. Besides Kakoee et al. [47], none of the above techniques is addressing fault tolerance on a service-oriented NoC nor analyzes the affect of faults to the performance of individual traffic classes. Moreover, the techniques proposed in this thesis are aimed to increase the network connectivity, therefore, are orthogonal to adaptive routing algorithms which can benefit from the existence of more paths between network elements. We provide more details on related work in Chapter 3.

1.3.3 Adaptive Error-Correcting-Codes (ECCs) for Main Memories

In the last part of this thesis, we focus on the design of reliable off-chip main memories that can adapt to changes in the reliability requirements. Our key objective is:

to design and evaluate a fault tolerance mechanism that allows us to trade capacity for reliability within a DRAM module.

In particular, we look for an adaptive solution which enables us to pay only a capacity cost –needed for a memory protection scheme– which is proportional to the critically of the data stored in the memory.

Recent studies show that faults in different data regions of an application have different

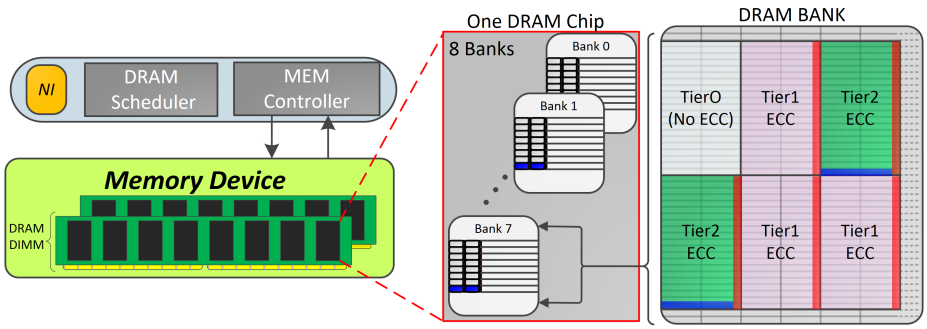


Figure 1.3: Adaptive ECC memory mapping scheme allows systems to select dynamically on-demand different protection levels (ECCs) for different allocated pages.

effect on the applications outcome and, hence on the reliability of the system [49, 50]. Consequently, fault tolerance schemes that offer flat, i.e. fixed, level of protection, potentially, impose unnecessary overheads on the DRAM capacity and performance. Motivated by this observation, the last part of thesis is dedicated to the design of flexible Error-Correcting-Codes (ECCs) that can adapt to the reliability requirements of different data regions, providing the possibility to trade memory capacity for reliability. We explicitly focus on DRAM modules and study how to provide support for dynamic ECC constructs in conventional 2D, as well as 3D-stacked DRAMs. In particular, we aim at designing an adaptive ECC DRAM mapping in which different data regions within a single DRAM DIMM are protected by different ECCs, i.e. Tier0, Tier1 and Tier2, as illustrated in Figure 1.3[2]. Concisely, in the third part of this thesis we:

(i) analyze the sensitivity of applications to faults in each part of their data, to gain insight about criticality of different data-regions;

(ii) design and evaluate a new DRAM mapping scheme which allows us to have different protection levels, i.e. ECCs, for different allocated pages inside a DRAM module.

Related Work: In the past there have been many works describing different metrics and methodologies to analyze applications sensitivity to faults [51–57]. The outcome of such analysis is used by some researchers to motivate different reliability techniques, applied to software and hardware [49, 50, 53, 56, 58, 59]. In this part of our work, we follow the same strategy and perform a comprehensive sensitivity analysis aiming to observe the impact of faults that happen in different data regions, on the outcome of an application. However, we further extend the analysis and propose a methodology to explore the impact of different protection levels per data region, on the applications reliability.

Tolerating faults in main memories has been widely studied in prior works, both on conventional 2D [60–70], as well as emerging 3D-stacked DRAMs [71–74]. Among the

[2]Mapping details can be found in Chapter 4.

techniques proposed for tolerating faults in 2D-DRAMs, some offer the flexibility to use different levels of protection [60–64, 75, 76]. However, except for [76], the flexibility of these techniques cannot be exploited to save DRAM capacity. Moreover, some of the resilience schemes are designed for specific memory system configurations, requiring multiple memory channels [65, 75], using ECC-DIMMs [66–68, 76], access granularity prediction [62] or data compression [69, 70].

On the other hand, most of the reliability techniques proposed for 2D-DRAMs can not be directly applied to 3D-stacked DRAMs. Therefore, in recent years there have been new proposals, addressing the reliability issues of 3D memory structures [71–74]. However, none of the proposed schemes supports having different protection regions within a single 3D-stacked DRAM.

With respect to the current techniques for improving the reliability of main memory systems, the work presented in this thesis provides different protection levels for data stored in the memory system, enabling the user to trade DRAM capacity for reliability. The proposed technique is applied to conventional non-ECC 2D-DRAM modules, as well as to generic 3D-stacked DRAMs. More detailed related work is provided in Chapter 4.

1.4 Contributions

According to the above objectives, this section presents a summary of the main contributions of this thesis for each of the three components at hand, i.e. micro-processors, NoC and main memories.

1.4.1 Adaptive Fault-tolerant Micro-processors

In the first part, we study the design of reliable fault-tolerant micro-processors that can adapt to permanent faults. We aim to find the most efficient granularity of substitutable units which delivers the best trade-off between the achieved reliability and overheads. In essence, this part of the thesis makes the following contributions:

> We introduce a generic methodology for analyzing the reliability of reconfigurable components with substitutable units. We take into account the area overhead of both reconfiguration options, i.e. coarse and/or fine-grain reconfiguration, at different granularities of substitutable units. The methodology is then used to perform a comprehensive design space exploration of an adaptive reliable micro-processor array at different fault densities. Our study allows us to identify the most efficient design choice, i.e. with specific reconfiguration options and granularity of substitutable units, at a particular fault density.

1.4.2 Adaptive Fault-tolerant NoC

In the second part of this thesis, we focus on the design of a reliable service-oriented NoC which can adapt to permanent faults. The key objective is to design a fault tolerance

scheme that enables us to trade service-isolation for reliability. Briefly, our contributions in this part are as follows:

> We categorize network resources based on the particular service that they support and, when faulty, bypass them, allowing the respective traffic class to be redirected. We propose two alternatives for service redirection: (i) *Service Detour* that uses longer alternative paths through resources of the same service to bypass faulty network parts, keeping traffic classes isolated, (ii) *Service Merge* that uses resources of other services providing shorter paths but allowing traffic classes to interfere with each other. The above fault tolerance techniques are extensively evaluated in terms of network performance (latency, throughput), implementation results (area, power, frequency) and also the percentage of successful packet delivery. Finally, a network connectivity model is created and used to measure the obtained reliability under different fault densities.

1.4.3 Adaptive Error-Correcting-Codes (ECCs) for Main Memories

In the last part of this thesis, we focus on the design of reliable off-chip main memories. Our main objective is to design flexible ECC constructs that enable us to trade DRAM capacity for reliability. Accordingly, this part of our work makes the following contributions:

> We analyze various applications, measuring their sensitivity (Mean-Time to Failure) to faults injected in different regions of their data. We introduce Odd-ECC, a new flexible memory mapping scheme that allows systems to select dynamically on-demand different protection levels (ECCs) for different allocated pages. We show how Odd-ECC is applied to both traditional 2D and 3D-stacked DRAMs. Afterwards, using the same applications studied and guided by their MTTF analysis, we select combinations of fault tolerance levels (i.e. Tier0, Tier1, Tier2) for each data region to be supported by the proposed Odd-ECC scheme. These Odd-ECC setups are then compared with flat fixed memory fault tolerance levels in terms of the overall MTTF of the application at hand, as well as in terms of overheads in memory capacity, system performance and energy costs.

1.5 Thesis Outline

The remainder of the thesis is organized as follows:

In Chapter 2, a probabilistic reliability analysis of a generic reconfigurable design is presented. The analysis is then used to evaluate the reliability of an adaptive micro-processor architecture. We first describe different forms of reconfiguration and identify the required architectural modifications of our use-case processor. Furthermore, we present a probabilistic method to calculate the fault tolerance of such designs. Different designs of the adaptive processor are then evaluated in terms of area, performance and

power. Moreover, the performance impact of pipelining reconfigurable interconnects, which connect SUs of the CMPs, is explored. Finally, based on the results of evaluation and the probabilistic calculation, a comprehensive design space exploration is presented.

In Chapter 3, an adaptive service-oriented NoC architecture is designed and implemented which supports inter-router and intra-router traffic redirection mechanisms, thereby providing tolerance to permanent faults. We first introduce our baseline service-oriented NoC and its specifications. Afterwards, we introduce our proposed NoC architecture where different fault-tolerant techniques are described to improve NoC reliability in presence of permanent faults at the links and the routers. More specifically, we present the Service Merge and Service Detour schemes as the main contributions of our work. Our techniques are subsequently compared with each other using a cycle-accurate simulator. Further, in order to evaluate the reliability of different schemes, an analytical study is performed to calculate the network connectivity in presence of different number of faults.

In Chapter 4, an adaptive ECCs scheme is proposed for DRAM main memories. We first perform an extensive applications reliability analysis and confirm that an application may have different sensitivity to faults in different subsets of the data it uses. Furthermore, we define a methodology to explore the impact of different protection levels per data region, on the applications reliability. Afterwards, we describe Odd-ECC DRAM mapping strategy that supports three different levels of protection. Finally, we evaluate Odd-ECC mechanism in terms of achieved applications reliability, performance and energy for both 2D and 3D-stacked DRAM memories.

In the end, Chapter 5 presents the concluding remarks. The chapter summarizes the content of this thesis, outlines contributions and findings and finally discusses possible future research directions.

2

Analysis of Repairable Adaptive Micro-processors

As technology scales chips become less reliable. Shrinking device features make circuits more susceptible to permanent faults due to manufacturing defects and wear-out phenomena [2]. Even a single permanent fault can damage an entire chip lowering production yields or jeopardizing availability when it appears in field. The wide use of embedded systems in various application domains and the fact that such systems tend to become more complex and advanced as time passes, makes reliability and availability an even more pressing issue. Moreover, there is a number of mission-critical applications whereby faults are unacceptable; e.g. in the space, automotive, and medical domains. Such applications require high fault tolerance to deal with the increasing number of faults in emerging technologies.

In this chapter, we focus on System-on-Chip (SoC) recovery from permanent faults – simply denoted in here as *faults* – aiming at increased system availability and reliability at high fault densities. One general approach to fault tolerance relies on dividing a design into basic blocks identical to each other, called *Substitutable Units* (SUs), whereupon so-called *sparing* strategies are employed: A faulty block of a system can be substituted by a spare (functioning) one. Clearly, this strategy requires redundancy of components and reconfigurability to replace damaged parts so as to keep the system functional.

Previous approaches divide a chip into smaller redundant SUs that can be isolated and replaced, if damaged, by spare identical parts. Such redundancy may appear in various granularities. In a *multiprocessor SoC*, core-redundancy is the most common choice [17, 77]. Alternatively, smaller parts, such as pipeline stages and functional units,

can be used as spares to repair a failing component [19–22]. At the other end of the design space, fine-grain logic (e.g., gate-level redundancy) can be used: for instance, Field-Programmable Gate Arrays (FPGAs) tolerate permanent faults by changing the configuration and/or the placement of a design [24]. The particular granularity (size of SU) chosen in a design introduces a *trade-off between reliability and overheads*. Finer granularities are more resilient to faults, but also less efficient in terms of area, performance and power consumption. Coarser redundancy tolerates fewer faults, but incurs lower overheads to a design.

This chapter presents a probabilistic analysis of the fault tolerance offered by hardware reconfigurability. Considering a generic design of components divided to SUs, we analytically estimate the resilience of different granularities. Thereby, we determine the most efficient fault-tolerant reconfigurable design choice for a particular fault density. Concisely, the main contributions of this work are the following:

- We analytically calculate the probability of constructing a certain number of fault-free components via reconfiguration having as an input parameter the fault density.
- Then, we measure the area overheads of reconfigurability using a reconfigurable RISC processor with substitutable parts as a use-case component.
- We evaluate various reconfiguration options, measuring the average number of fault-free components as well as the probability of delivering a minimum number of fault-free components in a given silicon area. In so doing, we identify the most efficient fault-tolerant design that gives us the best trade-off between reliability and incurred overheads, at a particular fault density.

In the remainder of this chapter, related work is presented first in Section 2.1, then Section 2.2 describes reconfigurable design alternatives for repairing faulty components. Section 2.3 details the design of the particular reconfigurable processor used in this study. Section 2.4 offers a probabilistic analysis of fault tolerance in reconfigurable designs while Section 2.5 evaluates the fault tolerance of different design points. Finally, in Section 2.6 overall conclusions are drawn.

2.1 Related Work

In the past, a large number of design techniques have been introduced for tolerating permanent faults. They consider different SU sizes ranging from entire components, e.g. micro-processors, to fine-grain logic and have different fault tolerance and design overheads.

Existing commercial systems offering high availability, such as the IBM z-series [15] and the HP NonStop systems, [16], provide redundant cores and dual modular redundancy (DMR) to combat permanent faults. LaFrieda et al. [17] showed that DMR of dynamically coupled cores improves fault tolerance. Disabling [78] and isolating [77] permanently faulty cores can sustain degraded performance, while dark silicon makes it possible to have additional spare components [79]. Moreover, partly damaged cores can potentially be used with degraded functionality by allowing affected threads to migrate to other

cores [80], or even to be rescued using advanced diagnostics and voltage/frequency tuning to reverse some of the wear-out phenomena [81].

In order to deal with the increasing number of permanent faults, designers turned to finer granularities. The SUs may be functional units [19] or pipeline stages [20–22]. In the latter case, processors — mostly simple RISCs — are modified to support dynamic replacement of stages.

Finer granularities further increase the fault tolerance of a design. A processor data-path can be protected by adding spare-bits [18], while a control-path can use field-programmable control logic to tolerate faulty states [23]. Finally, FPGAs and Programmable Logic Arrays (PLA) can be used as the hardware substrate of a fault tolerant SoC [24].

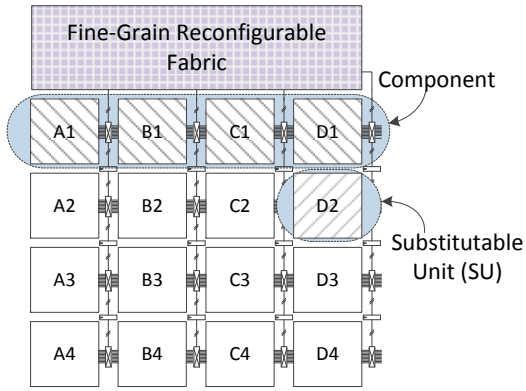
Following a passive fault-avoidance approach (disabling [78], isolating [77]) is inherently inefficient as it sacrifices availability when permanent faults occur. On the other hand, fault tolerance through sparing, replacing or repairing faulty components requires flexible wiring and multiplexing of the SUs, which not only adds significant delays to the design, but increases area and power consumption.

In general, larger SU sizes incur lower overheads, but also offer lower flexibility and hence lower fault tolerance. To the best of our knowledge, this is the first study to find the reconfigurable-design choices that maximize component availability for a given technology and a particular fault density. We explore a design space of various granularities of SUs using either coarse- or fine-grain reconfigurability, or a mix of both and analyze how component availability scales in different fault-densities.

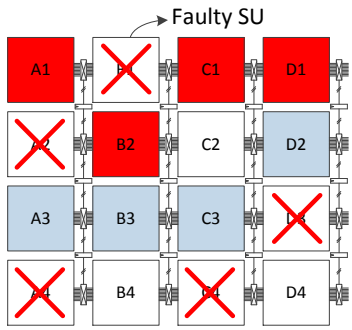
2.2 Reconfigurable Designs for Tolerating Permanent Faults

We discuss next the two options of reconfigurability, studied in this chapter, to combat permanent faults. A component can be partitioned into SUs that are interconnected with reconfigurable interconnects to allow one to substitute another; we call this coarse-grain reconfigurability. A second option is using fine-grain, FPGA-like, reconfigurable logic, shared among components and used to replace damaged parts when lacking identical spares; this is denoted as fine-grain reconfigurability. A faulty SU can be replaced by either an identical spare unit, presumably taken from a neighboring unused/faulty component, or instantiated in the fine-grain reconfigurable logic. We consider that a reconfigurable design may offer either one of the two reconfiguration options (coarse- or fine-grain) or both at different sizes of SUs (granularities).

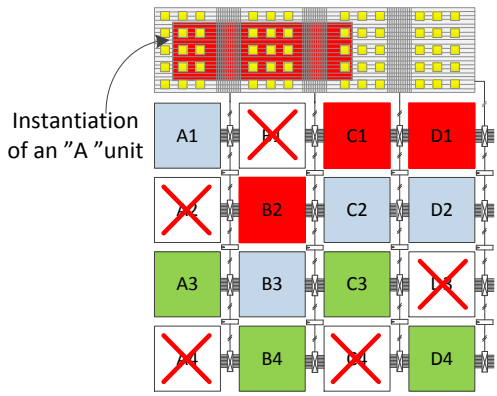
A generic example of a reconfigurable design that offers both coarse- and fine-grain redundancy is illustrated in Figure 2.1. The partitioning of the design is depicted as an array. Each column comprises identical, interchangeable, SUs, and each row represents a component in its fault-free configuration. The block at the top represents the shared fine-grain logic. A functioning component can be constructed connecting a single undamaged cell from each column; in case there is no fault-free cell for a particular column, the



(a) Four components with 4 Substitutable units (SUs) each and a fine grain logic block.



(b) Example of CG reconfiguration constructing 2 fault-free components: component-1 (A1, B2, C1, D1) and component-2 (A3, B3, C3, D2).



(c) Example of CG and FG reconfiguration constructing 3 fault-free components: component-1 (A{FG}, B2, C1, D1), component-2 (A1, B3, C2, D2), and component-3 (A3, B4, C3, D4).

Figure 2.1: A design of four identical components with coarse (CG) and fine-grain (FG) reconfigurability.

fine-grain block can be used as a wild card to replace it.

The granularity of a design affects its area overheads. The largest SU of a component defines the minimum size of the fine-grain block, consequently, it is more efficient to partition a component into units of similar size. Moreover, a larger number of smaller SUs requires more wiring, but also needs a smaller fine-grain block.

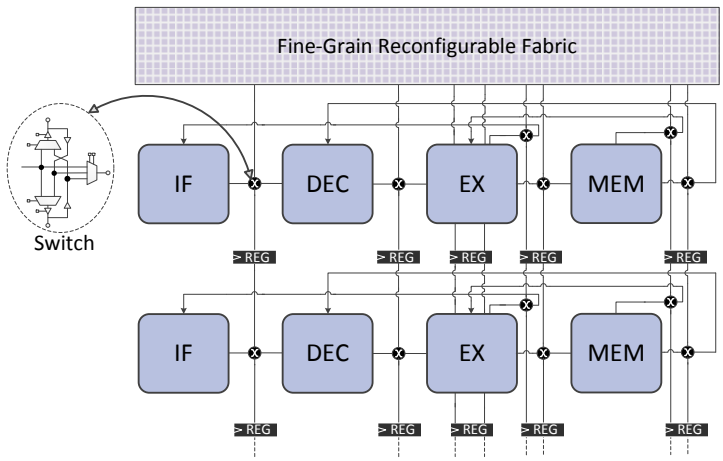


Figure 2.2: An adaptive processors array with substitutable stages.

2.3 A Reconfigurable Adaptive RISC Processor

In order to conduct our design-space exploration and retrieve real measurements for the area overheads of reconfigurability, we consider as a use-case a coarse-grain reconfigurable RISC architecture with substitutable parts, previously described in [82]. Moreover the architecture is modified to additionally support fine-grain reconfiguration [83].

Our use case is a 32-bit, in-order RISC processor with four pipeline stages: Instruction Fetch (IF), Decode (DC), Execute (EX), and Memory (ME). The processor has a local instruction- and data-memory (32Kbyte each) and a 16-entry register file (RF). The architecture offers decoupled pipeline stages, which can be substituted by any other identical stage or by fine-grain reconfigurable logic. This is achieved using reconfigurable interconnects. Interconnects that traverse across different processors are expected to be used infrequently. We therefore implement them as bidirectional interconnects and use switches with tri-states to go across different components of the same processor. Compared to unidirectional interconnects, this halves the wiring resources needed, but adds some extra delay to the switches.

In order to minimize the additional delay introduced by the reconfigurable interconnects and switches, interconnects that span across the processor boundaries

are also pipelined, as depicted in Figure 2.2. Thereby, sharing stages from neighboring cores introduces additional stages to a pipeline. As a consequence, we need to support a variable pipeline depth in different configurations, but operating frequency scales better with the number of components [82]. The direct effect of this is that the architecture must adapt and operate correctly while remaining oblivious to the number of (extra) stages added to its pipeline; that is, reconfigurability needs to be transparent to the application level guaranteeing binary compatibility among different processor configurations. The performance impact of pipelining the interconnects that connect the SUs of processors is studied in section 2.5.2.

This design requirement calls for several micro-architectural changes in the data flow and control flow of the processor. Allowing a variable number of stages per processor and eliminating global control directly influences the hazard resolution in a typical RISC architecture. Briefly, these issues have been addressed in the proposed architecture, described in [82], as follows:

1. Data-hazard resolution: As in between the original four stages (in the fault-free configuration) new empty pipeline stages may be inserted when substituting faulty parts, the data-hazard resolution cannot be supported with traditional bypassing mechanisms. Instead, we store the produced, yet uncommitted, results in bypass buffers at the stages they are produced and may possibly be reused. These bypass buffers are located in the EX and MEM stages.

2. Control-hazard resolution, Global stalls, pipeline flushing: All three aspects are supported via pipeline flushing in a distributed way. Similar to the StageNet [21], instructions carry an extra bit, read at the IF stage, indicating the flow they belong to. This bit is compared with a similar instruction-flow bit stored at the EX stage. A mismatch prevents the instruction from being executed. In order to flush the pipeline, the bit stored at the EX stage is inverted (e.g. after a branch mis-prediction identified at the EX stage) and all the subsequent instructions are flushed until the equivalent Instruction-flow bit stored at the IF stage is updated.

The architecture presented in [82] is further modified to support fine-grain replacement of a faulty part. This needs the following additions [83]: When implemented in the fine-grain logic, the EX stage requires substantially more area and has longer delay than any other SU of our processor. To reduce this imbalance, we further partition the EX stage into three concurrent sub-units, each roughly containing a chunk of the ALU. Thereby we keep the size of the SUs balanced – as the EX stage occupies more area than the IF, DC or MEM – and consequently reduces the area needed for the fine-grain logic. A second modification is performed to improve the speed of the processor when part of the EX stage is implemented in the fine-grain logic. We add the option to further pipeline the EX stage (breaking the ALU logic in two stages) when part of it is instantiated in the fine-grain logic.

2.4 Probabilistic Analysis of Reconfigurability

Considering the generic design described in Section 2.2, we next present a probabilistic analysis used in our evaluation to assess the fault tolerance of various design options. In particular, we derive the probability of having exactly M fault-free components out of N ($P_{ff}(N, M)$) as well as the probability of having at least M fault-free components out of N ($P_{\geq}(N, M)$) for different numbers of faults in area A for two different design approaches: (i) coarse-grain reconfigurability[1] and (ii) a mix of coarse- and fine-grain reconfigurability which offers the option of a faulty SU to be replaced either by an identical spare unit (coarse-grain replacement) or by fine-grain logic.

Assuming that an area A is divided into N SUs of equal size and that there are k faults in total in the area with a uniform random distribution, then the probability of a SU to have exactly i out of the k faults is:

$$P_{f=i}(k, N) = \binom{k}{i} \times \left(\frac{1}{N}\right)^i \times \left(1 - \frac{1}{N}\right)^{k-i} \tag{2.1}$$

That is, there are $\binom{k}{i}$ combinations of having exactly i out of k faults located in one SU (and exactly $k - i$ faults located in the rest of area A) and the probability of each case is equal to $\left(\frac{1}{N}\right)^i \times (1 - \frac{1}{N})^{k-i}$.

Then, the probability of a SU to be faulty, $P_{sf}(k, N)$, is equal to the sum of probabilities of having $1, 2, ..., k$ faults:

$$P_{sf}(k, N) = \sum_{i=1}^{k} P_{f=i}(k, N) \tag{2.2}$$

Considering that the P_{sf}, for a specific number of faults[2] k in the area, of a single SU is known based on the equation 2.2, it is possible to calculate the probability of having exactly M fault-free SUs out of N, denoted as $P_{ff}(N, M)$ [84]:

$$P_{ff}(N, M) = \binom{N}{M} \times P_{sf}^{N-M} \times (1 - P_{sf})^M \tag{2.3}$$

where $\binom{N}{M}$ enumerates possible ways of having *exactly* M non-faulty substitutable units out of N and $P_{sf}^{N-M} \times (1 - P_{sf})^M$ is the combined probability of having $(N - M)$ faulty and M non-faulty SUs. It is also possible to expand this formula to get the probability of

[1] Component redundancy is an extreme case of coarse-grain reconfigurability where the entire component is a SU.

[2] Henceforth, k is omitted from the equations, we consider that the analysis is continued for a specific k and repeated for the range of faults considered in the evaluation, e.g. 0 to 20 faults.

having *at least* M non-faulty SUs out of N for a specific P_{sf}, here indicated as $P_{\geq}$:

$$P_{\geq}(N,M) = \sum_{i=M}^{N} P_{ff}(N,i) \tag{2.4}$$

which is the sum of probabilities of having $i = \{M, \cdots, N\}$ fault-free SUs [84].

In a design with coarse-grain reconfigurability, each component is divided into a specific number of SUs. Considering the example illustrated in Figure 2.1 which depicts four components each of which are divided into four SUs, it can be observed that for having a fault-free (working) component, there should be at least one fault-free SU at each column available. Therefore, the number of fault-free components that can be constructed is defined by the minimum number of fault-free SUs in each column. For calculating the probability of having a specific number of fault-free components, we first calculate the probability of having at least M fault-free SUs out of N in one column, which is derived by equation 2.4:

$$P_{\geq}^{col}(N,M) = P_{\geq}(N,M) \tag{2.5}$$

Expanding this formula over the total number of columns will result in the probability of having at least M SUs in each column, which is equal to the probability of having at least M fault-free components available in a coarse-grain (cg) reconfigurable design:

$$P_{\geq}^{cg}(N,M) = \left(P_{\geq}^{col}(N,M)\right)^{c} \tag{2.6}$$

The exponent c is the total number of columns, which also defines the coarse-grain granularity[3]. In order to find the probability of having exactly M fault-free components in this case, we exclude from the probability of having at least M fault-free components ($P_{\geq}^{cg}(N,M)$) the probability of having at least $(M+1)$ fault-free components ($P_{\geq}^{cg}(N,M+1)$):

$$P_{ff}^{cg}(N,M) = P_{\geq}^{cg}(N,M) - P_{\geq}^{cg}(N,M+1) \tag{2.7}$$

As mentioned in the previous sections, fine-grain logic can also be used in order to instantiate different SUs and increase the resilience in the presence of permanent faults. We can calculate the probability of having a specific number of fault-free components when both fine-grain and coarse-grain reconfigurability is employed by modifying the probabilities for coarse-grain designs derived above.

Figure 2.1 can be used again as an example that depicts a design with four components, each divided into four SUs, having both fine- and coarse-grain reconfigurability. When only coarse-grain replacement is used, having two faulty SUs on one column and at most one in the remaining columns limits the number of fault-free components to two. In a

[3]The number of SUs in a component.

coarse-grain design the above event is included in the probability of having exactly two fault-free components. Introducing fine-grain logic to replace one faulty SU, rescues one additional component in the above example. Consequently, this event (two faulty units at one column and at most one in the remaining columns) should be included in the list of events which are counted under the probability of having three fault-free components and should be removed from the events that are considered under the probability of having two fault-free components.

Finding events that should be removed or added to the “coarse-grain" probability depends on the number of SUs that can fit inside the fine-grain logic. In this work we only consider the case where the fine-grain can be used to replace exactly one faulty SU. As all events are independent, it is possible to separately compute their probability and append it to the $P_{ff}^{cg}(N,M)$ of equation 2.7. $P_{append}^{+}(N,M)$ denotes the probability of events originally considered in the probability of having $(M-1)$ fault-free components but when using fine-grain logic are becoming part of the probability for M fault-free components. Similarly, $P_{append}^{-}(N,M)$ is the probability of events originally counted in $P_{ff}^{cg}(N,M)$, but using fine-grain replacement moves them to the probability of having $(M+1)$ fault-free components.

$$P_{append}^{+}(N,M) = \binom{c}{1} \times [P_{ff}(N,M-1)]^1 \times [P_{\geq}(N,M)]^{c-1} \tag{2.8}$$

where c is the number of columns, $\binom{c}{1}$ enumerates all possible choices of one column out of c, $[P_{ff}(N,M-1)]^1$ is the probability of having exactly $(M-1)$ fault-free SUs out of N in one column and $[P_{\geq}(N,M)]^{c-1}$ is the probability of having at least M fault-free SUs in $(c-1)$ columns. Similarly, $P_{append}^{-}(N,M)$ can be calculated:

$$P_{append}^{-}(N,M) = \binom{c}{1} \times [P_{ff}(N,M)]^1 \times [P_{\geq}(N,M+1)]^{c-1} \tag{2.9}$$

where $[P_{ff}(N,M)]^1 \times [P_{\geq}(N,M+1)]^{c-1}$ is the probability of having exactly M fault-free SUs in one column and at least $(M+1)$ fault-free SUs in $(c-1)$ columns. Then, the probability of having exactly M fault-free components out of N for a design which uses fine-grain logic in addition to coarse-grain replacement is:

$$\begin{aligned} P_{ff}^{cg+fg}(N,M) =& P_{ff}^{cg}(N,M) \\ &+ P_{append}^{+}(N,M) - P_{append}^{-}(N,M) \end{aligned} \tag{2.10}$$

Similar to the previous cases, in order to find the probability of having at least M fault-free components out of N in a design with both fine and coarse-grain reconfigurability, it is possible to add all probabilities of having exactly $M, M+1, ..., N$ fault-free

components:

$$P_{\geq}^{cg+fg}(N,M) = \sum_{i=M}^{N} P_{ff}^{cg+fg}(N,i) \tag{2.11}$$

The probabilities of having at least M fault-free components out of N in the different design approaches considered ($P_{\geq}^{cg}(N,M)$, $P_{\geq}^{cg+fg}(N,M)$) are used in the evaluation section to measure the probability of a design to guarantee a particular availability of components.

The probabilities of having exactly M fault-free components out of N in the considered design alternatives ($P_{ff}^{cg}(N,M)$, $P_{ff}^{cg+fg}(N,M)$) are used to evaluate the average number of fault-free components in a given area as described below. For a specific number of faults, k, the above probability ($P_{ff}^{j}(N,M)$, where j is "cg" or "$cg+fg$") is used to calculate the average number of fault-free components, $\overline{x}$, as the weighted average of the individual probabilities:

$$\overline{x} = \sum_{i=1}^{N} P_{ff}^{j}(N,i) \times i \tag{2.12}$$

where N is total number of components.

2.5 Evaluation

In this section, we first measure the reconfigurability overheads based on our use-case RISC processor. Moreover, the effect of pipelining reconfiguration interconnects is explored based on post place and route measurements. Finally we evaluate the fault tolerance of various reconfigurable designs using our probabilistic analysis.

2.5.1 Reconfigurability Overheads

Our use-case component is implemented using 65nm STM technology for the ASIC parts (an array of RISC processor and reconfigurable interconnects) and the Xilinx Virtex-5 65nm FPGA substrate for the fine-grain logic [83]. The area overhead of the fine-grain logic required to implement a SU is roughly 6× the area of the same unit implemented in ASIC[4] technology. The area overhead of the reconfigurable interconnects is measured considering a coarse-grain reconfigurable array of four processors, each divided in four SUs (pipeline stages); in that design reconfigurable interconnects add 12.8% more area to each processor [82]. In order to estimate the overhead of reconfigurable interconnects for coarser and finer granularities[5], we use Rent's rule [85] to estimate the number of

[4] In order to keep the area requirements of the fine-grain logic low, we consider that memory blocks, pipeline registers and buffers are implemented in ASIC, which is more area-efficient than in an FPGA implementation. Consequently, even when using a fine-grain block, only logic is mapped to FPGA logic cells.

[5] Number of SUs a component is divided into.

interconnects per SU, as follows:

$$W = K \times (N_p)^{\beta} \tag{2.13}$$

W, is the estimated number of interconnects for a component, K is the average number of interconnects per SU, N_p is the total number of SUs in the component and β is the Rent's exponent which is normally $0.5 \leq \beta \leq 0.7$. In this study, we consider $\beta = 0.5$, which is a typical value for a microprocessor design. The estimated overheads are used in order to define the number of components which can be accommodated in a fixed area for each case of this analysis.

It should be noted that the reconfigurable RISC processor presented in Section 2.3, operates at 450 MHz when considering only coarse-grain redundancy. This is 10% lower compared to the same baseline processor before the micro-architectural modifications required to make its stages interchangeable[6]. Finally, the slowest SU when instantiated in fine-grain logic operates at 200 MHz, which limits any processor configuration using the fine-grain reconfigurable part to the same frequency.

2.5.2 Evaluating the Effect of Pipelining the Reconfigurable Interconnects

This section explores the performance impact of using pipelined interconnects versus non-pipelined ones. We describe our experimental setup, used benchmarks and designs under study. Subsequently, the results are presented and evaluated and a comparison is performed. Pipelining maintains a high frequency but increases the number of stages in an adaptive processor, thereby potentially increasing the number of cycles for executing a program.

For our experiments, we acquired an RTL implementation of the processor using a high level description language (Lisa 2.0) through the Synopsys Processor and Compiler Designer tool. Different designs are then synthesized using Cadence RTL compiler and place & route is performed using Cadence SOC encounter.

Our evaluation is carried out by employing EEMBC CoreMark and Telebench 1.1[7] benchmarks [86]. Furthermore, a set of small custom benchmarks is created with the main purpose of evaluating the processor under extreme cases. These benchmarks are small C programs, running in loops and are briefly described as follows:

- **Function-Argument Heavy:** An application with large number of instructions prone to data hazards between memory operation and arithmetic.

- **Heavy Read-After-Write Conflict:** An application with large number of instructions prone to read after write (RAW) hazards.

[6]The original baseline RISC processor has an operating frequency of 500 MHz

[7]The FFT benchmark is not included as it requires floating point operations, which the under study processors do not currently support.

- **Heavy Branch-Miss-predict:** An application with large number of non-taken branches.
- **Normal-C for-loop:** Code with typical C for-loop.

Our evaluation is performed for variants of our adaptive processor with coarse-grain (CG) reconfiguration. In addition to the baseline processor, we have defined different versions of the fault-tolerant (FT) design as follows.

- **FT Design w/o switches:** The adaptive processor without switches for reconfiguration. This case is used to study the effect of micro-architectural changes inside the processor.
- **FT Design w/o registers:** The adaptive processor, being part of a 4-core CMP, without registers on the reconfigurable interconnects. This case is used to study the benefits of pipelined interconnects.
- **FT Design w/ registers every 2 cores:** In this case, considering the adaptive processor in a 4-core CMP, reconfigurable interconnects are pipelined at every other processor. This design is an intermediate case between two extremes of having fully pipelined and no-pipelined interconnects.
- **FT Design fault-free:** The adaptive processor, being part of a 4-core CMP, considering the fault-free scenario. Interconnects are fully pipelined. This case is illustrated in Figure 2.3a.
- **FT Design 2 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 2 extra stages, imposed by reconfiguration. The two extra stages are result of using one stage of a neighboring core to construct a processor as shown in Figure 2.3b.
- **FT Design 6 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 6 extra stages, imposed by reconfiguration. The six extra stages are result of using one stage of the farthest core to construct a processor as shown in Figure 2.3c.
- **FT Design 12 extra stages:** The adaptive processor, being part of a 4-core CMP with fully pipelined interconnects, with 12 extra stages, imposed by reconfiguration. The twelve extra stages are result of having an extreme case where every stage is connected to the farthest next stage to construct one processor as shown in Figure 2.3d.

It is worth noting that the overall efficiency of the CMP depends on the occurrence rate of various configurations. The configurations studied here are enforced by the density and location of faults and thus it is possible to calculate the probability of each of these cases to happen. With the fault density of one fault per baseline core, 38%,

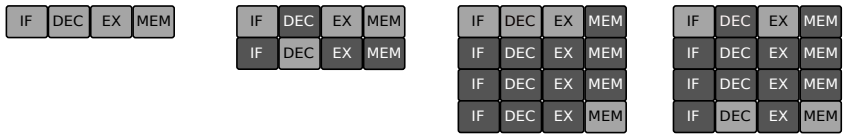


Figure 2.3: Four different configurations of the adaptive processor, being part of a 4-core CMP with fully pipelined interconnects: a. fault-free configuration, b. replacing the DEC stage with the one from the neighboring processor creates two extra stages, c. replacing the MEM stage with the one in the farthest processor creates 6 extra (empty) stages, d. Worst case scenario where the DEC and MEM stages are replaced with the ones three processors away creating 12 extra (empty) stages.

60%, and 2%, of the configurations in a CG design have zero, 1-4 and 5-12 extra stages, respectively [82].

We first evaluate the performance of the above designs measuring their operating frequency, number of clock cycles and absolute time for executing the benchmarks used. Table 2.1 presents the operating frequency of the baseline and the four FT designs. Whilst the clock period of the baseline design is measured at 2.5ns, the micro-architectural changes required for tolerating a variable pipeline depth (due to pipelined reconfigurable interconnects) introduce 16.4% increase in delay, yielding a clock period of 2.91ns (FT design w/o switches). The fault-free configuration of the adaptive processor incurs a 41% overhead compared to the baseline having a cycle time of 3.53ns. A 4-core FT CMP with non-pipelined reconfigurable wires has an increased critical path delay of 8.53ns, that is about 3.5× longer than the baseline and 2.4× compared to the FT CMP with pipelined reconfigurable interconnects. Finally, the design with pipeline registers at every other core has a clock period of 7.1ns.

Table 2.1: Place and Route Timing measurements for our CMP

Design	**Clock Period (ns)**	**Overhead**	
		vs Baseline	**vs Adaptive Processor w/o switches**
Baseline	2.5	-	-
FT Design w/o switches	2.91	16.4%	-
FT Design (fault-free)	3.53	41%	21.3%
FT Design w/o registers	8.53	241%	193%
FT Design w/ registers every 2 cores	7.1	184%	143%

Figure 2.4 shows the measured execution cycles of the baseline, FT design with no faults and FT designs with 2, 6 and 12 extra stages for the EEMBC benchmarks. All values are normalized to the baseline. The FT design with no faults requires the same number of cycles as the baseline processor and the FT core with non-pipelined

reconfigurable interconnects. However, configurations that add 2, 6 and 12 extra stages (due to pipelined interconnects) increase the number of execution cycles by 18%, 37% and 100%, respectively. We performed the same evaluation using our custom benchmarks. As depicted in Figure 2.5 the fault-free configuration of the FT design introduces a small overhead in terms of cycles. This overhead is more noticeable for the Argument-Heavy benchmark (7.5% increase), which is explained by frequent use of the flush/reload mechanism in this benchmark. FT design with 2 extra stages exhibit acceptable performance with around 20% overhead in cycle count. Furthermore, the two extreme cases of having 6 and 12 extra stages, increase the number of execution cycles by 1.8× and 2.5×, respectively.

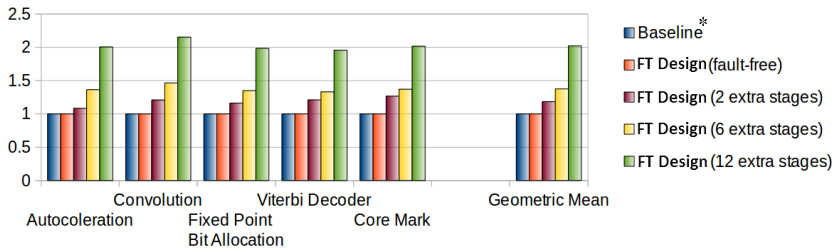


Figure 2.4: Execution cycles for EEMBC benchmarks normalized to the baseline. (*) The execution cycles for the design with non-pipelined interconnects (w/o registers) are considered to be similar to the baseline.

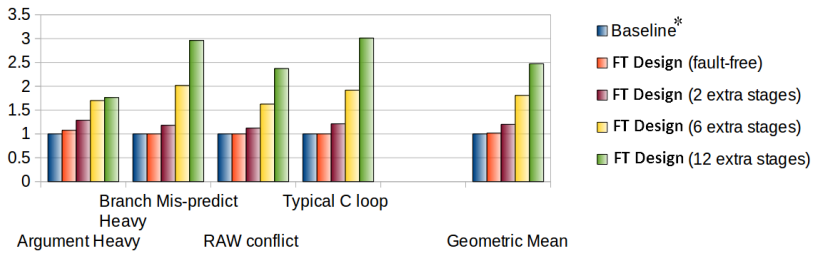


Figure 2.5: Execution cycles for each custom benchmark normalized to the baseline. (*) The execution cycles for the design with non-pipelined interconnects (w/o registers) are considered to be similar to the baseline.

In the next step, combining the measured operating frequencies with execution cycles, we calculate the overall execution time of each benchmark. Figure 2.6, depict the results for EEMBC benchmarks. The results clearly indicate that the FT design without pipelined interconnects suffers from the longest execution time which is almost 3.5× of the baseline. Using registers at every other core just slightly reduces the performance overhead to nearly 3× of the baseline. Moreover, the fault-free configuration of the fully pipelined adaptive processor increases the execution time by 1.4×, due to the lower clock frequency.

The configurations with 2, 6 and 12 extra stages achieve an execution time that is roughly 1.7×, 1.9× and 2.8× higher than the baseline, respectively. Our custom benchmarks show almost the same results for the execution time as depicted in Figure 2.7. The overhead of execution time for the designs with 6 and 12 extra stages degrades to 2.6× and 3.5× of the baseline, respectively.

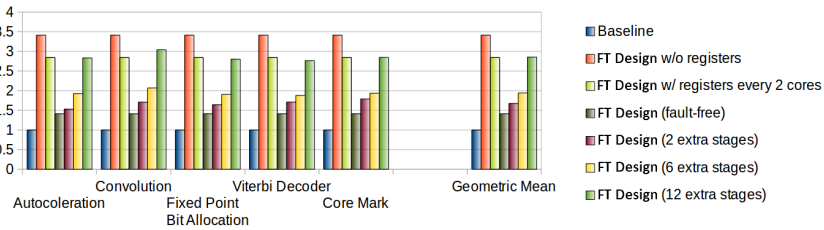


Figure 2.6: Execution time for EEMBC benchmarks normalized to baseline values.

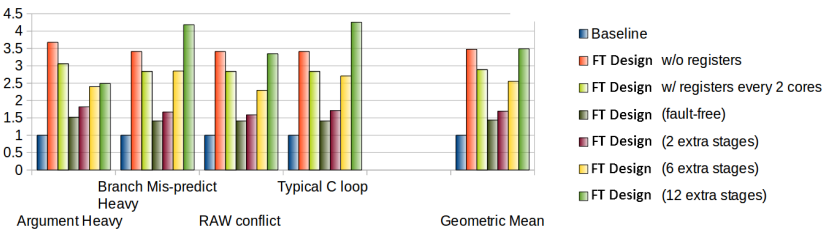


Figure 2.7: Execution time for each custom benchmark normalized to baseline values.

The power consumption of our designs is shown in Figures 2.8 and 2.9 for the EEMBC and the custom benchmarks, respectively. The power consumption of the FT design in the fault-free configuration is similar to the baseline, while for configurations with 2, 6, and 12 extra stages power consumption increases up to 26% for EEMBC and 39% for the custom benchmarks. The FT design with non-pipelined interconnects consumes only 40% of the baseline power due to their lower operating frequency, while adding registers every other core consumes 49% of the baseline power.

Figures 2.10 and 2.11 depict the energy consumption of the two benchmark sets. The FT design in fault-free configuration, requires 42% more energy than the baseline. The FT design with non-pipelined and semi-pipelined interconnects has 41% higher energy consumption than the baseline. Finally, the FT design configurations with 2, 6 and 12 extra stages consume $1.9 - 4.8\times$ more energy than the baseline.

Furthermore, the area overheads of the microarchitectural modifications and the reconfigurable interconnects with and without pipeline registers are measured. Employing non-pipelined interconnects in the FT Design incurs the area overhead of 3.3% compared

to the baseline. The FT-CMP micro-architectural changes, needed for tolerating variable pipeline depths, increase area costs by 4.3% as shown in Table 2.2. This stems from the bypass mechanisms in EX and MEM, the bypass buffers and the additional logic required by the flush/reload-mechanism for data- and control hazard resolution. Compared to the baseline, adding pipelined reconfigurable interconnects to support sparing of SUs has an area overhead of 14.1%.

Table 2.2: Area cost of an adaptive core (FT Design) with and without reconfigurable interconnects.

Design	**Area in** mm^2
Baseline	0.574
FT Design w/o registers	0.593 (+3.3%)
FT Design w/o switches	0.599 (+4.3%)
FT Design (fully pipelined)	0.655 (+14.1%)

In summary, our experiments showed that even in a 4-core resilient CMP design the interconnects added for sparing the SUs of adaptive processors introduce a considerable delay. Using post place and route results we measured that this delay reduces operating frequency 3.5×. CCA [20] is the only related work that addresses the above overhead applying clock borrowing. However, clock borrowing can save at best half of the original

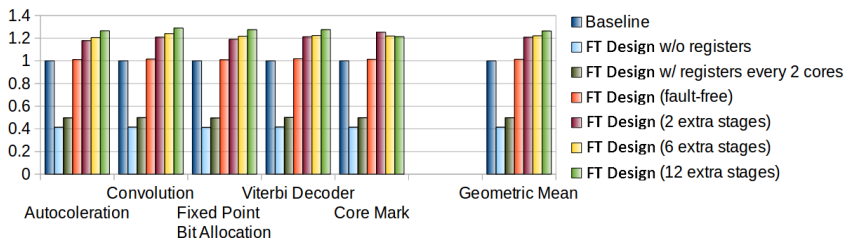


Figure 2.8: Power consumption for EEMBC benchmarks normalized to the baseline.

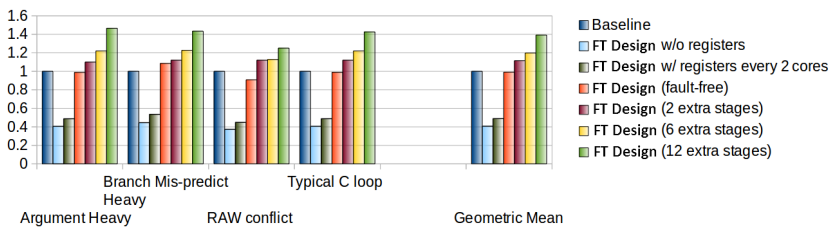


Figure 2.9: Power consumption for each custom benchmark normalized to the baseline.

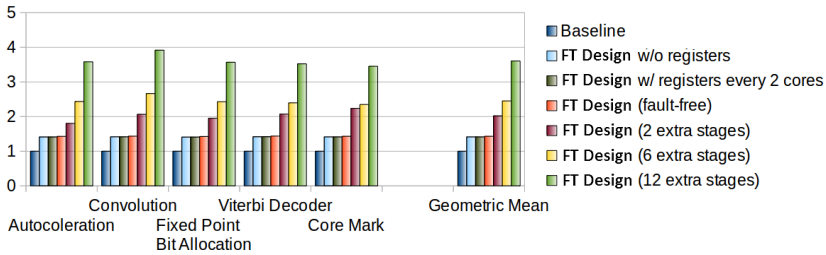


Figure 2.10: Energy consumption for EEMBC benchmarks normalized to the baseline.

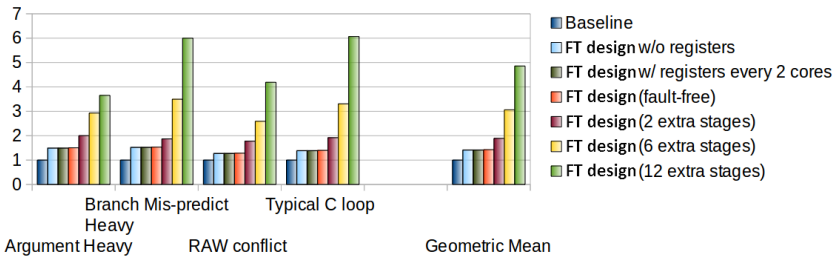


Figure 2.11: Energy consumption for each custom benchmark normalized to the baseline.

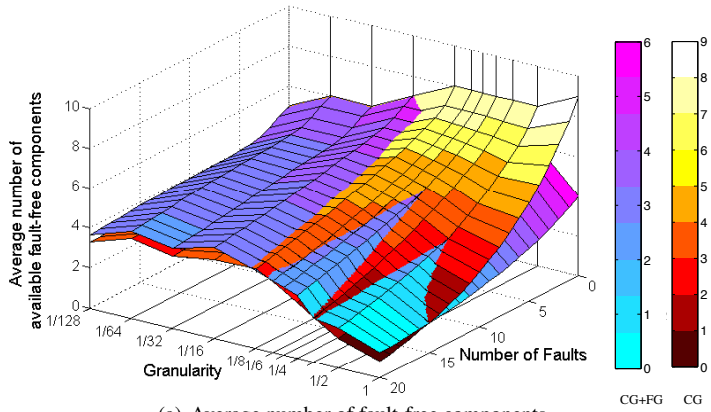
clock period in a design [87]; as shown in Table 2.1, non-pipelined reconfigurable interconnects add more than 6 nsec delay to the baseline 2.5 nsec clock period, making clock borrowing an infeasible option at higher clock rates. We further showed that pipelining the reconfigurable interconnects achieves better performance compared to reducing the clock frequency. The fault-free configuration of an adaptive processor with pipelined reconfigurable interconnects increases the baseline execution time by only 41% and is 2.4× faster than the alternative of scaling down the clock. Even processor configurations with longer pipelines are up to 2.3× faster and in all cases not slower than a design with non-pipelined interconnects. Finally, the energy consumption of these two design alternatives is similar unless multiple (6 to 12) extra-stages are used.

2.5.3 Fault tolerance of Reconfigurable Designs

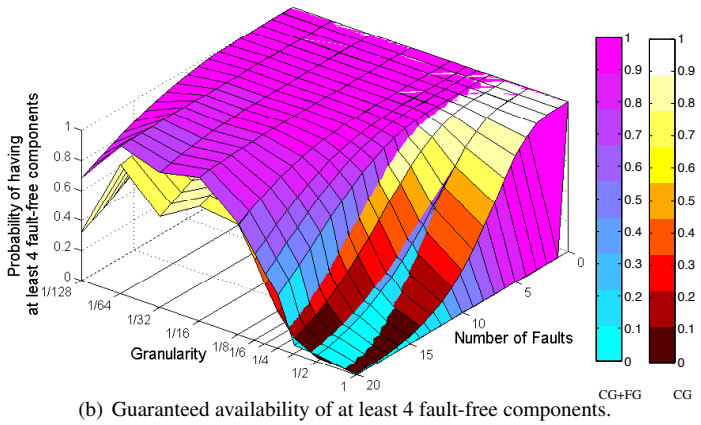
We take into account the area overheads for each reconfigurable design and evaluate their fault tolerance. We consider designs that offer only coarse-grain (CG) reconfiguration or both coarse and fine-grain (CG+FG), varying their granularity (SU size). All designs occupy roughly the same area[8] and therefore, due to their particular resource overheads, each design fits a different number of components in the fault-free case. We consider a fault density that causes up to 20 faults in the silicon area. This translates, even for our

[8]The silicon area constraint in our design-space exploration is equal to that of 9 baseline components. A baseline component is a component with granularity equal to 1 (the entire component is considered as one SU).

small use-case microprocessor, to a probability of a single transistor to fail to be up to 2.2×10^{-6}.



(a) Average number of fault-free components.



(b) Guaranteed availability of at least 4 fault-free components.

Figure 2.12: Average availability and probability of guaranteed availability (4 fault-free components) of coarse-grain (CG) and coarse+fine-grain (CG+FG) reconfigurable designs with various granularities at different fault densities. Granularity is the fraction of a component that constitutes a SU. All designs use area smaller or equal to 9 baseline components; that is a component with granularity equal to 1.

Figure 2.12 depicts the average number of fault-free components that can be constructed in each design at a particular fault density (number of faults), as well as the probability of a design to deliver at least four fault-free components.

The average number of available components is measured as explained in Section 2.4 based on equation 2.12 by analytically calculating the probability of constructing correctly functioning components for the given reconfiguration options offered by each design. Figure 2.12(a) shows that for low number of faults ($\leq$2), component redundancy provides a higher number of fault-free components, as the area overheads of reconfigurability are

not capitalized yet. For higher number of faults (≤ 8), coarse-grain reconfigurability of granularity $\frac{1}{8}$ (CG($\frac{1}{8}$)) maximizes availability of components. Adding fine-grain logic at smaller SU sizes (CG+FG($\frac{1}{16}$)) is up to 13.5% better for fault densities beyond that point. Even at high number of faults, CG+FG($\frac{1}{16}$) provides almost five available components, tolerating over $3\times$ and $2\times$ more faults than component-redundancy (CG(1)) and other CG points, respectively.

Guaranteed availability is measured using equations 2.6 and 2.11 as the probability of a design to deliver at least 4 fault-free components at a particular fault density. As shown in Figure 2.12(b), the probability for the CG+FG($\frac{1}{16}$) design does not drop below 99% for up to 20 faults. On the contrary, component redundancy (CG(1)) is below 90% and 50% beyond 6 and 10 faults, respectively. Finally, the best CG granularity crosses the threshold of 90% for more than 17 faults.

2.6 Conclusions

A probabilistic analysis was presented for estimating the number of fault-free components that can be constructed in various reconfigurable designs at the presence of permanent faults. Adding fine-grain logic in a design can increase availability, tolerating $3\times$ more faults than mere component redundancy. Different fault densities require different granularities of substitutable component-parts in order to maximize fault tolerance. Component redundancy is better for a low number of faults. As the number of faults increases, coarse-grain and mixed-grain reconfigurability (of granularity $\frac{1}{8}$ and $\frac{1}{16}$, respectively) provide the best availability.

Moreover, the effect of pipelining the reconfiguration interconnects was explored, in terms of operating frequency and execution time. The results showed that in the processor with no pipelined interconnects, the frequency and execution time both increase by almost $3.5\times$, compared to the baseline. Adding pipeline registers shows significantly better performance where for the fault-free case the overhead of operating frequency and execution time are $1.41\times$ and $1.7\times$.

3
Resilient service-oriented NoC

In a multicore system, a faulty processor can be isolated using runtime mechanisms to redistribute its workload to other available cores. However, tolerating permanent faults at the interconnects can be less simple. A Network-on-Chip (NoC) shares its resources with every component on a chip. It interconnects multiple processing elements, memory blocks and IO and therefore constitutes a single point of failure for an entire System-on-Chip (SoC). A fault-tolerant NoC architecture is therefore critical for the design of a fault-tolerant SoC [1, 88].

The NoC of an advanced embedded system needs to support multiple traffic classes with certain quality of service (QoS) requirements, such as latency and throughput. This is often achieved using virtual channels and priorities among different classes of traffic related, for instance, to control messages or data-exchange. However, many systems require their traffic classes to interfere as little as possible with each other in order to best fit their particular QoS constraints. In the past years, several service-oriented NoCs address this design objective by statically allocating their resources to different traffic classes (services). For example, QNoC provides four separate router data-paths each supporting different services [31], moreover, Tilera's iMesh is composed of five separate NoCs entirely isolating each traffic class of their tiled architecture [30].

This chapter addresses a single challenge pertaining to the design of future reliable multicore systems: the *tolerance of permanent faults on a service-oriented NoC through traffic redirection*. We categorize network resources based on the service they support and provide mechanisms to bypass them when faulty, allowing a traffic class to be redirected. Traffic redirection can be performed either by modifying its routing while sustaining the isolation between different services, or alternatively, by using a router data-path dedicated to a different service, thus allowing service-isolation breach. Network resources that

are common to all services, such as the links and the router control, employ additional mechanisms for fault tolerance. Concisely, the main contributions of this work are the following:

- We extend previous detour techniques, allowing the network to selectively detour traffic per service, rather than detouring all packets to avoid faulty network parts, substantially improving NoC reliability as well as significantly increasing performance.
- Within a router, multiple services can be merged, still preserving their priorities, in order to bypass the damaged data-path of one or multiple services.
- We analyze the trade-offs between the two proposed approaches for redirecting a NoC service, namely (i) using alternative resources of the same service and (ii) using resources of a other services. In the former approach, lanes of different services are isolated in expense of using longer paths, while in the latter approach, service-isolation is traded for shorter path and higher connectivity. We measure the area and power overheads of the above techniques, evaluate their impact on performance and QoS and estimate their effect on reliability.

In the remainder of this chapter, first related work is presented in Section 3.1. The description of the baseline service-oriented NoC used in this study follows in Section 3.2. Then, Section 3.3 details the proposed mechanisms for tolerating permanent faults on the NoC. Section 3.4 evaluates the fault tolerance of our approach, measures its area, power and performance overheads, and compares with related work. Finally, in Section 3.5 overall conclusions are drawn.

3.1 Related Work

In the past a plethora of techniques, some of them summarized in [89], have been suggested to combat permanent faults in Networks-on-Chip. They are applied at different levels of a NoC design: ranging from entire network regions (entire routers or links) down to individual wires of a link, a single buffer or crossbar of a router. Some techniques modify the routing of packets to bypass faulty links and routers, others divide routers or links to smaller parts and isolate, spare them or share them to tolerate permanent faults. In general, the finer the granularity of the considered fault-model the better the fault tolerance, but the higher the design overhead. Bellow, a brief overview of such techniques is presented elaborating more on the techniques most relevant to our approach.

Many approaches modify the routing algorithm of a NoC to be dynamically adaptive and avoid faulty links or routers [32–36, 38]. Faults on links or routers are broadcasted to all nodes in order to update their routing tables and avoid damaged network parts [34]. Packets are sent across multiple non-intersecting paths to ensure that they are delivered correctly [32]. Feng et al. [33] introduced deflection routing where the packets are routed based on a defect map of the neighboring links and switches. Maze is new fully-distributed, low-cost fault-tolerant routing which guarantees packet delivery in presence

of a physical path [36]. In the works presented by Parikh et al. [35] and and Aisopos et al. [38], routing tables and routing paths are updated, respectively, based on the detected faults in order to avoid faulty components in a distributed way. In general, methods that adapt the routing according to the faulty components of the network introduced low area and power overheads ($\sim$5% in Murali et al. [32]), however they provide limited fault tolerance. On the other hand, a routing algorithm based on reinforcement-learning improves fault tolerance but suffers from high area and power overheads (90% and 120%, respectively) [33].

Other techniques re-route packets without modifying the actual routing algorithm. Instead, they perform a detour selecting an intermediate destination for packets that would normally need to pass through some faulty links or routers. Although detour does not change the routing algorithm, deadlocks need to be reconsidered in case packets are not stored entirely in the intermediate destination. ReliNoC [47] performs detour using a Logic-based Distributed Routing table proposed by Rodrigo et al. [39] . Routing reconfiguration through intermediate destinations uses the turn model to avoid deadlocks in Fick et al. [40]. Han and Fu [41] have also described a detouring technique for a 2D mesh with XY-routing and avoid deadlocks considering the turn-model. Furthermore, Vitkovskiy et al. [42] proposed a technique where detour is handled locally at network regions that suffer from faulty resources; a packet entering such region is redirected through an alternative path. In general, detour introduces low area and power overheads (less than 10%), but the performance decreases rapidly with increasing number of faults, since even partially faulty components are omitted from network resources.

Besides rerouting of packets, faulty network parts can be bypassed in hardware. An alternative path from an input port to the output is employed to bypass a faulty router by Koibuchi at al. [43]. A faulty crossbar is bypassed in a similar manner in Vicis [44]. Finally, a single input buffer can be bypassed in RoCo [45]. In order to contain a fault in the router, RoCo additionally splits the router in two distinct parts responsible for either row or column traversal of packets.

Evripidou et al. [46] exploited the redundancy offered by Virtual Channels (VC) and proposed renaming of the VC buffers. They modified the input ports of a router to be able to avoid faulty buffers and redirect traffic classes through the remaining available VCs. ReliNoC considers a NoC that has two separate physical networks one for traffic that requires some QoS and one for normal traffic [47]. ReliNoC considers that the two physical networks have interchangeable links and routers. In essence, a faulty link or router of a physical network can be replaced by the equivalent part of the second network. As a consequence, upon the occurrence of permanent faults some links or routers need to accommodate both traffic classes.

At the router level, Vicis reduces the impact of faulty input and output ports using port swapping [44]. This technique allows to pair on-demand input and output ports of neighboring routers in order to increase the connectivity of the network at the presence of faults. Furthermore, Liu et al. [48] partition links and router data-path in four slices, then, only the fault-free slices are used in a time-division multiplexing manner. Besides the performance overhead, slicing has a significant area cost (about 65%).

The above approach by Liu et al. [48] allows, among others, to use partly faulty links. Another approach for protecting links is proposed by Lehntonen et al. [90] ; they use spare wires at the links to replace faulty ones. Moreover, Vitkovskiy et al. [42] used shifting and retransmission of flits to communicate correctly data across partially faulty links. In doing so, they can tolerate up to 50% faulty wires on a link, suffering however, the performance overhead of one retransmission. Both approaches have high area cost (about 75% and 30%, respectively).

Finally, BulletProof router introduced by Constantinides et al. [1] described a method to automatically inject sparing logic to the netlist of a router and be able to replace faulty circuitry. However, this method has a substantial area overhead of up to 3.4×.

Besides ReliNoC, none of the above techniques is addressing fault tolerance on a service-oriented NoC nor analyzes the affect of faults to the performance of individual traffic classes. Even ReliNoC considers only two classes of traffic (one for QoS and one for best-effort), not discussing scalability to a higher number of services. Supporting only one QoS traffic class together with best-effort traffic is expected to simplify arbitration and flow control; presumably this is the reason why these aspects of ReliNoC are not elaborated [47]. Moreover, Virtual Channel renaming has some similarities with the proposed Service Merge [46], however, it can be applied to a NoC with VCs rather than a service oriented NoC with separate router data-paths such as the QNoC. VC renaming has complex buffer management that uses pointers to stored packets and suffers from fragmentation. It further increases the minimum buffer size to allow multiple virtual channels on a physical one, while priorities are fixed between physical buffers. Finally, VC renaming protects only the input buffers rather than an entire router data-path. On the contrary, this work offers a complete solution for tolerating permanent faults on a service-oriented NoC, which is scalable to multiple traffic classes, presenting flow control and credit handling modifications to support service redirection. As shown in Section 3.3, RQNoC does not affect the input buffers and offers the flexibility to maintain traffic class priorities despite the physical datapath they use. It should be mentioned that the hardware techniques employed in the design of RQNoC are intended to increase the availability of network resources and thus, are orthogonal to fault-tolerant routing algorithms that guarantee deal-lock free packet delivery.

3.2 The QNoC Architecture

We use the QNoC architecture as our basis to apply our fault-tolerant techniques [31]. QNoC can be positioned between two extreme approaches for providing service-oriented interconnects. The first one is using different virtual channels for different traffic classes. At the other end of the spectrum, iMesh uses separate physical networks per service [30]. iMesh may not increase the area cost of the routers compared to a network with virtual channels and the same buffersize[1] and wiring of separate links per service may be cheap in current technologies [92], however, the power consumption of a NoC with multiple

[1] that is due to the fact that buffers occupy the largest part of a NoC router [91].

links is expected to be increased. QNoC offers separate router data-paths per service however links are shared between all services and are allocated using priority arbitration at the output ports. Gilbert et al. [93] showed that a NoC architecture with shared links and separate router data-paths per service can be more efficient in terms of area, power and performance compared to using virtual channels.

Table 3.1: Specification of four considered traffic classes

Traffic Classes	**Traffic Classes**	**Average Packet Size [flits]**	**Proirity**
Signaling (Control signals, interrupts)	5% (Very Low)	5 (Very small)	Very High
Real-Time data (Streaming)	15% (Medium)	20 (Large)	High
Single Read/Write	10% (Low)	10 (Medium)	Low
Block Read/Write	70% (High)	20 (Large)	Very Low

QNoC is a packet switched network composed of routers with service flow control interconnected on a 2-D mesh topology. It uses XY routing with multi-class wormhole routing [31]. It supports four distinct services (SVCs) to accommodate equal number of traffic classes for *signaling*, *real-time* data, *single read/write* (short data access) and *block read/write* (large data access) as shown in Table 3.1. Traffic parameters are chosen based on requirements and specification of the particular MPSoC, used in the DeSyRe research project [94]. Signaling traffic includes critical control messages and interrupts, these are infrequent short packets that require low latency and high priority. Consequently, in our setup signaling traffic is 5% of the total traffic, uses 5-flit packets and has the highest priority. The second highest priority class is dedicated to real-time/data streaming. Streaming/dataflow processing requires data to have a continuous flow in order to achieve good processing throughput, consequently large packets of high priority are required. For our systems this represents 15% of the traffic using 20-flit packets. The third traffic class is dedicated to Single Read-Write. These are medium-size (10 flits) packets as they constitute a single memory access. This traffic is not critical for the system hence it is assigned low priority, and it is consider to be 10% of the entire traffic. Finally, the lowest priority is assigned to Block Read-Write traffic which is used for passing large packets of data and constitutes the largest volume of our traffic (70%).

A router connects to each neighbor with two 40-bit links (32-bit data, 8-bit control), one per direction. It further provides four local connections, one per SVC, through a Network Interface (NI) to the local component(s). A hop-by-hop credit based flow control is used; each router sends credits per SVC to its neighbors corresponding to available input-buffer slots. XY routing guarantees deadlock freeness. Starvation is avoided by setting boundaries for maximum traffic in each SVC, i.e. services of high priority are set to have a low upper bound of traffic load and vice versa [31].

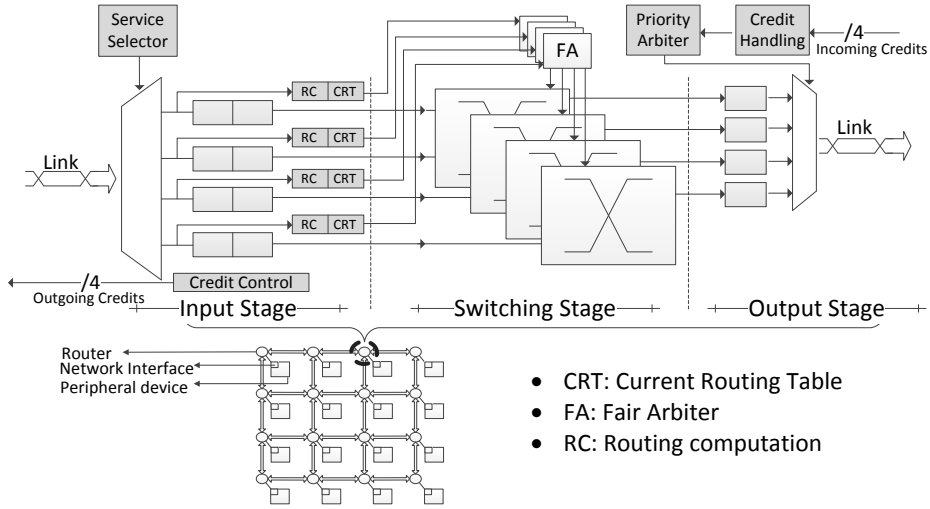


Figure 3.1: The architecture of QNoC.

As illustrated in Figure 3.1, a QNoC router has three stages: input, switching, output. At the input stage, flits are sent to the corresponding input buffer based on their SVC. In case of a header flit, the output port is computed at the Routing Computation (RC) unit and stored for the remaining flits of the same packet in the Current Routing Table (CRT). Subsequently, flits are switched from the input buffer to the selected output port buffer, through a crossbar (one per SVC) according to the CRT information. When multiple packets of a single service, located at different input ports, compete for the same crossbar output, a *round-robin* mechanism is used for fair arbitration (FA). At the output, the output buffer stores flits ready to be sent. The SVCs are then multiplexed and access the output port based on their priority and their available credits. In summary, each SVC has a separate data-path in the router and the only logic shared between them is related to the priority arbiter at the output. In our implementation, flits are 40-bits, input buffers store 2 flits and output buffers store 1 flit.

3.3 RQNoC: a Resilient QoS NoC

Tolerating permanent faults in a service-oriented NoC can be achieved considering each SVC separately. Faulty SVC resources can be bypassed either allowing the respective traffic class to be redirected using an alternative path of the same service, called *Service Detour (SDetour)* or alternatively redirect it through the router data-path of another service, called *Service Merge (SMerge)*. Each of the two options have their advantages and disadvantages. Service Detour prevents services to interfere with each other, but increases the latency of traffic that belongs to the faulty service. Service Merge has a lower performance overhead as the packet route remains the same, however, it allows

services to share the same router data-path and consequently to affect each other's performance. In order to prevent in an RQNoC with SMerge traffic classes with hard QoS requirements to share router-resources with other traffic, we further introduce Service Renaming, which allows dynamic assignment of router data-paths to a particular traffic class. To provide a complete fault-tolerant solution, the remaining network resources that are common for all services need to be protected, too. Links can tolerate faults by adding to a shifting technique spare wires, thereby increasing the number of faulty wires they can sustain. Finally, the common control of a router[2] is protected using Triple Modular Redundancy (TMR). Next, we describe separately each proposed technique.

3.3.1 SDetour: Service Detour

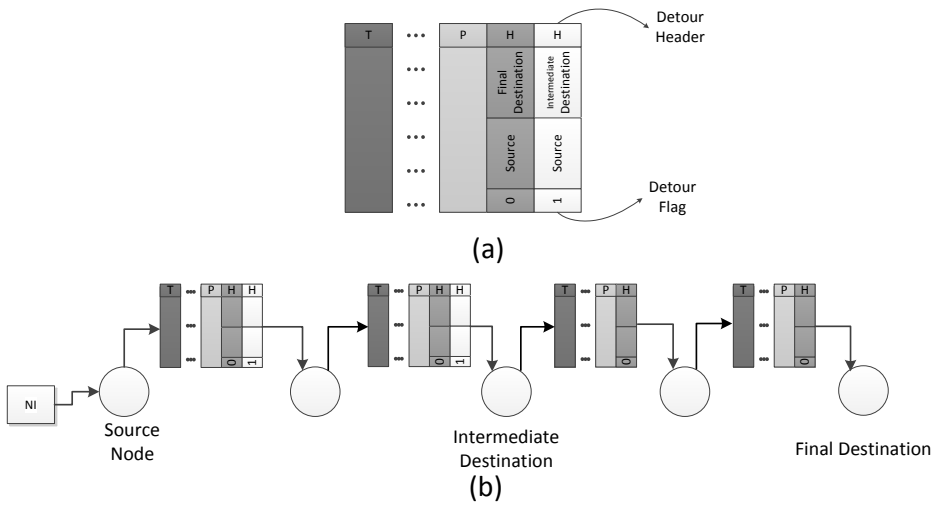


Figure 3.2: General format of a detoured packet (a). After reaching the intermediate destination, the detour header is discarded and for the rest of the path to final destination, the real header will be considered in the routing computation units (b).

Detour is a well-known technique for mitigating permanent faults at links and routers. It allows to bypass faulty network regions without however modifying the routing algorithm. This simplifies the design compared to adaptive routing techniques for fault tolerance. We describe first the general (global) detour mechanism applied in the proposed RQNoC and subsequently explain how it is selectively used per service.

Packets that, according to the routing algorithm, need to traverse a path through damaged links or routers, can still reach their destination taking a detour through an intermediate node. Such cases are detected at the source NI using a detour table, which is computed based on the defect map of the NoC. Before a packet is sent to its destination,

[2]Practically the service selector in the input ports and the priority arbiter in the output ports.

the detour table is consulted to check whether there is a detour to be taken. In such case an additional *detour-header* flit will be added before the original header flit indicating the intermediate node as the first destination.

In addition, one bit in the header is reserved for a *detour flag* that distinguishes detour headers from normal headers. The detour flag of a header flit is checked at each input port of a router. If the detour flag is set and the current node is the detour intermediate destination, then the detour header is discarded at the output of the input buffer. From that point on, the next flit (the original packet header) is considered as the header, as depicted in Figure 3.2. This approach allows detour to scale to multiple intermediate destinations as multiple header flits can be added to a packet and one-by-one be discarded at each intermediate destination.

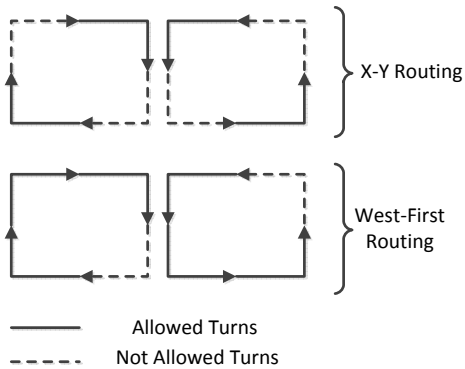


Figure 3.3: Possible turns in X-Y routing and West-First turn model.

There might be several candidates to be used as intermediate nodes. We choose the one that minimizes the path and avoids deadlocks. XY routing is deadlock-free, however detoured packets are not entirely stored-and-forwarded at the intermediate node, for performance reasons, and therefore may create dependency loops. Such loops are avoided considering a turn model. As depicted in Figure 3.3, the additional turns introduced by detour should be the ones of West-First turn model, a subset of which is XY routing. As a consequence, intermediate nodes are selected so that the packets use only the turns allowed by the turn model. The routing algorithm (in our case XY) and the restrictions posed by the turn model may not allow a destination to be reached although it may still be connected to the rest of the network. To exemplify, Figure 3.5 shows that a packet from any node (X,Y), where $Y \leq 2$, cannot be sent to (0,0) when the vertical link between (0,3) and (0,2) is broken, although the node is still connected to the rest of the network; that is because the North-West turn is not allowed by the turn model. This introduces a disadvantage compared to the subsequent SMerge technique described in the next subsection. It should be noted that deadlocks due to detour could be avoided by storing the entire packet at the NI of the intermediate node before forwarding it to its final destination, however this would significantly increase communication latency.

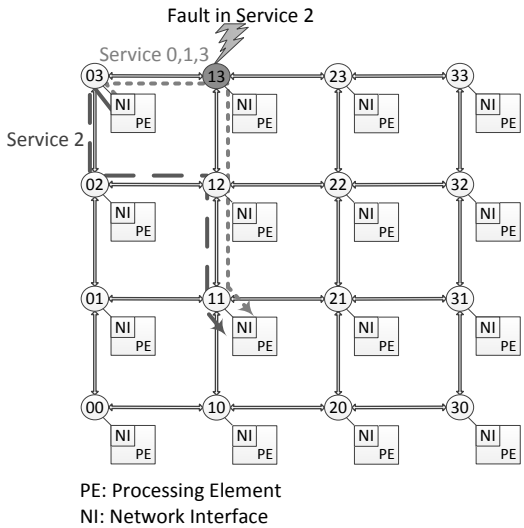


Figure 3.4: Basic concept of SVC detour.

In our approach faults are detected in the granularity of a service (router data path dedicated to a particular SVC). Consequently, upon the detection of a fault, only the traffic class belonging to the faulty SVC needs to be detoured, the remaining traffic can still use the original paths. This is achieved by modifying the detour table at the NI to store intermediate destinations per service. Figure 3.4 illustrates an example of Service Detour as described above. Packets sent from node (0,3) to (1,1) are routed through node (1,3) using XY. In case the data path of service 3 in router (1,3) is faulty, then service 2 is detoured through (0,2). The traffic of the remaining services still enjoy the original route. Service Detour does not introduce any additional turns. As described in the previous paragraphs intermediate nodes are selected according to the West-first turn model.

In the worst case, SDetour doubles the number of hops a packet needs to take from its source to a destination and consequently doubles worst case packet latency. This is taken into account for traffic classes with hard packet latency requirements as follows: an intermediate node is selected that does not exceed the maximum allowed number of hops of a particular traffic class. In case that is not possible then the system should need seek other mitigation techniques described below, or discard (if possible) the respective destination node in order to avoid system failure.

3.3.2 SMerge: Service Merge

The redundant data-paths per service in a QNoC router can be exploited for fault tolerance. As opposed to Service Detour, this approach allows a router with a faulty service to still be used by the respective traffic class redirecting (merging) it to the data-path of another

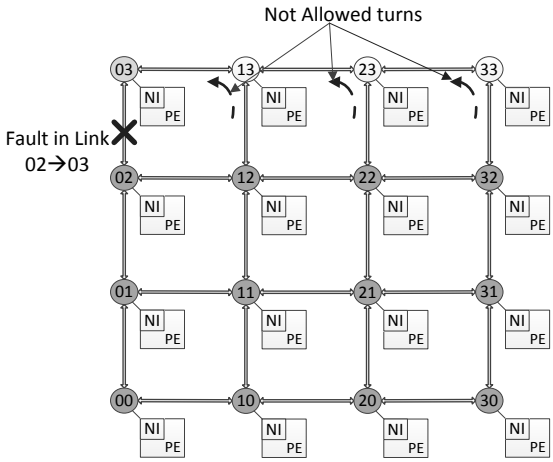


Figure 3.5: Example of a fault not mitigated by detour.

service. Each service data-path in the proposed RQNoC router is able to host the traffic of any other service. Such data-path is called host and the services it hosts are called guests. In general, an RQNoC router can be configured to host any combination of guests in each service data-path. In the extreme case, a single data-path may host all four services.

3.3.2.1 Router modifications

In order to support SMerge, the architecture of the original QNoC router needs to be modified as follows. The *input port* should be able to merge packets of different services to a single data-path and the *output port* to send them out separately based on their original SVCs. In addition, each *neighboring router* needs to be aware of merging decisions and handle correctly the credits received. Different credits correspond to the input buffers of different SVCs, consequently, credits of faulty SVCs should be ignored. The configuration of a router, indicating which services are merged through which data-paths, is encoded in a configuration array (CA). In the following, these modifications are detailed and an example of consecutive routers merging different services is used, as illustrated in Figure 3.6.

At the *input port* of the router, each arriving flit is sent through the SVC multiplexer to its corresponding SVC data-path, in particular to its input buffer. For SMerge, the select of the SVC multiplexer is configurable. In essence, a packet is sent to a particular datapath considering, besides its traffic class, the SMerge configuration of the router ,i.e. CA. Moreover, in order to keep track of the original SVC of the flits and be able to distinguish them at the output, each flit is tagged with its original SVC-id. This adds 2

extra bits in the data-path.

Flits guided to a particular SVC datapath are stored to the corresponding input buffer and subsequently switched to the correct output port as described in the QNoC: RC computes the output port when the header flit arrives and updates the CRT which is used to route the upcoming flits of the packet. Each SVC datapath has its own crossbar which uses fair arbitration. In our current implementation, even when multiple traffic classes are hosted in the same SVC data-path, switching is still performed with fair arbitration.

At the *output port*, flits are stored in the output buffer and wait for permission to be sent out. A flit is sent out based on its priority and the available credits in the corresponding credit counter. The *priority arbiter* is modified to support SMerge. Originally, the id of the output buffers (each corresponding to a service data path) was used to grant access to the link. However, in the RQNoC flits of different traffic classes may share the same data-path. In order to maintain priority arbitration between the original services (even when sharing the same data-path) the arbitration logic is modified to consider the original SVC of the flit using the SVC-id bits added to the datapath.

Additional changes were made so that the neighboring routers can send packets to a faulty router (for correct credit handling). Neighboring routers should consider the merging decisions at the faulty router, discarding credits that belong to faulty paths and consuming credits that correspond to the actual receiving buffer. This is achieved by modifying the *credit handling* module located in the output port and by adding two new modules, namely the *Preemption unit* and the *Counter-mask*.

In credit-based flow control, each SVC has a dedicated credit counter at each output port which keeps track of the credits received from the immediate neighboring router and used in a service. Each counter indicates the available SVC input-buffer space of the next router. The credit counter corresponding to a faulty SVC data-path of a neighboring router should be masked. This is achieved by the *counter mask* module. In practice, the counter mask is a multiplexer that for each output buffer points to the right counter that holds the available credits. This multiplexer is controlled by the SVC-id of the first flit in the buffer and the merging configuration of the next router. In the example of Figure 3.6, the credit counter of SVC2 in router-0 will be ignored and the counter mask module will redirect the priority arbiter to look at the credit counter of SVC3 instead; this is because in router-1 traffic of SVC2 is redirected to the data-path of SVC3. Traffic of SVC3 will still use its original credit counter.

Credit Handler increments and decrements the credit counters based on the credits received and the flits sent. This is performed considering how multiple traffic classes are merged *in the receiving router* as follows:

- received credits for a particular SVC increment the respective counter, unless that SVC is faulty (and disabled) in the receiving router; in that latter case the credits are ignored. In the example of Figure 3.6, credits sent from router-2 to router-1 increment their respective counter normally unless they belong to the faulty SVC-1, in which case they are ignored.

- Sent flits consume credits from the counter that corresponds to the SVC datapath

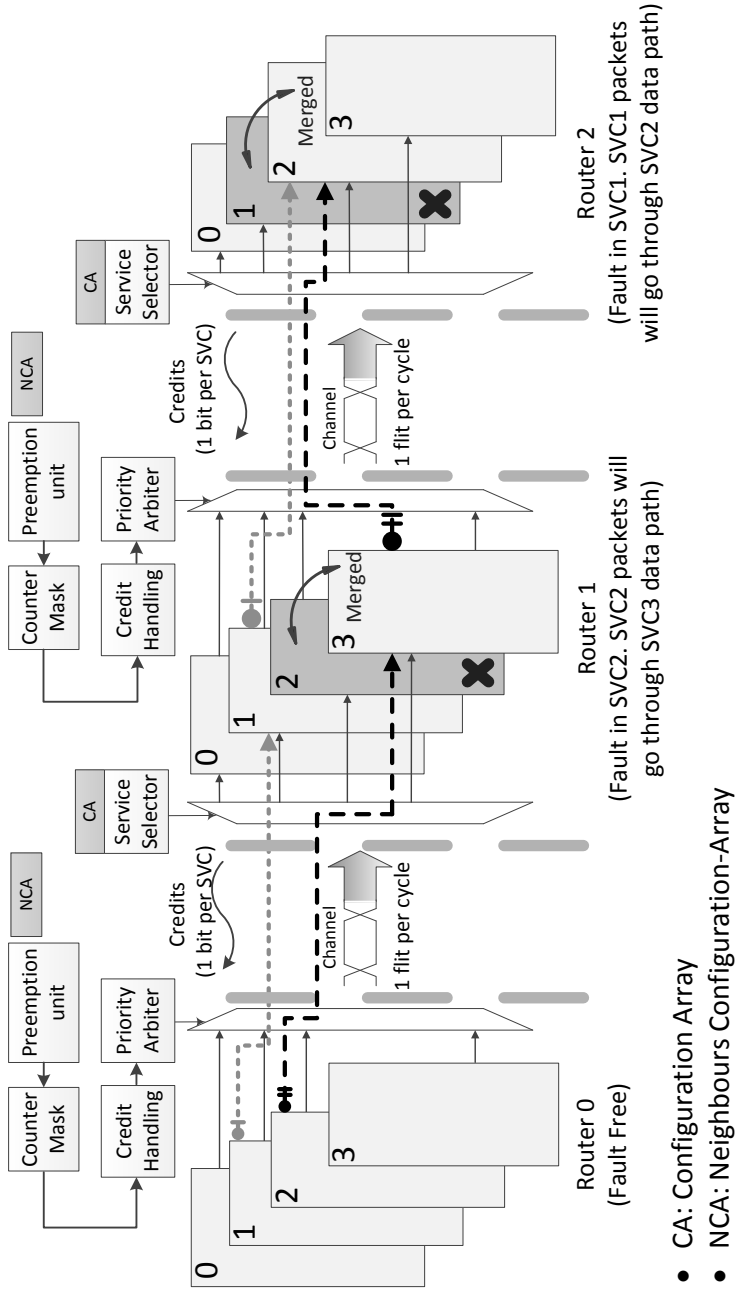


Figure 3.6: An example of Service Merge.

that hosts their traffic class in the receiving router. For example, in router-0 of Figure 3.6, sending a flit of SVC2 will decrement the credit counter of SVC3, as in router-1 it is mapped to the SVC3 data-path.

Finally, it should be noted that flits of different services cannot be interleaved when sharing the same SVC data-path. That is due to a packet-integrity check performed at the input port, which ensures that flits of a packet come in order. Then, mixing packets of different services at the same SVC data-path requires an additional mechanism to ensure that. The *Preemption unit* module, guarantees that a neighboring router will send to a faulty router all flits of a packet that belongs to a specific traffic class, before selecting flits of other class that shares the same SVC data-path. In the example of Figure 3.6, router-0 sends to router-1 an entire packet of SVC2 and then selects flits of SVC3 as both traffics are mapped to the SVC3 datapath of router-1.

An SMerge example: To give a better insight on credit handling we detail further the SMerge configurations of of router-1 and router-2 in Figure 3.6. SVC2 in router-1 is faulty and therefore the traffic classes of service 2 and 3 share the data-path of SVC3. SVC1 in router-2 is faulty which forces traffic classes of service 1 and 2 to share the data-path of SVC2. Figure 3.7(a) shows how the credit counters in router-1 are linked with each traffic class. Each counter corresponds to a particular SVC of the receiving router (in this case router-2). At the output of router-1, the SVC-id of the first flit in each buffer is checked. The *Counter Mask* unit assigns a counter for each flit on an output buffer, based on the flit SVC-id and the *Neighbors Configuration-Array* (NCA). Consequently, in this example the counter-1 in router-1 is always ignored as SVC1 in router-2 is faulty. Then, packets of SVC2 (which are hosted in datapath of SVC3) and packets of SVC1 (in the datapath of SVC1) are mapped to counter-2, as in the next router they share datapath of SVC2. Finally, packets of SVC3 and packets of SVC0 will consult counters 3 and 0, respectively, as they use their original data-paths in router-2. After checking the value of the assigned counters, the *Priority Arbiter* (PA) grants link access to the flit with the highest priority SVC-id, regardless of which buffer it resides in. Afterwards, the *Credit Handling* unit updates counter values based on the id of the fit which is sent to the link, incoming credits and NCA.

Figure 3.7(b) shows a timing diagram of the same example. The initial credit counter values are 0,-,2,2 for counters 0, 1, 2 and 3, respectively. In our example, router-1 receives credits from router-2 only in the first cycle. As mentioned before, counter-1 is isolated and all SVC1 credits from router-2 are ignored. In the first cycle, one credit per service is received from router-2. The highest priority flit is $P0$ in SVC0 and the incoming credit makes it possible for router-1 to send it on the next cycle. Thus, one flit of SVC0 is sent and counter values are updated to be 0,-,3,3; although counter-0 received a credit, it was not incremented as a service-0 flit was sent. In the next cycle, $T0$ in SVC0 cannot be sent as counter-0 has zero credits. For sending $T1$ counter-2 is checked for credits as in router-2 SVC1 is hosted in SVC2. Counter-2 has credits so $T1$ is sent and counter-2 is decremented. In the third cycle, counter-0 is still zero and there is no waiting flit in SVC1. Therefore, the PA sends $T3$ in SVC3 on the link and *Credit Handling* unit decrements counter-3. In the next cycle, the next flit available in SVC3 buffer is $H2$. Based on the

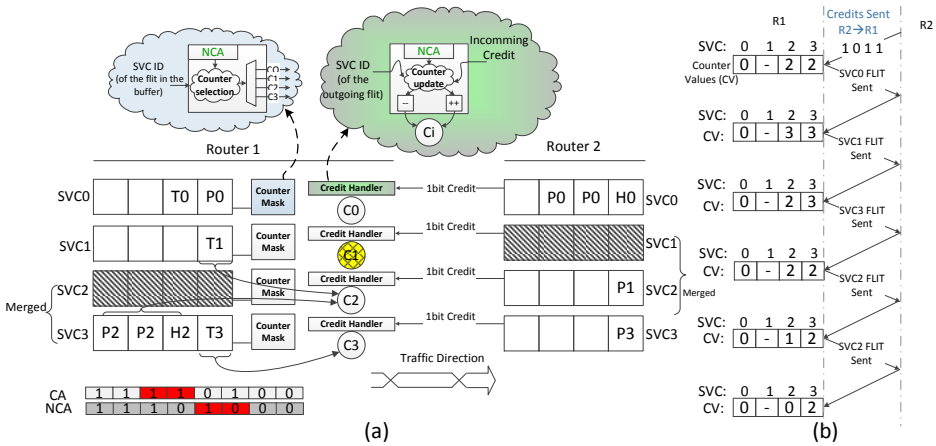


Figure 3.7: Credit handling example in a RQNoC router with Service Merge.

flit SVC-id, i.e. SVC2, and NCA, counter-2 is checked, $H2$ is sent and the counter is decremented. Similarly, the next flit that belongs to service 2 are sent next and counter-2 is decremented to zero.

3.3.2.2 Service renaming

An RQNoC with SMerge allows different traffic classes to share router resources. This may not be acceptable for some high priority traffic classes such as SVC0 (Signaling) and SVC1 (Real-time data). In order to overcome this problem the proposed RQNoC router architecture allows the router data-paths to be dynamically assigned to a traffic class via Service renaming. In practice, this is supported by the credit handling mechanism, which reassigns the credit counters, and by the service selector at the input ports.

3.3.2.3 Service Merge policy

A faulty service data-path is merged with the data-path of the next lowest priority. This ensures that higher priority services are not affected. At the output port arbitration of merged traffic is performed considering the original packet priorities. The only exception to this policy is when the lowest (SVC3) or the highest (SVC0) priority traffic is faulty. In the first case, SVC3 is merged with the next available higher priority service data-path. Even so, arbitration in the output port is performed considering the original packet priorities. In the latter case, when faulty the highest priority SVC0 should not be merged for QoS reasons. It is then renamed to an available router data-path and lower priority SVCs are merged together. For an RQNoC that requires its highest priority traffic class to be isolated, each router should have at least 2 out of its 4 data-paths available or otherwise an SDetour should be taken as explained below.

3.3.3 Combining SDetour and SMerge

An RQNoC may use either SDetour or SMerge to mitigate faults or combine them both to maximize fault tolerance. In the latter case, a fault is first attempted to be tolerated with SMerge and if this is not possible then SDetour is employed. It would be inefficient to apply the two techniques in the opposite order. Applying SDetour first to bypass a faulty service on a router would prevent the local port from sending traffic of that service and therefore reduce network connectivity from the first fault. On the contrary, applying first SMerge allows the packets of a faulty service to be injected through another SVC data-path. All three alternative RQNoC designs provide fault-tolerant links as described next and protect common router control logic with TMR.

3.3.3.1 Configuration arrays

At each port of the router an 8-bit configuration-array (CA) is stored. Service merging decisions are made off-line and the information about the current status of each SVC is stored in the CA at each router. Figure 3.8 illustrates the default format of the CA. Each 2-bits of CA are dedicated to one SVC ,i.e. bits 7-6 for SVC3, 5-4 SVC2, 3-2 SVC1, 1-0 SVC0. The value of each 2-bits indicates the SVC data-path that host the corresponding traffic class. For instance, a CA configuration of 11101000 indicates that SVC0 and SVC3 use their own resources, SVC1 and SVC2 use the SVC2 resources. Each router is also aware of the CA of its neighboring routers in order to send correctly packets to faulty routers. We call this *Neighbors Configuration-Array* (NCA). The NCA is a copy of the CA of the immediate neighboring router.

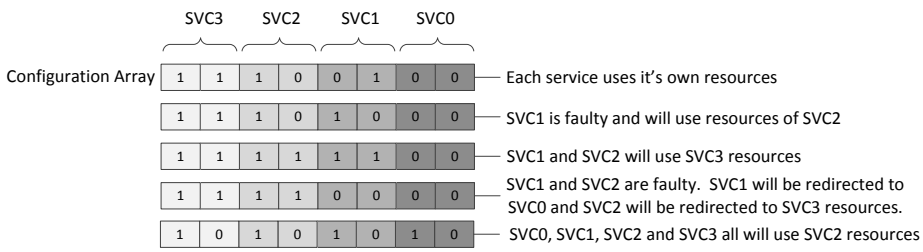


Figure 3.8: Configuration Array.

3.3.4 Resilient Links

A single faulty wire may render an entire link unusable. In order to tolerate faulty wires at a link and use with a degraded bandwidth, Vitkovskiy et al. [42] suggested shifting and retransmission of flits traversing a partially faulty link. Thereby, the flit data will be able to be sent across the fault-free wires after multiple retransmissions. The number of retransmission required depends on the number and location of faulty wires. A flit needs

to be shifted by one bit and retransmitted to tolerate one faulty wire and in the best case up to 50% of the wires being faulty; this reduces the link bandwidth to half. Consecutive fault wires have a more severe impact on performance. In general, N consecutive faulty wires require N shifting and retransmissions of the same flit substantially affecting the bandwidth of the link. In order to reduce the performance overhead of the shifting method we added two spare wires to each link. In effect, this allows to tolerate two faulty wires before degrading link bandwidth at all.

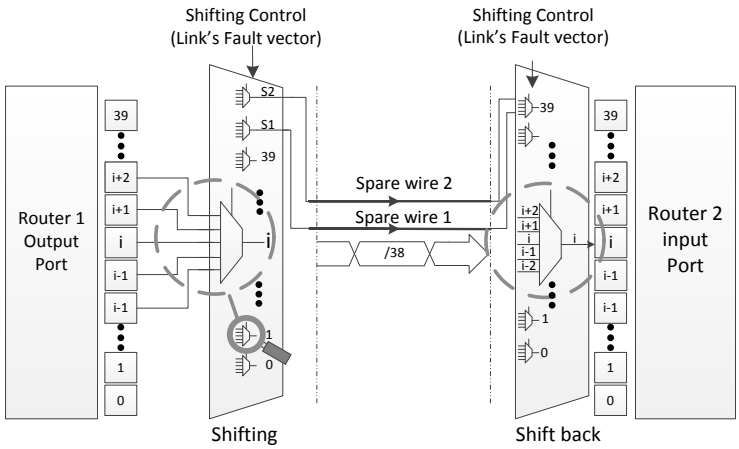


Figure 3.9: Two spare wires and a shifting mechanism can tolerate up to 2 faults without compromising link performance. More faults are tolerated by shifting and retransmission at reduced link bandwidth.

Figure 3.9, shows the design of the links used in our approach providing spare wires and shifting. At every output port of a router, the 40 bits of a link provide input to 42 5-to-1 multiplexers in order to support shifting. In particular, the i^{th} multiplexer receives input from bits $i-2, i-1, i, i+1, i+2$ and the multiplexers corresponding to the spare wires have the same inputs as the 39^{th} and 40^{th} multiplexer. At the other end of the link 5-to-1 multiplexers are used to shift back transmitted data to their original form. When retransmission, the output buffer is controlled to keep the transmitted flits and the input buffer is controlled to overwrite the respective bits of the flit. We do not allow more than one retransmission, as this would need additional hardware changes in order to shift the previously shifted values, requiring the insertion of registers at the two ends of the links.

3.3.5 Fault Model, Diagnosis and Reconfiguration

From the above techniques it is evident that our fault model distinguishes faulty resources – due to manufacturing defects or aging faults – in the following granularity: single router data-path (dedicated to one service), router control, and individual link wires. Such faults are permanent, manifested first as (repeated) transient faults which subsequently trigger

an online diagnosis test. Consequently, detection mechanisms are required to determine whether each one of the above network parts are faulty.

RQNoC packets are protected with Error-Detection Code (EDC) which are checked at each input and output port of a router, similar to Grecu et al. [88] and Kohler et al. [95]. This allows us to locate the particular link or router data-path where the fault occurred. Each EDC checker in addition provides a simple mechanism to keep track of multiple faults using a simple Finite State Machine (FSM), similar to Kohler et al. [95]. When multiple repeated faults are detected on a link or a router data-path, then a test packet is generated by the NIs of the respective and a neighboring router(s)[3]. The neighboring routers are informed to send test packets via dedicated wires. A router in test mode will have a fixed control directing the test packets to the correct output port as it cannot rely on the test packet. Test packets for links and router data-paths are constructed considering the test vectors of Vitkovskiy et al. [42] in order to diagnose particular faulty wires on a link. The same test pattern is also used for testing the data-paths. During testing, the same test packets are replicated at the demultiplexer of the input port to all four service data-paths; thereby the control logic which is repeated per service (RC, CRT, FA) can be tested. Finally, faults at the common parts of the router control are detected via TMR and an FSM is used to detect repeated faults. Then, a consistently faulty copy of the control is disabled; at a second permanent control fault the entire router is considered faulty.

All three techniques presented above require a way to communicate the test results and modify the configuration of the network in order to tolerate the faults. Service Detour requires the detour table at the NI to be updated according to the status of network resources. Service Merge needs the configuration arrays of the faulty router and its neighbors to be updated. Also the link configuration needs to be updated in order to bypass the faulty wires. This is performed via control packets generated and broadcasted from the NI receiving the test packets. In cases where this is not possible (the faulty router cannot communicate the results of a test), a time out at the nodes that initiated the tests will declare the router under test faulty, inform the rest of the network and update their configuration accordingly. In general, broadcasting of detected faults is performed to all connected network nodes using a flooding algorithm supported in the network interfaces [96]. The outcome of the network tests are forwarded by each node to its immediate neighbors. Considering that a single fault occurs at a time, then, all connected nodes get informed. Subsequently, the detour tables, merging and link configurations are updated by software running locally at each receiving node based on the new detected faults.

3.4 Evaluation

In this section, we evaluate three variations of the proposed RQNoC. We measure networks performance and fault tolerance, in terms of network connectivity preserved in

[3] A faulty link requires a test packet from the neighboring NI and a faulty router data-path from the NIs of all neighbors as well as the local NI towards the output port that reported multiple detected faults.

the presence of faults, all contributing to the QoS provided by the networks. Moreover, we retrieve post-synthesis results analyzing the overheads of our techniques in the operating frequency, power consumption and area of the QNoC. The RQNoC designs are compared to the baseline QNoC and a QNoC with global detour.

3.4.1 Implementation Results

We have implemented in RTL three 4x4 RQNoC 2D mesh designs that provide (i) Service Detour (RQNoC SDetour), (ii) Service Merge (RQNoC SMerge), or (iii) both (RQNoC SMerge+SDetour). We further implemented a baseline 4x4 QNoC 2D mesh (QNoC baseline) and a QNoC providing global detour[4] (QNoC GDetour). All designs, except the baseline, tolerate permanent faults at the links and router-control as described in Section 3.3.

The designs are implemented using ST 65nm low-power library in Synopsis Design compiler. Link delay, area and power values were measured separately considering 1 mm link length. Place & Route was used for estimating link delay and Orion [97] for link area and power. Networks power consumption was evaluated annotating the design netlist with the switching activity retrieved from simulating a set of scenarios. These scenarios consider fault-free networks with uniform random traffic at various injection rates.

Table 3.2 summarizes the overheads in operating frequency, power consumption and silicon resources of the three alternative RQNoCs versus the baseline QNoC. The QNoC GDetour has the same area and frequency with the RQNoC SDetour and is therefore omitted from the table. Supporting Service Detour increases the area by 9%; $\frac{2}{5}$ of it is due to the fault-tolerant links, $\frac{2}{5}$ due to the triplication of the common router control parts, and the remaining $\frac{1}{5}$ due to the changes in flit handling for the additional detour header. SDetour power consumption is increased by 7.3% and its operating frequency is equal to the baseline. SMerge introduces a higher area overhead of 22.4% primarily due to the changes in the control needed for merging services and to a smaller extend due to a few data-path additions for tagging packets of different services. SMerge power consumption increases compared to the baseline by 26.8% and its clock rate decreases by 9.1% due to the complexity of the router control. Finally, an RQNoC integrating both SDetour and SMerge requires 24% additional resources mainly due to SMerge and consumes 31.7% more power, having 9.1% slower clock.

Table 3.2: Implementation results of the QNoC and RQNoC designs in a 4x4 2D-mesh.

Design	**Power (Watt)**	**Area (μm^2)**	**Frequency (MHz)**
Baseline QNoC	1.64	2501k	1100
RQNoC SMerge	2.08 (+26.8%)	3060k (+22.4%)	1000 (-9.1%)
RQNoC SDetour (also QNoC GDetour)	1.76(+7.3%)	2749k (+9%)	1100
RQNoC SMerge + SDetour	2.16 (+31.7%)	3108k (+24%)	1000 (-9.1%)

[4] All services are redirected the same way.

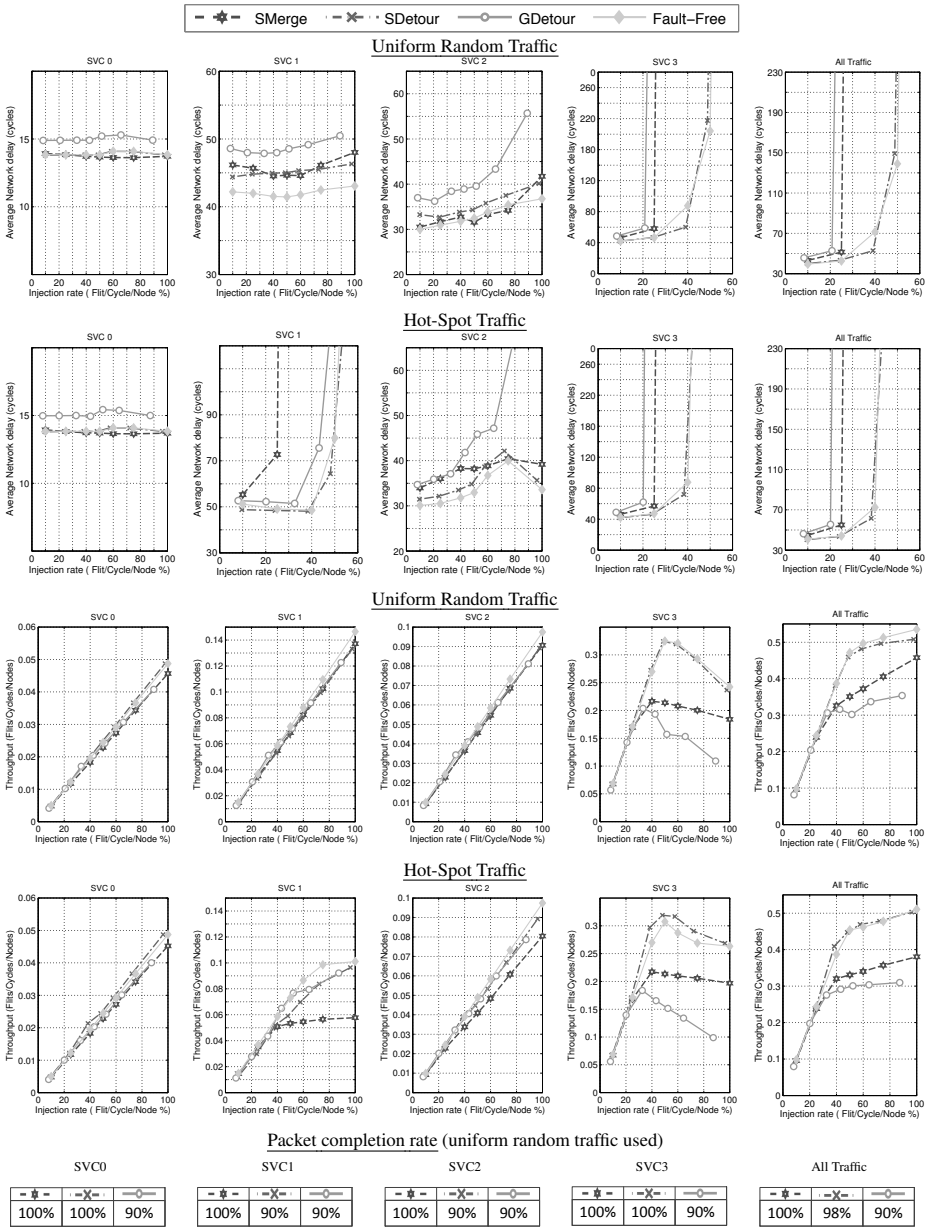


Figure 3.10: Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 1 permanent fault.

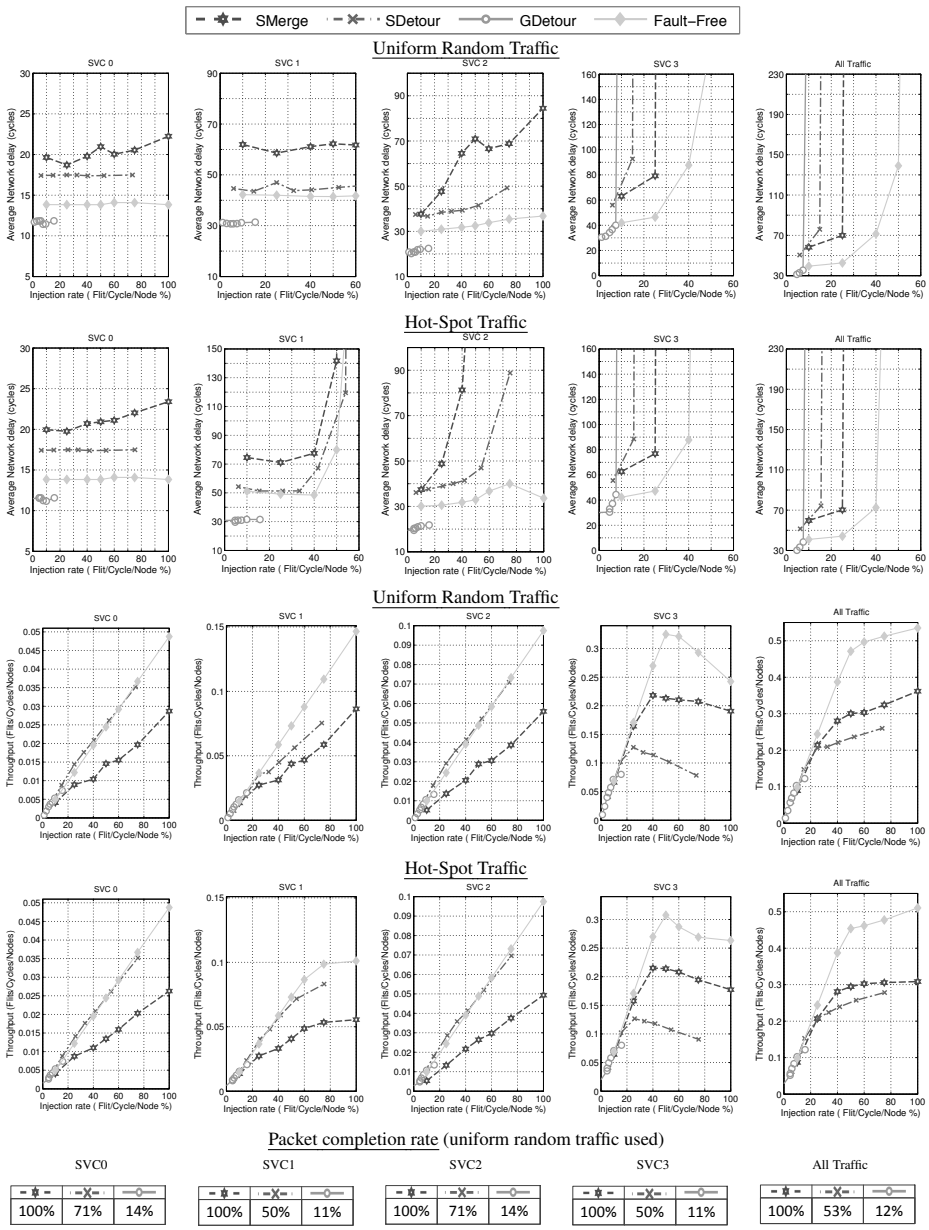


Figure 3.11: Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 8 permanent fault.

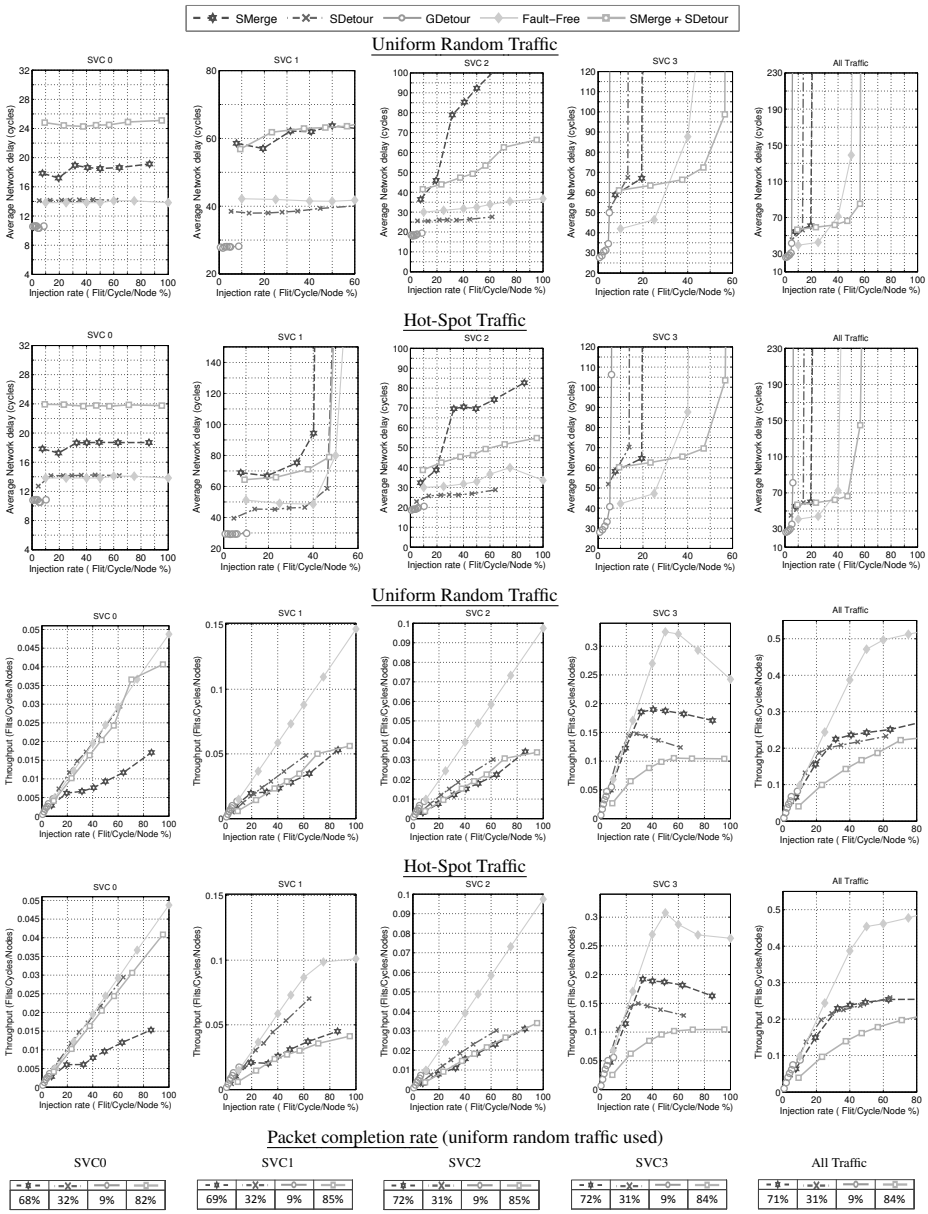


Figure 3.12: Network latency, throughput, and percentage of successfully transmitted packets (per service and total), for the RQNoC and QNoC designs. Each network has 32 permanent fault.

3.4.2 Performance Results

We evaluate the above 4x4 RQNoCs and QNoCs in terms of performance, measuring latency (in cycles), throughput (flits per cycle), and also the percentage of successfully received packets. The above indicate the preserved QoS of RQNoC in the presence of faults compared to the fault-free case. We consider that SVC0 traffic requires to always have separate router resources in order to avoid protocol-level deadlocks [98] and therefore apply service renaming to achieve this when SMerge is used. This, as a consequence, demands that each router has at least two undamaged data-paths before it fails.

Our experiments were performed simulating the RTL of the network designs for 1.5 million cycles each, having a warmup phase of 20 thousand cycles. Both synthetic uniform random and hotspot traffic were injected with up to one flit/cycle/node injection rate following the parameters described in Table 3.1. In the hot spot traffic experiments, 60% of the “real-time data" , i.e. SVC1 packets, have the same destination, while the rest of the SVC1 traffic as well as the traffic of the other SVCs remain uniform random.

We perform experiments injecting to the network 0, 1, 8 and 32 faults, the location of which is randomly selected considering the silicon area of each network part. For each of the above fault densities, multiple runs have been performed[5] and the average results are plotted in Figures 3.10 to 3.12. It is worth noting that, all networks have the same performance when fault free, which is also depicted in the figures for comparison.

Using uniform random traffic, we first measure the percentage of the packets successfully delivered to provide a better insight of the actual network load and a first estimate of fault tolerance[6]. The RQNoC SMerge+SDetour improves fault tolerance compared to RQNoC SMerge only in the case of 32 faults and therefore its performance is depicted only in Figure 3.12. In these experiments, RQNoC SMerge delivers successfully 100% of the injected packets for up to 8 network faults and about 70% for a network with 32 faults as some links or routers are entirely damaged. RQNoC SMerge+SDetour is able to mitigate some of the above faults increasing the successfully delivered packets to 84% as shown in in Figure 3.12. RQNoC SDetour follows in fault tolerance delivering 98% to 31% of the injected packets for 1 to 32 faults. Finally, GDetour treats a partially faulty router as entirely damaged preventing traffic that would otherwise use it from being sent, even if it belongs to a fault-free SVC. As a consequence, QNoC GDetour accommodates 90% down to only 9% of the packets sent.

The performance of each network is only fair to be measured considering an injection

[5]We have performed 4 runs for each particular number of faults, injecting faults to different (randomly selected) network locations and present their average results. We recognize that a particular technique may have different performance when injecting a specific set of faults, however exhaustive fault injection and network simulation would be practically impossible. Even so, our experiments show a clear trend in the performance and fault tolerance of the described techniques, allowing us to derive useful conclusions about their efficiency.

[6]Fault tolerance is more accurately evaluated in the next Section 3.4.3 where half a million fault injections have been performed to measure the connectivity of the different networks. Even so, the percentages of successfully received packets presented in this subsection are quite close to the analysis of Section 3.4.3 and in most cases within a 10% margin from the estimated mean network connectivity.

rate after excluding the packets dropped at the network interface[7]. In so doing, we consider the actual load of the networks. The injection rate in Figures 3.10 to 3.12 refers to the total traffic load of a network, a fraction of which belongs to each service as indicated in Table 3.1.

In general, under uniform random traffic packet latency remains more stable as the injection rate increases for higher priority services and saturates only for the lowest priority service (SVC3). A similar observation holds for throughput which scales (almost) linearly with the injection rate for higher priority services. As expected, with hot spot traffic, SVC1 saturates in high injection rates and affects the performance of other (mostly lower priority) services.

RQNoC SDetour incurs low performance overheads in low fault densities and high priority packets. However, as the number of faults increases and routing paths get longer, SDetour becomes inefficient and this is mostly evident in low priority services. More precisely, in networks with 1 fault and in SVC0, SDetour has similar performance to the fault free case, for SVC1 and SVC2 average packet latency increases 6%-11%, while SVC3 saturates at the same point as the fault-free network. Throughput is almost similar to the fault free case with about 6% decrease or even better (in SVC1) than fault free as shown in Figure 3.11 as detouring packets contributes to load balancing and avoiding congestion. The same pattern can be seen for the hot spot traffic where SDetour in SVC0 has almost the same latency with the fault free, while SVC1, which is the stressed traffic, saturates at the same point as the fault free case. The latency for SCV2 increases by up to 5%. At higher number of faults, SDetour starts having a significant performance decrease. For 8 and 32 faults, SDetour maintains about 50% of the overall fault-free throughput and saturates at 10-15% injection rate versus 50% of the fault free case. In all fault densities, SDetour performs well for SVC 0 and 1 showing that longer paths may have a smaller impact in the performance of high priority services under uniform traffic. However, in high injection points of hot spot traffic, the two low priority traffic classes are effected significantly. More precisely, in network with 8 faults, SVC2 and SVC3 saturate at 75% and 25%, respectively. In the network with 32 faults, due to high number of dropped packets, the latency slightly decreases.

SMerge incurs a significant latency overhead in most cases. Allowing different traffic classes to share, besides the network links, the same router data-paths increases the overall latency, even if priority arbitration is maintained in the output ports. An additional reason for the lower SMerge performance is that flits of different packets cannot be interleaved in data-paths shared by multiple services as they would if they had their own data-path. Packet latency increases more for lower priority traffic and higher fault densities. In particular, for one network fault latency increases by 1%, 11%, and 30% for SVC 0, 1 and 2, respectively, while SVC3 saturates at 25% of the injection rate versus 50% in a fault free network. Throughput drops notably only in SVC3 to about 70-80% of the fault free. Similarly, for the hot spot traffic, latencies of SVC0 and SVC2 increase by 1% and 15%, respectively while SVC1 and SVC3 saturate at 25% injection rate. At 8 and

[7]These are packets that require to pass through faulty network resources which cannot be repaired or avoided. As a consequence, these packets are dropped before injected in the network.

32 network faults, latency increases by 1.5-2.3×, 1.3-3.8×, respectively. In these fault densities, the saturation point is at 20% of the injection rate versus 50% in the fault free case and throughput is about 50%-70% of a fault-free network. It is worth noting that for 32 faults, SVC2 also saturates due to high number of merging with SVC3. For the hot spot traffic, and up to 8 network faults, SVC1, 2 and 3 all saturate as a consequence of merging policies and highly stressed SVC1 traffic. For the network with 32 faults, the latency of SVC2 improves due to reduced number of packets in the network.

An RQNoC with SMerge and SDetour can improve fault tolerance in networks with 32 faults delivering 84% of the generated packets. The combination of sharing resources among services (SMerge) and using longer paths (SDetour)increases the packet latency for high priority SVCs, as shown in Figure 3.12. However, for the larger, low priority traffic-load, using different paths (SDetour) implicitly avoids congestion points and reduces latency. On the contrary, throughput is better than SMerge for high priority traffic and worse for lower priority. This is explained by the fact that sharing NoC resources favors higher priority packets, which get a larger share of the links and router data-paths. Overall, latency is 1.5-2× higher than the fault free QNoC (or RQNoC) and throughput is similar to fault free for SVC0 and drops to about 50% for the other services.

In most cases, GDetour substantially increases packet latency although throughput is close to the fault-free case. To exemplify, GDetour latency increases by 7%, 15% and 50% for SVC 0,1,and 2, respective, even with a single network fault and reaches saturation point at 10% injection rate with 8 faults. In high fault densities we cannot derive useful conclusions about GDetour as the percentage of successfully transmitted packets is below 20%.

In summary, there is a clear performance-reliability tradeoff between the two proposed RQNoC mechanisms. Using alternative resources of the same service (SDetour) to mitigate faults achieves better performance but lower fault tolerance compared to using resources of another service (SMerge). In all cases, tolerating permanent faults using RQNoC incurs (in some cases significant) overheads, but it offers better network connectivity. Allowing a minimum number of network nodes to remain connected can be more critical than performance for avoiding system failure. Besides, many systems are designed to offer graceful degradation of performance and/or functionality in the presence of permanent faults [99]. Despite the RQNoC performance overheads only the lowest priority service saturates as it does so in the fault-free case, too. This shows that traffic classes still enjoy QoS even when parts of the NoC are damaged.

3.4.3 Network Connectivity and Fault Tolerance

We present next a more accurate evaluation of fault tolerance and network connectivity; which directly relates to NoC and SoC QoS as system failure may depend to a minimum number of connected nodes, besides minimum performance. This is achieved creating network models for each RQNoC design and performing multiple fault-injections for different fault densities. In each case, our models return the network connectivity defined as the percentage of available paths in the network out of the total number of source

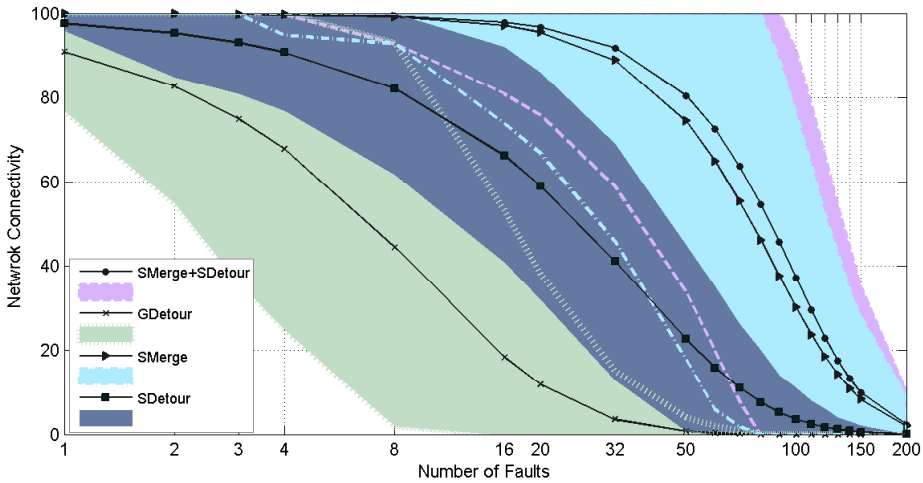
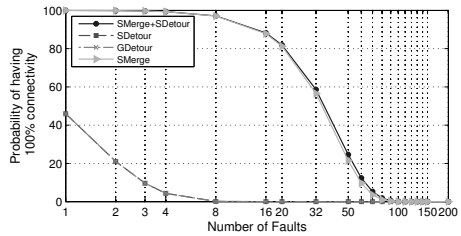


Figure 3.13: Mean and $\pm 3\sigma$ range of connectivity values for the proposed RQNoC designs at different fault densities.

to destination pairs. In this part the following 4x4 2D mesh network are evaluated: SMerge+SDetour, SMerge, SDetour, and GDetour. We considered fault densities ranging from 1 to 200 network faults. For each fault density, half a million different fault injections have been performed. Faults are injected randomly considering the probability of failure for different network parts based on their area.

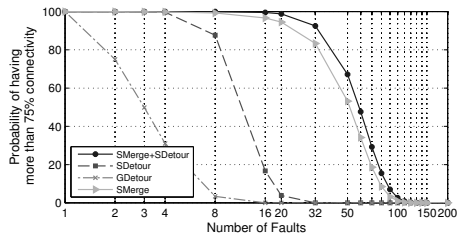
Figure 3.13 depicts, for different fault densities (number of faults), the mean network connectivity as well as the range of values that are up to 3 standard deviations ($\pm 3\sigma$) away from the mean. SMerge+SDetour provides the highest network connectivity, which is above 99% for up to 8 network faults, 92% for 32 faults, 80% for 50 faults and 37% 100 faults. SMerge alone can support a similar connectivity preserving 99.3%, 89%, 75% and 30% of the network paths for 8, 32, 50 and 100 faults, respectively. SDetour follows with 82%, 41%, 20% and 3.6% connectivity for the same fault densities. Finally, GDetour shows poor fault tolerance maintaining only 44% of the network connectivity at 8 network faults, 3.5% for 32 faults and below 1% for more than 50 faults.

Figure 3.14 illustrates the probability of a network to offer 100%, 75%, 50% and 25% connectivity at a particular fault density. This is important for networks that are expected to guarantee a particular quality of service. SMerge and SMerge+SDetour offer similar fault tolerance having above 80% probability to offer a fully connected network even with 20 permanent faults. The probability of full connectivity is significantly reduced for SDetour and GDetour as the turn model used for the routing cannot always connect all node pairs. However, as the connectivity requirements reduce to 75%, 50% and 25%, SDetour performs better than GDetour. Even so, SMerge and SMerge+SDetour can deliver at fault densities 2-3$\times$ higher the same probability as SDetour for a particular network connectivity.



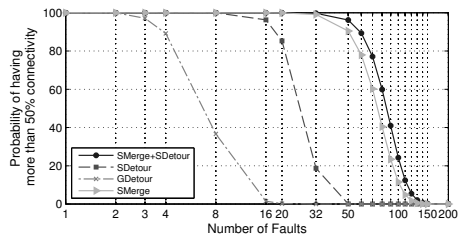
(a)

Probability of having full network connectivity under different fault densities.



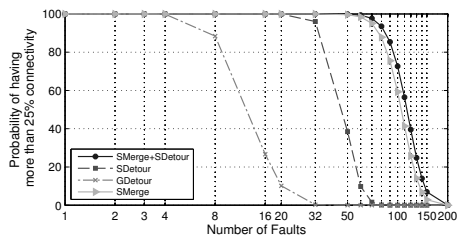
(b)

Probability of having at least 75% network connectivity under different fault densities.



(c)

Probability of having at least 50% network connectivity under different fault densities.



(d)

Probability of having at least 25% network connectivity under different fault densities.

Figure 3.14: Probability of different RQNoC networks to deliver a particular network connectivity under different fault densities.

3.4.4 Comparison

Providing a complete and detailed comparison of RQNoC with related work is extremely difficult as related techniques are applied to different networks with different network sizes, baseline router architectures, topologies, or even using different technologies. In addition, various different metrics are used to evaluate NoC fault tolerance and its overheads. Even so, we attempt next to compare with related techniques to the degree that is possible, retrieving RQNoC results that match other networks parameters and metrics.

Compared to FoN [100], Cost-based [95] and FTDR-H [33], RQNoC has lower overheads and maintains a larger fraction of its performance in the presence of faults. In particular, FoN, Cost-based and FTDR-H are bufferless networks, as opposed to RQNoC that uses 60 bytes of buffering per port, and are all implemented in 65nm like RQNoC. Compared to its baseline FoN has an area overhead of 1.35× due to a more complex routing controller. For the Cost-based and FTDR-H the area cost increases by 2.85× and 2.58×, respectively, due to the use of large routing tables.On the other hand, RQNoC[8] requires 1.24× more resources than its baseline. The power overheads of FoN, Cost-based and FTDR-H, measured in mWatts/MHz, are 4×, 17×, and 9× that of their baseline, respectively, while RQNoC increases its power consumption by only 1.32×. Moreover, FoN, Cost-based and FTDR-H maintain 50-25% of their throughput; at the same fault rates RQNoC maintains 81% to 66% of its throughput. It is worth noting that performance in FoN, Cost-based and FTDR-H is measured in 8x8 meshes operating at 500MHz.

ReliNoC [47] is implemented also in 65nm and uses similar to iMesh two parallel channels. ReliNoC increases its area by 15%, when using 4 buffers per port of 4 bytes each (16 bytes per port), due to changes in the input ports of the routers, versus 24% area increase in RQNoC. However, the percent of fully connected ReliNoC networks at 20 to 100 faults ranges from 90% to 30%. For the same network size (8x8 2D mesh) RQNoC is able to deliver 99%-70% at the same number of faults delivering substantially higher network connectivity and fault tolerance.

Bulletproof [1] is applied to network routers with 8 buffers of 4 bytes per port (32 bytes per port) and targets also 65nm technology. It has high area overhead of 3.42× compared to its baseline due to additional spare resources, TMR and Error Correcting Codes (ECC) used; that is 10× higher overhead than the RQNoC. Reliability is measured in Bulletproof as the mean number of faults that cause a router failure. Bulletproof can sustain up to 38 faults before a router fails, while an RQNoC router can tolerate about 6. We can observe a clear trade-off between area and reliability in the two approaches, which is based on the granularity of the considered fault model. An RQNoC router is divided in 7 distinct parts, as opposed to over 200 partitions in the Bulletproof router, having significantly lower area overhead but tolerating fewer faults. Fault densities of tens of permanent faults in a silicon area that a single NoC router occupies is quite unlikely even in future emerging technologies and therefore we consider that a coarser fault model

[8] In all comparisons, we consider the RQNoC with both SMerge and SDetour.

granularity is preferable.

Vicis [44, 101] uses port swapping, ECC and router bypass requiring 51% more area than its baseline at 42nm technology (versus 58% for RQNoC at 65nm). Vicis is able to keep about 90% and 80% of its routers connected at fault rates of 2.5 and 5 faults per router, respectively. At the same fault rates RQNoC keeps 85% and 55% of its routers connected. Although Vicis appears to be more reliable than RQNoC, it uses a torus topology which is inherently more fault-tolerant than the RQNoC 2D-mesh topology.

Row-Column decoupled router (RoCo) implemented in a 8×8 2D-mesh network offers only 85% packet completion when tolerating 4 faults [45]. RQNoC is significantly more reliable as it achieves above 99% packet completion even with 16 faults in a 4 times smaller network (4×4 2D-mesh). Area, performance and power overheads are measured using 90nm technology and are combined into a single metric rather than presented separately and therefore we cannot compare them. RoCo evaluation results are for 4-port routers with three virtual channels per port and 5-flit deep buffers per VC (240 bytes per port).

Finally, Virtual channel renaming uses 4-stage pipelined switches with four, 8-deep physical virtual channels (128 bytes per port). The design has very low area and power overheads using 65nm TSMC (5.3% and 3.7%, respectively) and low performance costs at low fault rates (similar to RQNoC) [46]. However, it requires each physical buffer to fit two flits per virtual channel; for a router that supports 4 VCs with 4 physical buffers that means 8 flits per buffer, which is 4× larger than the minimum buffer size of a regular VC-based network. Such area (and power) costs are possibly not reflected in the above overheads. Moreover, VC renaming protects only the buffers of a router, while packet priorities are only fixed to each physical buffer. On the contrary, RQNoC does not have these limitations.

3.5 Conclusion

This part of the thesis described RQNoC, a solution for tolerating permanent faults on a Network-on-Chip that supports multiple services. Service Detour and Service Merge were proposed to redirect traffic and bypass the faulty resources of a particular service. On the one hand, Service Detour preserves the isolation between service lanes and uses alternative, longer paths of the same service to work around faulty resources. On the other hand, Service Merge allows sharing of a router data-path between multiple traffic classes, trading service-isolation for shorter paths and higher fault tolerance. The two approaches were evaluated in terms of implementation costs, network performance per traffic class and fault tolerance. SDetour has lower cycle time, area and power costs compared to SMerge as it requires fewer hardware modifications. It also has lower network performance overheads in low fault rates, but becomes inefficient as the number of faults increases, substantially reducing network reliability. SMerge requires roughly 22% more resources than our baseline QNoC network and has about a quarter of higher power consumption. SMerge network latency may double and its throughput can be reduced to 50% compared to a fault-free case, but it is able to tolerate substantially

higher number of faults. SMerge preserves 90% of the network connectivity even with 32 network faults, which is more than double compared to SDetour. Combining SMerge with SDetour further improves fault tolerance increasing connectivity by up to 5%.

4

Adaptive Fault-tolerant Main Memories

Main memory reliability is a major challenge as the exponential growth of the total DRAM sizes leads to higher failure-rates [102] despite the fact that the per Mbit failure-rate is not increasing [103, 104]. In current systems more than 40% of the reported hardware faults are related to main memory and this number is expected to increase rapidly in future exascale systems with tens of Petabytes of DRAM [102, 105]. Furthermore, more recent DRAM cell technologies have higher failure-rates compared to previous generations, as shown by a recent in-field study of Facebook servers [106]. On the other hand, in the future replacing faulty devices will become more difficult and costly, if not impossible. For example 3D-stacked memories may be permanently attached to the processor directly or through a silicon interposer [71].

Error correcting codes (ECCs) are widely used to improve DRAM reliability. Different Hamming codes (SEC-DED, DEC-TED) or more advanced block codes (Reed-Solomon, BCH) provide different detection and correction capabilities at different capacity overheads, energy and performance costs. For instance, the widely used SEC-DED (Single-Error-Correct-Double-Error-Detect) protects[1] each 64-bit data with 8 additional bits of code, often stored in a ninth DRAM device, introducing 12.5% capacity overhead. Although SEC-DED can cover single-bit memory faults, larger granularity multi-bit DRAM failures are becoming equally frequent [104, 107]. Multi-bit faults could be the result of faults in shared internal circuitry of DRAM [103], or disturbance faults that affect an entire DRAM row, a phenomenon known as *row-hammer* [108]. More advanced schemes, are employed to provide Chipkill[2]-level protection and tolerate

[1] Throughout this section we interchangeably use the terms *fault tolerance* and *protection*

[2] Chipkill is a trademark used by IBM for an ECC scheme that protects up to an entire DRAM chip failure.

multi-bit faults or even entire DRAM chip (device) failures at the cost of wider memory accesses [65]. Compared to SEC-DED, Chipkill reduces the uncorrected DRAM error rate 42 times [103], but requires $\times 4$ devices (4-bit wide) to increase the protection level. That increases energy costs by 30% compared to $\times 8$ devices [109]. Supporting Chipkill in a single ECC-DIMM with $\times 8$ devices requires 26.5% capacity overhead [66].

Spending up to a fifth of DRAM capacity for ECC is expensive and for many systems impractical. Total DRAM sizes may continue to increase, but the DRAM size per core is expected to reduce in future large scale systems [110, 111]. In addition, while it is relatively easy to add an extra device in a DRAM DIMM, the same approach is not feasible for 3D-stacked DRAMs. Adding an extra die to handle the protection imposes increased fabrication cost in addition to exacerbating thermal impacts for the lower dies [112].

Many industrial and academic approaches reduce the capacity overheads while maintaining Chipkill-level protection. They require specific memory system configuration [65], wider memory accesses [61], employ more complex ECCs [64, 67, 113], or otherwise rely on data compression to save space for the ECCs [69, 70].

Although effective, all these schemes apply the same flat protection level to the entire DRAM. Nevertheless, recent studies have shown that in many cases, not all data are equally critical, *i.e.* an application may have different sensitivity to faults happening in different parts of its data [49, 50, 59]. Thus, there is an opportunity to exploit memory protection schemes that allow for variable protection levels depending on the criticality of the data. So far very little has been done to exploit this observation. Luo et al. [49] suggested using heterogeneous DRAM DIMMs with different fault tolerance, but this offers a fixed partitioning of the memory. Ideally, a finer granularity of selecting the protection level of data stored in DRAM would minimize the storage, performance and energy costs of ECCs. VECC [60] allows selected data to be coupled with ECCs which are stored separately and therefore require an additional memory access that penalizes performance. Finally, Flikker allows lower refresh rates to pages with less critical data trading fault-rate for refresh energy [59].

In this chapter, we propose Odd-ECC, a new flexible memory mapping scheme that allows systems to select dynamically on-demand different protection levels (ECC) for different allocated pages. It requires minimal changes to the memory controller, which handles the different mappings corresponding to different levels of protection. The ECC overheads are hidden in pages marked as unavailable to the user by the Operating System (OS). Physically, however, ECCs are aligned with the data they protect so to reduce access latency. The OS further manages page allocation maintaining different pools of pages for different protection levels. Our approach is applied to both traditional 2D and 3D-stacked DRAMs and offers for each supported protection level the same access latency as they would have in a flat protection equivalent. Thereby, ECC storage (as well as performance and energy) costs are proportional to the actual protection level required for particular data, without compromising the reliability of the application at hand.

In this thesis we use the term chipkill-level protection to refer to any technique that provides such level of protection.

We analyze various applications measuring their sensitivity (Mean-Time to Failure) to faults injected in different regions of their data. In doing so we identify data regions that impact significantly or very little the reliability of an application. The former ones are then candidates for higher protection and the latter ones for lower protection. The proposed Odd-ECC is then employed to dynamically provide this mixed memory fault tolerance on demand.

The main contributions of this work are the following:

- We extensively study a variety of applications analyzing the impact of faults on their different data regions to their reliability (MTTF).
- We present Odd-ECC, a new memory mapping scheme able to support different levels of fault tolerance for data stored in memory, decided dynamically on demand for each allocated physical page.
- We show how Odd-ECC is applied to both conventional 2D DRAM DIMMs as well as to 3D-stacked DRAMs.
- We evaluate Odd-ECC using the same applications and comparing the achieved MTTF, the ECC capacity, performance and energy overheads versus flat memory protection schemes.
  - Odd-ECC can be used to reduce capacity overheads (besides slightly reducing performance and energy costs) maintaining similar reliability to a flat protection scheme.
  - Alternatively, Odd-ECC can significantly improve reliability under the same capacity constraints as a flat protection scheme at the cost of some performance and energy overheads.

The remainder of this section is organized as follows. A large variety of applications are analyzed in Section 4.1 showing that they have different sensitivity to faults injected in different regions of their data. Section 4.2 describes the proposed Odd-ECC approach for dynamically selecting the fault tolerance level of data stored in memory. Sections 4.3 and 4.4 explain how Odd-ECC is applied to conventional 2D DRAMs and 3D-stacked DRAMs, respectively. Section 4.5 presents our experimental results, evaluates the impact of Odd-ECC in performance and energy, and shows the savings in memory capacity. Section 4.6 presents related work, and Section 4.7 summarizes our conclusions.

4.1 Application Reliability Analysis

We analyze various applications and confirm the findings reported in literature that indeed an application may not be equally sensitive to faults in different regions of its data [49, 50, 59]. Consequently, each data region may require a different level of memory fault tolerance. This observation can also be extended to systems hosting multiple applications.

In this section, we describe the methodology followed to conduct the sensitivity analysis and mean time to failure (MTTF) study of applications. Subsequently, we define failure scenarios for various applications at hand and identify separate regions of data that contribute differently to application failures. These data regions are then candidates for applying different protection levels in the main memory.

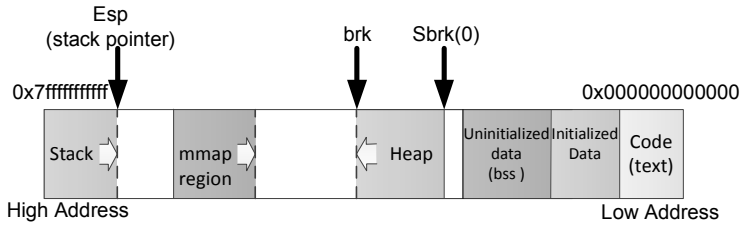


Figure 4.1: Organization of users virtual address space for a typical application in $x86\_64$ architecture.

4.1.1 Application Data Regions

Although the users can select the protection level of data at arbitrary granularities, previous works most often experiment with standard regions, such as heap, stack, and global [49, 50, 59]. Such partitioning of application data is shown in Figure 4.1 and used in the analysis of the applications considered in our evaluation.

In Odd-ECC the protection level of data in memory can be selected at page granularity, for simplicity we restrict the protection regions to the standard, *i.e.* stack, heap and global[3]. We study the effect of faults on data regions using applications from NAS Parallel Benchmarks (NPB) 8-threaded Class A [114] and AxBench approximate benchmarks [115]. In addition to the three main memory regions, a fourth region of interest in the memory is identified for applications of both benchmarks. NAS parallel benchmarks are OpenMP multi-threaded applications. In OpenMP the main thread (also known as the master thread or thread 0), is responsible for allocating a stack for every newly spawned thread (known as a child thread) on the heap [116]. Although the stack of each thread is its private memory region [117], all threads may access the stack of thread 0 [118]. Therefore, for NPB we defined the stack region as the stacks of all child threads, and consider a fourth region, which is the stack of thread 0. For the AxBench benchmarks, given their nature of being more tolerant to relaxed accuracy in part of their operations and data, we define the input data of each application as the separate fourth region of interest.

[3]Code region is excluded.

4.1.2 Application Sensitivity Analysis

The sensitivity of an application to faults injected in different data regions is carried as follows. A Pin [119] based fault injector was developed and used for evaluating the effect of single-bit and multi-bit faults. Before fault injection, the application is profiled to find the address boundaries and total number of memory loads to each data region [4]. Using the information collected in the application profiling, separate experiments are carried for each data region and fault type.

Every experiment requires multiple runs of an application, one for every injected fault. During a run, memory load requests to the targeted data regions are identified and one is randomly selected for fault injection. The *SafeCopy* function of the Pin-tool is employed to inject the fault[5] before the application proceeds further.

After fault injection, the application is resumed until completion and the output is logged and compared with the expected one. Based on this comparison, the effect of a single fault injection is classified in the following three categories: *(i)* **Crash** when the injected fault causes the application to stop before completion [6]; *(ii)* **Wrong result** is when the logged output is different from the expected output and *(iii)* **Correct result** is when the logged output value is equal to the expected value. For each data region and fault type 10,000 faults are injected, and their outcome is logged. The distribution of fault injection outcome gives us an indication of how sensitive each data region is to faults, without yet considering the probability of encountering such a fault in that region, which is subject to the fault rates and to the size of the corresponding region.

4.1.3 Application MTTF Study

After the sensitivity analysis that shows the possible impact of a fault in a particular data region of an application, we proceed with calculating the *mean time to failure* (MTTF) of an application due to faults in different regions. This requires the following steps. The time intervals during which each memory location is vulnerable to faults should be taken into account as well as the sizes of the data regions, which in turn affect the probability of a fault. Then, the failure definition needs to be described per application. That is because for some applications a wrong result may be acceptable if it is within a certain threshold, or easily detectable at the application level. The different fault rates are then calculated before the MTTF can be retrieved.

During applications execution, a single memory location can be accessed multiple times. Depending on the sequence of reads and writes to a memory location, soft faults can be manifested or masked, *i.e.* a memory write-back will eliminate any possible fault that happened after the last read. In effect, during the execution of a program there are time intervals during which a memory location is vulnerable to faults and others

[4] Address boundaries of different regions are the same for every execution as long as the address space layout randomization (ASLR) is disabled.

[5] Flipping a random bit for modeling single-bit faults and flipping two random bytes for modeling multi-bit faults.

[6] Delayed runs with more than 2× execution time are timed-out and classified as crash.

Table 4.1: Classification of applications' outcomes in fault injection experiments. In the table, ϵ is relative error and RMSE is root-mean-square error.

Possible Outcomes of Fault Injection					
	Wrong Results			Correct Result	Crash
Application	SDC	Detectable at Application Level	Acceptable Results Deviation		
Blackscholes	ϵ >10%	Execution timed-out\ Output equal to zero	ϵ <10%	Equal to the expected	App Crash\ ϵ >2x
FFT	ϵ >10%	Execution timed-out\ Output equal to zero	ϵ <10%	Output equal to the expected	App Crash\ ϵ >2x
Inversek2j	ϵ >20%	Execution timed-out\ Output equal to zero	ϵ <20%	Output equal to the expected	App Crash\ ϵ >5x
Jmeint	Miss-rate >1%	Execution timed-out	Miss-rate <1%	Output equal to the expected	App Crash
Jpeg	RMSE-diff>20%	Execution timed-out	RMSE-diff>20%	Output equal to the expected	App crash\ RMSE-diff>2x
NAS-PB	Wrong results	Execution timed-out	-	Output equal to the expected	App Crash

intervals during which it is not. This phenomenon is taken into account in our experiments following the methodology used by Luo et al. and Mehrara et al., to find the *safe-duration* and *vulnerable-duration* of data regions and incorporate them with our analysis [49, 50]. In general, a memory location is in the vulnerable-duration between a write and the following read(s) and is considered to be in safe-duration between reads/writes followed by a write. Similar to Luo et al., we define the safe-ratio for an address in the memory as the sum of safe-durations divided by the applications execution time. For this purpose, we use an in-house system simulator (detailed in Section 4.5) and sample the safe and vulnerable duration cycles[7] for a sample of about 1% to 10% of the memory locations in each data region. The measured safe-ratio for each application is then considered as a factor in the MTTF calculation.

The size of the data regions used by an application can be measured using available Linux tools, such as GDB [120] software debugger and Valgrind massif [121]. In our experiments, Valgrind massif heap profiler was used to take snapshots of an application during its run-time and calculate the average heap and stack sizes based on sampling intervals. The sizes of the data regions were subsequently incorporated in our analysis.

For different applications each of the three possible outcomes of a fault (crash, wrong result, correct result) may have a different effect. In some cases, a wrong result may be considered as correct if within some acceptable margins. In other cases, a wrong result may be easily detectable at the application level because it is not within the expected value range; then, that it can be considered a crash. All other cases of wrong results are classified as Silent Data Corruption (SDC). Table 4.1 summarizes for each application at hand the definition of the above cases. More precisely the (i) SDC, (ii) wrong results detectable at application level, (iii) wrong results within acceptable error, (iv) correct results, and (v) crashes are defined. Note that, besides an unexpected termination of a program, longer than expected execution times (timeout) may also be treated as crashes.

[7] For a given last level cache size.

Henceforth, cases (iii) and (iv) are merged as correct results, cases (ii) and (v) as crashes, and case (i) SDC constitutes the failure definition of each application. Similar to previous works, crashes are not considered as failures as they are detectable [122–124]. Indeed for some applications crashes can be acceptable if they do not happen too often and they don't prevent computations from advancing. As shown in Table 4.1 scientific HPC applications (NPB) do not tolerate any deviation from the expected program outcome. On the contrary, AxBench programs allow some relative error ϵ. For approximate benchmarks, we put a threshold of a relative error[8] between the actual (after fault injection) and the correct results and consider anything beyond that as SDC. Finally, some programs consider zero output as invalid and some others have an expected execution time beyond which they timeout.

The probability of single-bit and multi-bit faults on each data region is calculated based on the fault rates and measured data region sizes. We use the fault rates[9], *i.e.* λ, reported in [73, 104] and the first-order-probability analysis explained in [125], to find the probability of faults, *i.e.* the probability of having a single-bit fault or a multi-bit fault in a data region. Subsequently, for each fault type x and data region i, we calculate the probabilities of having a crash ($P_{x,i,Crash}$) or SDC ($P_{x,i,SDC}$). The results of sensitivity analysis are used in this calculation as the percentages of crash ($Per_{i,crash}$) or SDC ($Per_{i,SDC}$) for each region i. This is performed by using:

$$P_{x,i,Crash} = P_{x,i} \times Per_{i,crash} \times (1 - Safe\_Ratio_i) \tag{4.1}$$

$$P_{x,i,SDC} = P_{x,i} \times Per_{i,SDC} \times (1 - Safe\_Ratio_i) \tag{4.2}$$

where $P_{x,i,Crash}$ and $P_{x,i,SDC}$ are the probabilities of crash and SDC events when in region i a fault of type x happens, $P_{x,i}$ is the probability of fault type x in region i and $Safe\_Ratio_i$ is the safe-ratio of region i. Finally, from the calculated probabilities, we find the *region crash/SDC rates* using:

$$\lambda_{x,i,crash} = -\ln(1 - P_{x,i,crash}) \tag{4.3}$$

$$\lambda_{x,i,SDC} = -\ln(1 - P_{x,i,SDC}). \tag{4.4}$$

where $\lambda_{x,i,crash}$ and $\lambda_{x,i,SDC}$ are the rate of crash and SDC events, respectively, in region i, with fault type x, in one billion hours.

Using the above methodology and fault rates of 2D- and 3D-DRAMs shown in Tables 4.2 and 4.3, we can calculate the rates of observed SDC or crash in each region of the applications at hand. Note, that besides the raw fault rates, these Tables further report the effective fault rates after applying fault toleance schemes that (i) detect multibit errors and correct single bit errors (Tier-1 /T1), and (ii) in addition correct multibit errors (Tier

[8] For *jpeg* application root-mean-square error (RMSE) and for *Jmeint* miss-rate is used.

[9] Fault rate is typically reported in FIT where one FIT is equal to having one fault in 10^9 device hours.

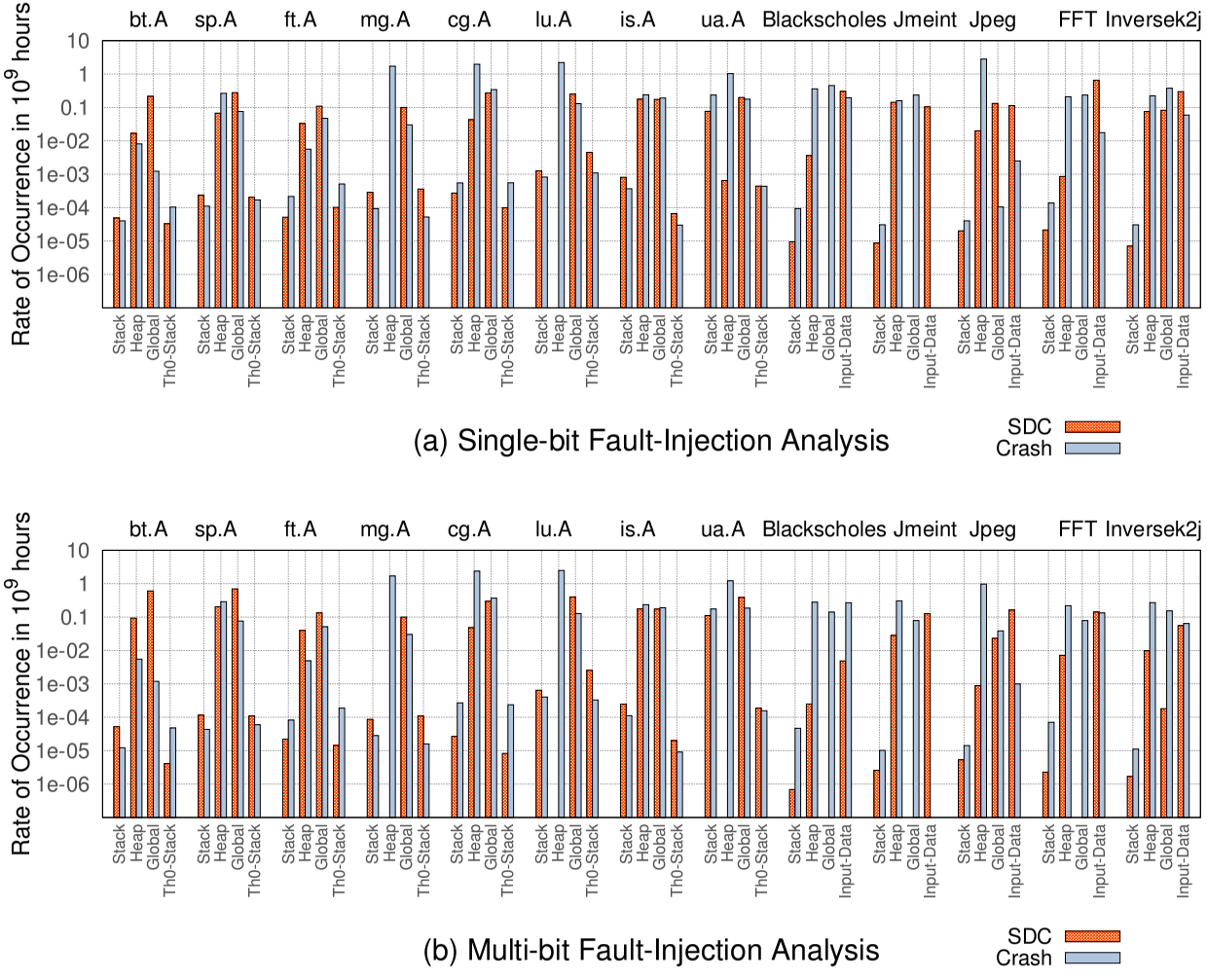


Figure 4.2: Rate of observing silent data corruption (SDC) or crash under: (a) single-bit and , (b) Multi-bit faults in each memory region considering 2D-DRAM fault rates.

2 / T2). Both these fault tolerance schemes are used in the following sections.

Figures 4.2 and 4.3 show the rates of observed SDC and crashes in each region of the applications at hand, using the fault modes of 2D-DRAM and 3D-DRAM, respectively. The results confirm the error resilience diversity among different regions of application data and, in addition, highlight the most vulnerable ones. This analysis is used in the evaluation of the proposed Odd-ECC as it guides us to select the appropriate protection level for each data region of an application, reducing the ECC overhead without significantly affecting MTTF.

So far, we have shown how to calculate the failure (SDC) rate for a single data region based on the sensitivity analysis results. Failure rates of independent entities can be added with each other. In doing so, we can calculate the failure rate of the entire application by summing up the failure rates of all its data regions. That is: $\lambda_{application}$=$\sum \lambda_i$; where λ_i is the failure rate of memory region i. Assuming constant fault rate [10], the MTTF of an application is equal to $\frac{1}{\lambda_{application}}$. This way of calculating the MTTF of an application makes it possible to explore the impact of different protection levels per data region, on the application reliability. Concisely, when a data region is protected with a particular fault tolerance mechanism, based on the fault modes and protection coverage, a new effective fault rate (*i.e.* in FIT) can be calculated. In Table 4.2 and Table 4.3 the two columns under effective FIT rate are examples of changes when different protection schemes are used. Then, the new effective fault rates are used to find the new region's failure rate, *i.e.* λ_i^{new}. Finally, new application MTTF can be calculated using failure rates of all regions as follows: $MTTF^{new} = \frac{1}{\sum \lambda_i^{new}}$, where each i corresponds to a different memory region.

Table 4.2: Failure rates for 2D DRAM. Effective FIT rates are calculated according to correction/detection capabilities of two protection schemes, i.e. T1 and T2, based on estimated coverage reported in [74, 126].

Fault Mode	Fault rate (FIT/Mbit)	Effective FIT rate(FIT/Mbit)	
		With T1-ECC	With T2-ECC
Single-Bit	0.03	0	0
Single-Column	3.8E-03	5.7E-04	0
Single-Row	5.2E-03	5.2E-03	0
Single-Bank	4.2E-03	4.2E-03	0
Multiple-Bank	4.4E-04	4.4E-04	0
Multiple-Rank	4.8E-04	4.8E-04	4.8E-04

4.1.4 Application Analysis Results

Figures 4.2 and 4.3 show how the resiliency of an application varies with faults in different data regions. The results for 2D- and 3D-DRAM are very similar considering the small differences in fault rates (summarized in Table 4.2 and Table 4.3 respectively). Each pair

[10] Sridharan *et al.* [104] have shown that DRAMs have approximately constant transient fault rate.

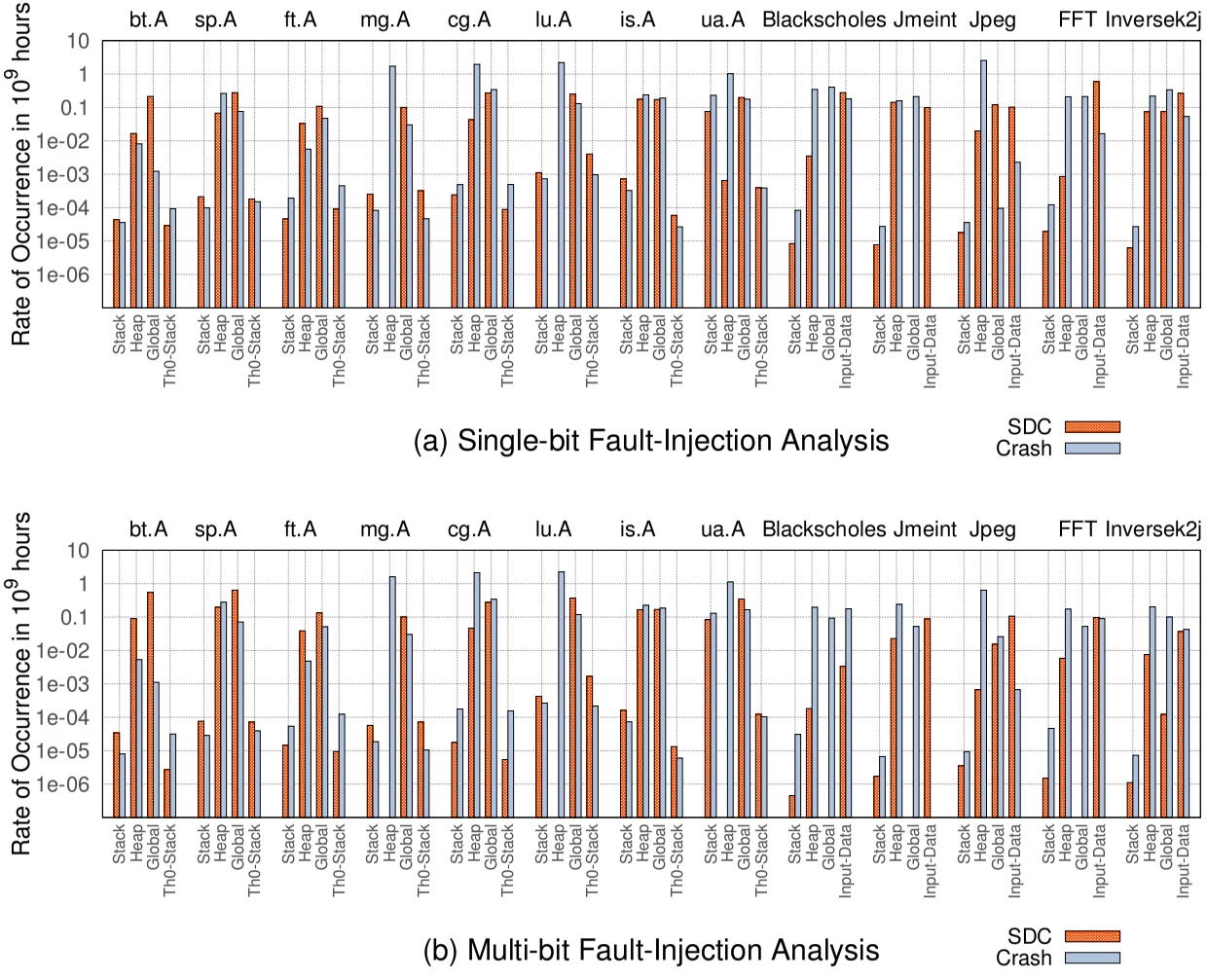


Figure 4.3: Rate of observing silent data corruption (SDC) or crash under: (a) single-bit and , (b) Multi-bit faults in each memory region considering 3D-stacked DRAM fault rates.

of bars in the figures corresponds to the crash rate and SDC rate of a data region. Next, we highlight some of the findings of this analysis which motivate Odd-ECC and guide our choices of fault tolerance levels selected per data region in our evaluation in Section 4.5.

For some NPB applications, *i.e. mg* and *lu*, single-bit faults in the heap have a SDC rate close to zero, while the crash rate is high. Such low SDC rates can be possible in some application when the algorithm inherently tolerates wrong values, for instance in iterative algorithms that converge to a solution as explained in [127]. A similar trend is observed for some AxBench applications, *i.e. Blackscholes*, *Jmeint* and *FFT*, where single-bit fault on the global region will cause a crash in most cases. In these cases, the heap can have a reduced level of fault tolerance, if the goal is to avoid SDC. For multi-bit faults, the same trend is observed for even more applications, e.g. *ua*, leading to the same conclusion. Considering the fault modes in Tables 4.2 and 4.3, the probability of a

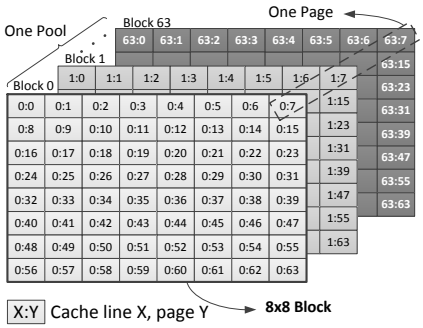
Table 4.3: Failure rates for 3D-Stacked DRAM [73], scaled based on our example 3D-stacked configuration. Effective FIT rates are calculated according to correction/detection capabilities of two protection schemes, i.e. T1 and T2, based on estimated coverage reported in [74, 126].

Fault Mode	Fault rate (FIT/Mbit)	Effective FIT rate(FIT/Mbit)	
		With T1-ECC	With T2-ECC
Single-Bit	0.03	0	0
Single-Column	7.60E-03	1.14E-03	0
Single-Bank	2.35E-03	2.35E-03	0
Single-Die	4.40E-04	4.40E-04	0
Single-Vault	1.20E-04	1.20E-04	1.20E-04

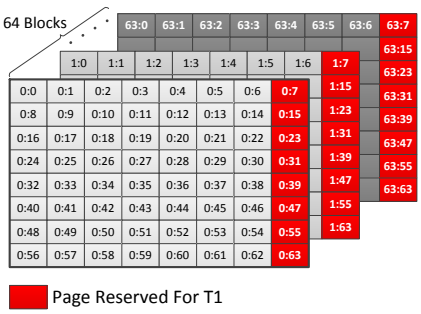
multi-bit fault in DRAM is more than 2$\times$ lower than the probability of a single-bit fault. However, due to the greater impact of multi-bit faults on the applications' behavior, the rate of crash and SDC are comparable with those observed in single-bit faults.

A careful analysis of the AxBench applications reveals that, almost in all cases, the input-data region has the highest SDC rate under both single and multi-bit faults. Consequently, a proper protection scheme for this region should be able to tolerate both single-bit and multi-bit faults. Moreover, the stack region of AxBench applications has higher SDC rate when single-bit faults are injected compared to multi-bit faults. This can be explained by the lower probability of multi-bit faults to happen. Therefore, for this region a simpler protection scheme tailored for single-bit correction would be sufficient rather than stronger protection schemes.

Finally, almost in all cases, the stack region has the lowest SDC and crash rate among regions. However, given the fact that stack sizes are typically very small, strong protection against faults can improve the system reliability with relatively low storage cost.



(a) T0 Pool



(b) T1 Pool

(c) T2 Pool

Figure 4.4: Placement of pages and cachelines in one pool composed of 64, 8×8 blocks.

4.2 Odd-ECC Memory Reliability

Odd-ECC introduces a new DRAM data placement that offers different protection levels to be chosen on demand for each physical page. Odd-ECC reserves the space required for storing the ECC bits of a particular protection scheme in a dynamic manner based on the user/program choice. Reserving space for the ECC is handled by the OS. ECCs are stored

in separate pages that are marked unavailable for the user as described in Section 4.2.4.3. In order to reduce the storage and access latency overheads, 4KB pages are grouped in pools of 64 pages (256KB in total), each pool having a particular protection level. Then depending on the protection offered, a number of pages in a pool is reserved to store the ECCs. Thereby, ECC storage overhead is economized by being proportional to the protection level required by the particular data. Although, ECCs are hidden in separate physical pages, Odd-ECC placement allows them to be aligned with the data they protect, making it simple for the memory controller to retrieve them with the same access latency as in an equivalent flat protection scheme. In other words, on the one hand, for the memory controller knowing the protection-level of an accessed cacheline is sufficient for finding its corresponding ECCs, on the other hand, for the OS all ECCs in a pool are stored in separate pages unavailable to the user.

Odd-ECC makes use of three different protection schemes, namely *Tier Zero* (T0), *Tier One* (T1), and *Tier Two* (T2), where T0 provides no protection, T1 provides lower level protection (single-bit correction, multi-bit detection, and T2 provides higher level protection (adding multi-bit correction to T1). Odd-ECC can be used in various memory architectures. For simplicity, we first provide a generic view of the scheme using an abstract view of a DRAM block and then explain how this is used in 2D and 3D-stacked DRAMs.

4.2.1 Odd-ECC Data Layout

Let us consider a generic abstract view of a DRAM block composed of rows and columns. As illustrated in Figure 4.4, Odd-ECC is applied in two-dimensional 8×8 blocks of 8 rows, each row storing 8 cachelines (512B). Each cacheline in a block belongs to a different page. A page has 64 cachelines found in 64 different blocks at the same block-position. A pool of pages is composed of 64 blocks, each containing a different cacheline from 64 different pages. This generic view of a pool, is also the view of an unprotected pool, called Tier Zero (T0) pool, shown in Figure 4.4a. So, a T0 pool offers all 64 pages to the user without any reliability provisions.

4.2.2 Odd-ECC Tier One (T1)

Tier One (T1) offers detection and correction of single-bit faults, and in addition detection of larger granularity (multi-bit) faults. As illustrated in Figure 4.4b, a block with data protected in T1 dedicates one cacheline per row to store ECCs, which protect the remaining 7 cachelines on the same row. From the OS view this means that a pool with T1 dedicates pages 7, 15, 23, 31, 39, 47, 55, and 63 for ECC storage. Concisely, the available capacity in T1 pool is reduced to 56 pages, having a data-to-ECC ratio of 14.28%.

In practice, a fraction (8B) of that ECC cacheline stores code that protects one of the other seven cachelines in the same row. That leaves 8 out of the 64B in the eighth cacheline unused. In order to simplify and make more efficient the task of the memory

Table 4.4: Starting column address for unprotected (T0) and T1 protected cachelines.

Starting Address for Unprotected (T0) Cachelines					Starting Address for T1 Protected Cachelines				
CL#	Address (Byte)	Address [8:0] (Binary)			CL#	Address (Byte)	Address [8:0] (Binary)		
0	0	000	000	000	0	0	000	000	000
1	64	001	000	000	1	72	001	001	000
2	128	010	000	000	2	144	010	010	000
3	192	011	000	000	3	216	011	011	000
4	256	100	000	000	4	288	100	100	000
5	320	101	000	000	5	360	101	101	000
6	384	110	000	000	6	432	110	110	000
7	448	111	000	000	-	504	111	111	000
A	$A \times 64$	$b_8b_7b_6$	000	000	A	$A \times 72$	$b_8b_7b_6$	$b_8b_7b_6$	000

controller, these 8B of T1 should be placed next to the cacheline they protect, rather than at the end of the block-row. In effect, the starting column of each cacheline needs to be a modulo of 9 (9×8 Bytes), instead of a modulo of 8, used in T0. It is preferable that this is abstracted from the OS, in order to be able to pack the T1 ECCs in separate pages, and be handled by the memory controller. On the one hand, the OS uses addresses that assume T1 ECCs are placed at the space of the eighth page of the block row. On the other hand, the memory controller should find the T1 ECCs next to the cacheline they protect by re-coding the address provided by the OS.

Re-coding the physical (column) address bits, sent by a memory request and received by the memory controller, which point to the starting byte of a cacheline in a row, introduces some complexity as it is not a power-of-two computation anymore [74]. More precisely, finding the starting address of each cacheline in a row will require either division or modulo operation [68, 128]. In Odd-ECC, we found a way around this restriction, which fits well with our 8 cachelines (512B) length of the block-row. Essentially, the 512B require 9 column bits to be addressed. As shown in Table 4.4, a starting address in T0, which is a multiple of 64, has bits 5:0 zero, and bits 8:6 indicate one of the 8 stored cachelines. By repeating column address bits 8:6 in bits 5:3 the new 9 column address bits point to a 72B (cacheline including its ECC) stride, as illustrated in Table 4.4. This reduces the memory controller complexity of re-coding to a single 2-to-1 multiplexer which selects the correct address bits based on the protection level (T0, T1) of the accessed data.

Without loss of generality, in our T1 implementation we select an ECC coding scheme from prior work that fits our single-bit correction and multi-bit detection requirements. As mentioned before, in T1 each 64B cacheline is protected separately. Therefore, we can use the ECC of [73] where a 64B cacheline is protected by 8B of mixed-ECC. For T1 we propose using low overhead 16bit SSC-DSD (Single Symbol Correct, Double Symbol Detect) [74], with strong 32bit CRC (Cyclic Redundancy Check) detection code. Overall, the T1 ECC for a 64B cacheline forms up to be 48 bits (6B), leaving 16 bits (2B) unused.

4.2.3 Odd-ECC Tier Two (T2)

Odd-ECC Tier Two (T2) is meant for correcting larger granularity, multi-bit faults[11]. T2 builds on top of T1 correcting large granularity failures detected by the T1 codes. It uses one additional cacheline (in the last block-row) to protect the above seven vertically neighboring cachelines as depicted in Figure 4.4c. T2 codes are constructed by computing the bit-wise parity of the above 7 cachelines allowing to reconstruct entirely one of them in case of a large granularity failure. In addition to the pages reserved for T1, a pool with T2 protection reserves the eighth row of each block for the parities, which corresponds to pages number 56-63 in a pool. Consequently, the available capacity in a T2 pool becomes 49 pages having a data-to-ECC ratio of 30.6%. A memory controller can access the T2 code that protects the above 7 cachelines by using the same column address bits and by finding the eighth block-row, the calculation of which depends on the considered memory architecture as explained next. The re-coding needed for finding the T2 codes address is as simple as replacing the 3 LSBs of the row address bits with “111".

4.2.4 Hardware/Software Modifications

In order to support different fault tolerance levels, Odd-ECC requires some modifications in the memory hardware and the OS. For each memory request, the memory system needs to identify the fault tolerance level for the corresponding memory content, and trigger the corresponding required action for error detection and/or correction. In addition, the OS is responsible for creating and managing pools of 64 pages and reserving space for ECC-bits depending on the ECC level, which is not visible to the user.

4.2.4.1 Memory Controller Modifications

Odd-ECC requires from the memory controller to (i) re-code the physical addresses for T1 and T2 accesses in order to find the realigned cachelines with their T1 codes, (ii) to perform for every T2 write access an extra read modify and write of the respective parity, and (iii) to support the detection and correction computations needed for the T1 and T2 memory accesses.

Memory accesses on T1 or T2 protected cachelines are placed every 72 consecutive bytes to include their respective T1 code. The memory controller should modify the address provided by the processor to find the correct beginning of the cacheline accessed. As described earlier re-coding requires copying 3 bits of the address adding a complexity of a 2-to-1 multiplexer. In addition, a T0 access requires burst of eight to read only the cacheline data, while accessing protected data requires longer burst to include the T1 codes. Similar to prior work [129, 130], Odd-ECC memory controller supports dynamic burst lengths of eight and ten[12], needed for accessing T0 and T1/T2 cachelines, respectively.

[11] Up to complete failure of a single DRAM chip when employed on 2D-DRAMs

[12] Reading 8B T1 ECC requires one additional half-cycle. Inevitably, during the other half-cycle the channel is not used.

In Odd-ECC, the T2 code for a group of seven cachelines is placed in a fixed position, *i.e.* on the eighth row of an 8×8 block. T2 code needs to be accessed either when T1 code detected a fault or in write accesses of T2 protected data. When writing a T2-protected cacheline, its respective parity (as well as the old cacheline value) need to be read, updated based on the new value of the cacheline and written back. These extra accesses are supported by the Odd-ECC memory controller. The memory controller can find the address of the T2 code by masking specific bits of the respective cacheline address (the 3 least significant bits of the row address part) to "111". In practice, this is implemented with a bitwise OR operation of the cacheline address bits and a fixed T2-MASK vector[13].

Finally, the Odd-ECC memory controller is modified to support T1 and T2 ECC logic, *i.e.* generator, checker, and corrector [14]. The complexity of these computations is well studied and implemented in existing solutions.

4.2.4.2 Page Tables Modification

The memory controller needs to know the protection level of each requested cacheline, which depends on the page it belongs to. To this end, two extra bits containing fault tolerance level information are added to each Page Table Entry (PTE) and they become available to the processor Translation Lookaside Buffer (TLB) entries in every memory access. These two extra bits can be easily supported since most of the modern architectures have already some spare/user bits (14 spare bits in $x86$-64 architecture) in their PTE [132]. These two bits are carried along in the memory hierarchy, including the cache levels, until they are used by the memory controller. Memory accesses from external devices such as GPUs, network cards, and DMA engines are supported via the IOMMU page tables, the same way they are supported for processors. Based on the fault tolerance level of a cacheline, the memory controller can find: *(i)* the starting column address, and *(ii)* the T1 and T2 (if applicable) codes for each cacheline.

4.2.4.3 OS Support

The changes in the OS to support the different memory fault tolerance levels are minor. As described before, the granularity of the physical memory is increased to blocks of 256KB, which corresponds to 64 × 4KB frames (pages). Out of these 64 frames, some may not be available to the users, since they may be reserved for fault tolerance T1 or T2 codes. While the OS still returns memory to the application at page granularity, the physical space is managed at the 256KB block granularity. In this work we consider three degrees of protection thus, the OS handles three different types of blocks: (i) T0 or non-protected memory blocks; (ii) T1 memory blocks; and (iii) T2 memory blocks. Consequently, the OS keeps three empty frame lists, one per type of block, in addition to a free empty block list. When a memory request reaches the OS, it includes not only the memory size

[13] T2-MASK depends on the memory system configuration.

[14] In 3D-stacked DRAMs the ECC logic is implemented on the logic die at the bottom of the stack. [131].

requested but also the protection level. Therefore, the space requested will be allocated from the corresponding pool. If no frame is available in the corresponding pool, then a new 256KB block is allocated from the empty block list and its frames (excluding the ones reserved for fault tolerance) are added to the corresponding frame type list. The fault tolerance level of the memory requests is provided by the *OS memory request API*, which is a simple extension of the traditional API. Dynamic memory allocation can be performed using a *p-malloc* function call that adds an extra parameter indicating the protection level for the memory requested. For static memory allocation (e.g. global or stack) an extra option describing the protection level of the different static memory spaces is passed to the compiler so that it augments the executable with the extra information to be used by the loader when requesting memory.

Table 4.5: Example 2D-DRAM memory system configuration

Example configuration with DDR3-1600 (x8)			
Rows	16K	Ranks	2 per DIMM
Columns	1K	DIMMs	2
Banks	8	Channel	1
Device Size	128MB	Cacheline(CL)	64B
Channel width	64-bit	Total Capacity	4GB

4.3 Odd-ECC in 2D DRAMs

Considering the generic 8×8 DRAM block used in the previous section for explaining our approach, we describe next how Odd-ECC is applied in conventional 2D DRAM DIMMs. Figure 4.5 illustrates an example 4GB memory system of two DRAM DIMMs with two ranks per DIMM. A typical 64B (Byte) cacheline is spread in 8B segments over eight chips in one rank. Spreading a cacheline in this manner, makes it possible to read the whole 64B in four DRAM cycles (*i.e.* eight half-cycles) where in each half-cycle, 8b (bits) are read from each of the chips simultaneously and sent to the memory controller using a 64b channel. Considering this arrangement, a row of 1KB in one DRAM chip can accommodate 8B segments of 128 64B cachelines. Without loss of generality, Table 4.5 provides the 2D DRAM configuration used to exemplify our approach.

As the cachelines are spread across all 8 chips in a Rank (and bank), the 8×8 block is also spread across the 8 chips containing 8 cachelines in a row and spanning 8 consecutive rows (row i to row i+7) in the same Bank. This is illustrated in Figure 4.6. We conceptually split the rows of each DRAM chip into 64B boundaries. Each 64B boundary can store eight 8B segments of eight cachelines. A 64B boundary over eight neighboring rows and across the 8 chips forms a block of 8×8 cachelines. In our 2D DRAM example, cachelines of a page are spread across 2 DIMMs, 2 Ranks, and 8 Banks. Since this parallelism (of 32) is not enough, two cachelines of the same page are stored in different consecutive blocks in the same row, as shown in Figure 4.6. Then, a pool of 64 pages is formed by 64 blocks stored in pairs across 2 DIMMs, 2 Ranks, and 8 Banks.

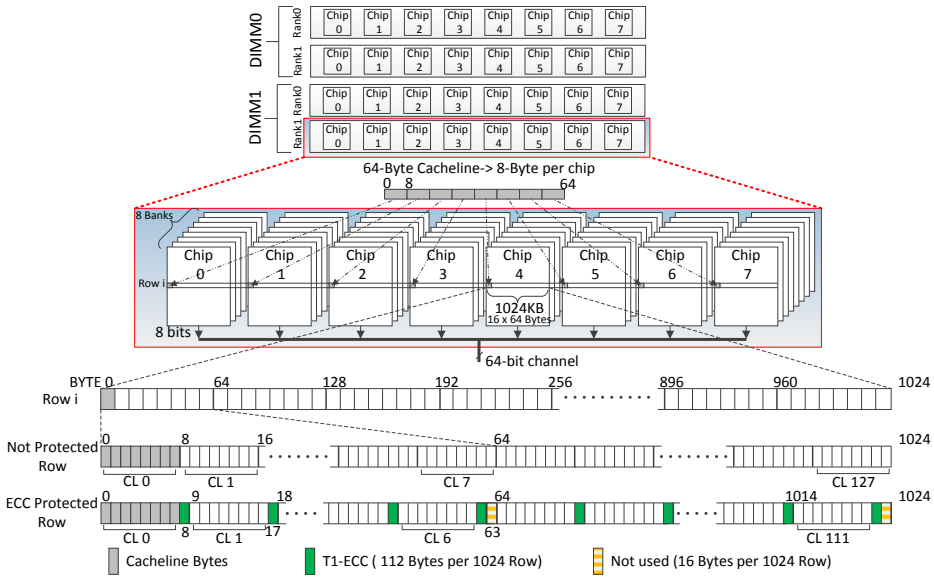


Figure 4.5: Overview of 2D-DRAM memory system.

The above defines T0 of Odd-ECC in 2D DRAMs. T0 offers all 64 pages in a pool to the user without any protection.

4.3.1 T1 ECC

For T1 the last (eighth) cachelines in each row of a block stores the ECCs of the other 7 cachelines. We dedicate 8B of T1 to each of the seven 64B cachelines which leaves 1B left unused. Similar to the data, the 8B T1 ECC is divided into 8 segments, spread across 8 chips, and stored alongside the data, occupying one additional column in the row on each chip. As described in the previous section, the column address in every chip needs to be re-coded for T1. In T0 the column address points in the beginning of an 8B cacheline segment, but for T1 it should have a stride of 9B. Applying the method explained in the general Odd-ECC description, the stride of 9 is calculated by copying column address bits 5:3 to bits 2:0 as shown in Table 4.6. As described in previous section, the T1 ECC for a 64B cacheline forms up to be 48b (6B), leaving 16b (2B) unused. Effectively this means every 8B data segment on each chip has 6b of T1 with 2 spare bits. In 2D-DRAMs Odd-ECC, these 2 spare bits are used to store the parity of each 8B segments, one per 32b.

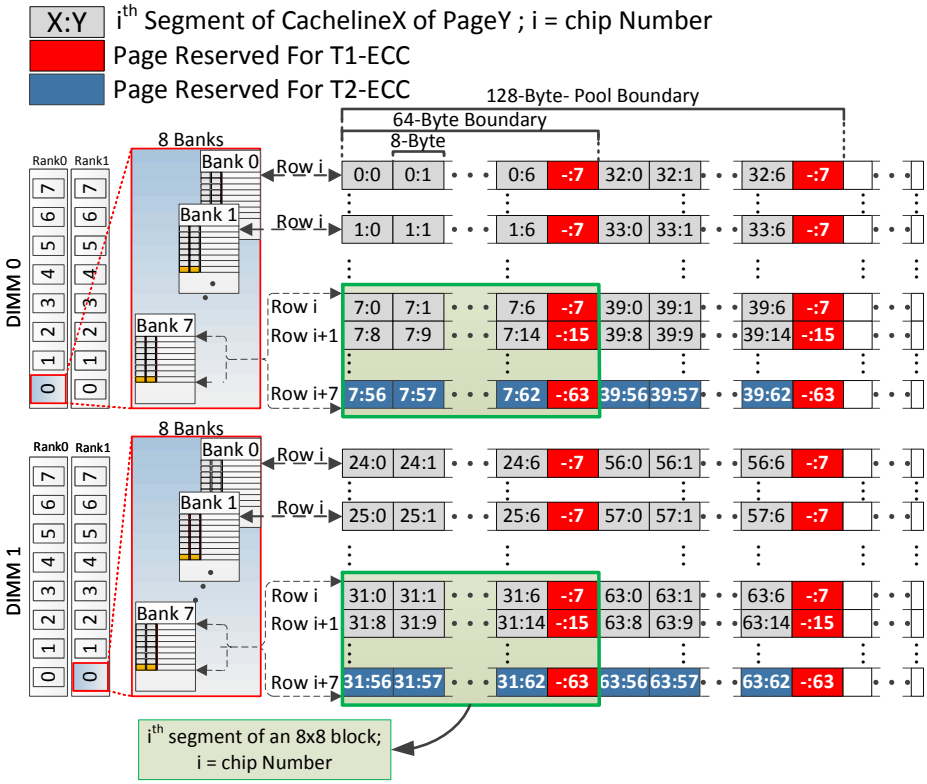


Figure 4.6: The physical layout of pages and cachelines for the 2D-DRAM memory system configuration described in Table 4.5. The placement starts from DIMM0/Rank0/Bank0 and ends at DIMM1/Rank1/Bank7.

4.3.2 T2 ECC

T2 codes, responsible for correcting multi-bit faults, are placed in the eighth row of every block as shown in Figure 4.6. Similar to LOT-ECC [66], in 2D DRAMs Odd-ECC T2 code is the bit-wise parity of the 8B segments of seven vertically neighboring cachelines. In particular, a single parity P_i, where i is the chip number, is stored in row 8 of the block and protects cacheline segments X_y stored in rows $X = 1, 2, ..., 7$ in chips $y = (i + X)\ mod\ 8$, resembling RAID 5 [133]. Simply stated, a single parity protects a single cacheline segment in each chip, as shown in Figure 4.7. This way, T2 codes can be used to reconstruct the contents of a block in case of a single chip failure.

In Odd-ECC, T2 parity covers only the data of the 7 protected cacheline 8B segments and not their additional 1B T1 ECC. Consequently, for each T2 parity stored in the last row of a block there is a leftover ninth byte. In order to provide chipkill-level protection,

Table 4.6: Starting column address for 2D-DRAM memories in case of unprotected (T0) and T1 protected cachelines with 1-KB row size.

Starting Column Address for Unprotected Cachelines					Starting Column Address for T1 Protected Cachelines				
CL#	Column (Byte)	Column [9:0] (Binary)			CL#	Column (Byte)	Column [9:0] (Binary)		
0	0	0000	000	000	0	0	0000	000	000
1	8	0000	001	000	1	9	0000	001	001
2	16	0000	010	000	2	18	0000	010	010
3	24	0000	011	000	3	27	0000	011	011
4	32	0000	100	000	4	36	0000	100	100
5	40	0000	101	000	5	45	0000	101	101
6	48	0000	110	000	6	54	0000	110	110
7	56	0000	111	000	-	63	0000	111	111
8	64	0001	000	000	7	64	0001	000	000
9	72	0001	001	000	8	73	0001	001	001
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i	B	$b_9b_8b_7b_6$	$b_5b_4b_3$	$b_2b_1b_0$	$j = \lceil i - (i/8) \rceil$	$j * 9 + \lfloor i/8 \rfloor$	$b_9b_8b_7b_6$	$b_5b_4b_3$	$b_2b_1b_0$

each 8B T2 parity should be followed by 1B interleaved parity of its own bytes, similar to the detection code used in [74]. We use the above ninth leftover byte to store such 1B parity and detect faults in each 8B T2 code.

In 2D DRAMs, Odd-ECC T2 is placed on the eighth row of an 8×8 block, protecting the seven cachelines above that row. Thus, the corresponding T2 of each cacheline will be on the same bank and same column, but on a different row. The address of T2 for every cacheline can be directly found by using a fixed T2-MASK vector[15] on the cacheline address, changing the three LSBs (least significant bits) of the row address to "111".

T2 is added on top of T1. Hence, large granularities of faults over the whole cacheline can be detected using the strong T1 CRC code. But, in order to use T2 to correct multiple-bit (*i.e.* burst) faults up to a complete chip failure, we need a mechanism to locate the particular faulty cacheline segment, located in a single chip. Considering independent faults constrained within chip boundaries and the fault modes listed Table 4.2, we describe next the fault locating mechanism with an example using Figure 4.7. Let us assume that CRC has detected a burst fault in cacheline A, which is uncorrectable by T1. As a first step for locating the fault, the 2-bit parity of 8B segments are used, which are stored in T1. However, due to their low detection capability, T1 parity bits might not reveal which segment of cacheline A is faulty. As a second resort, the T2 codes are checked. Each 8B T2 segment is augmented with an additional 1B interleaved parity to check its correctness. In case of a fault in a T2 parity, e.g., in P_4, it shows that chip 4 is encountering a burst fault [16]. Consequently, the correction mechanism starts regenerating the A_4 segment. Otherwise, if all T2 parity codes are fault-free, the fault locating mechanism proceeds with regenerating all segments of the T2 parity by reading the cachelines in the other 6 rows B, C, D, E, F, and G. Comparing the new T2 segments with the previously ones

[15] In our example 2D-DRAM memory system the T2-MASK vector is `0x1C0000`

[16] This event can be associated with multiple row, column, bank or complete chip failure.

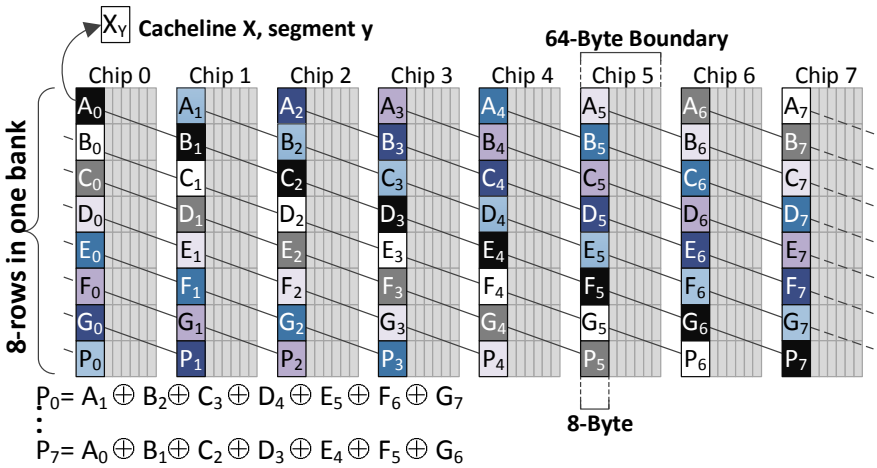


Figure 4.7: The structure of T2 ECC in 2D-DRAM.

results in at least one mismatch, since a segment of cacheline A is faulty. In case there is a single parity mismatch, the fault is located in the specific segment of cacheline A that is used for computing that particular T2 parity [17]. Otherwise, if multiple parities mismatch with the newly calculated ones, a simple analysis leads to the detection of the faulty cacheline segment. For instance, if both P_0 and P_1 do not match their old values, segments A_1 and A_2 are the two candidates to be faulty. However, considering our assumptions for independent fault modes, the only way that both P_0 and P_1 can give a mismatch is that A_2 and B_2 on chip 2 are encountering burst faults [18]. In such case, the A_2 segment is the faulty one and the correction mechanism will regenerate it. Regenerating a faulty segment is done using the respective T2 parity and the six other cacheline segments similar to LOT-ECC [66].

4.3.3 Address Mapping

We explained how Odd-ECC can be employed on 2D-DRAMs by spreading 64 pages of each pool and 64 cachelines of each page in 8×8 blocks. Clearly, this arrangement requires modifications in the way a physical memory address is mapped to DRAM. Considering the generic 8×8 block in Section 4.2, we describe the address mapping of our example 2D-DRAM configuration below. As shown in Figure 4.8(a), 18 bits are needed to address any byte within a 256KB pool. Out of these 18 bits, the LSBs 5:0 are the byte offset, bits 11:6 indicate the cacheline in a page and MSBs 17:12 indicate the page in the pool.

[17] This event can be associated with single row failure.
[18] This event can be associated with multiple row or column failure.

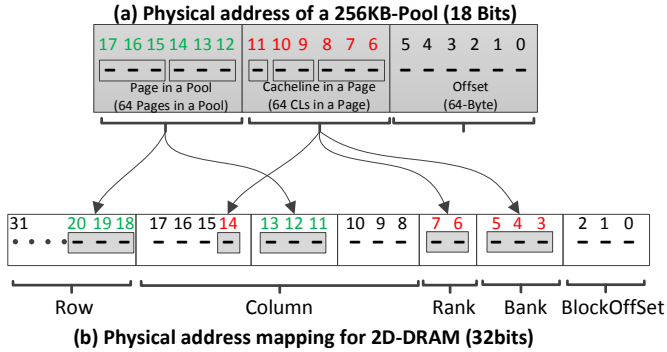


Figure 4.8: 2D-DRAM address mapping based on the memory system configuration of Table 4.5.

This means, bits number 17:12 address all cachelines within one 8×8 block, and bits number 11:6 address each of the 64, 8×8 blocks. In essence, Odd-ECC only requires changes in the way these 12 bits of a pool address (*i.e* 17:12 and 11:6) are mapped to the memory, because all requests to the memory are cacheline aligned. Figure 4.8(b) illustrates how each bit of a pool's physical address is mapped to the rows, columns, ranks, and banks of our example 2D-DRAM configuration. Since all eight chips in a rank use the same address, for simplicity we describe the mapping for one chip, which stores one 8B segment of each cacheline. Each pool spans 128 bytes of 8 rows across all banks, ranks and devices. The first 32 cachelines of a page are spread over one row and column of all banks and ranks (*i.e* bits 7:3). The remaining 32 cachelines of a page are placed in the next neighbouring 8×8 block defined by bit 14. This mapping is repeated for every group of 8 pages until all 8 DRAM rows of a pool are mapped. In essence, this mapping is a special case of cache-line-interleaving address mapping, *i.e. Row::Column::Rank::Bank::BlockOffSet*, which is a typical mapping for closed-page policy in 2D-DRAMs [134]. Odd-EEC 2D-DRAM mapping is not expected to cause performance issues as it offers maximum bank and rank parallelism.

4.3.4 Extensions to Other 2D-DRAM Memory System Configurations

We used an example memory configuration for describing how Odd-ECC can be employed on conventional 2D-DRAM. However, Odd-ECC can be employed on different 2D-DRAM configurations using ×8 non-ECC devices. Depending on the available bank parallelism in the system, the only thing that changes is the pool boundary, *i.e.* the number of consecutive 8×8 blocks used for each pool. This means Odd-ECC can be employed even on a single non-ECC DIMM memory system, proving chipkill-level protection. Moreover, the reserved pages in Odd-ECC can be used with other error correcting codes, with different detection or correction capabilities, as long as the storage requirements do not exceed the available space. Finally, other common memory reliability techniques

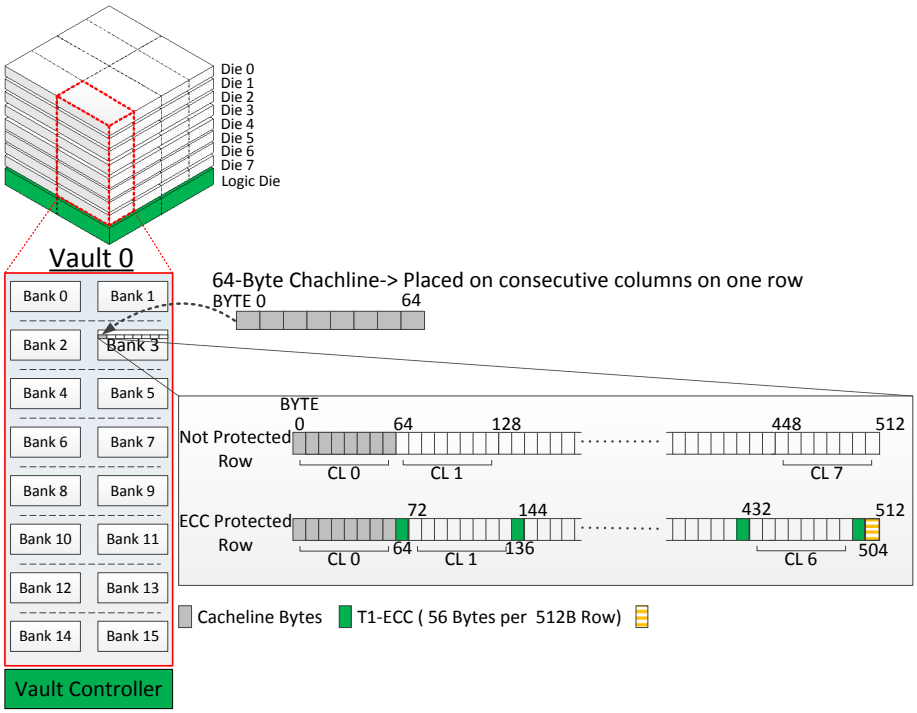


Figure 4.9: Overview of 3D-stacked memory system.

such as *memory scrubbing* [135], i.e. periodical read-rewriting of every memory location to avoid accumulation of memory faults, or *page retirement* policies [136], i.e. disabling faulty memory regions at the software level, can be applied in conjunction with Odd-ECC.

4.4 Odd-ECC in 3D-stacked DRAMs

Odd-ECC can be applied to 3D-stacked DRAMs, using the same generic 8×8 DRAM block of Section 4.2. In 3D-Stacked memories multiple DRAM dies are stacked on top of each other using *Through Silicon Vias* (TSVs). In current 3D-stacked standards like HMC [137] and HBM [138], the stack is divided into several vertical vaults. Each vault is a single memory channel with its own memory controller, placed on a logic die at the bottom of the stack. Each vault has multiple banks and on each bank DRAM cells are organized in rows and column similar to traditional DRAMs. However, the row sizes in 3D-stacked DRAMs are made shorter to save power [139]. Figure 4.9 illustrates an example 3D-stacked memory with eight dies and eight vaults. Each vault consists of 16 banks, two per die, with row size of 512B (Bytes). Contrary to 2D-DRAMs,

in 3D-stacked DRAMs the whole cacheline is placed in consecutive columns in one DRAM chip. This way, considering a typical 64B cacheline and 64b (bit) data-TSV channel, a cacheline can be accessed in four DRAM cycles, (*i.e.* eight half-cycles) where in each half-cycle, 64b are read from a same row in a bank and sent to the memory controller at the logic layer. With this arrangement, a row of 512B in one DRAM chip can accommodate eight 64B cachelines. Without loss of generality, Table 4.7 provides the 3D-stacked DRAM configuration used to exemplify our approach.

Table 4.7: Example 3D-stacked memory system configuration

Example 3D stacked DRAM configuration			
Capacity	8GB	# of Banks	16 per Vault
# of Dies	8	Row Size	512 Bytes
# of Vaults	8	Channel width	64 TSVs

Placing a cacheline in one row of one bank makes the mapping of the generic 8×8 block simpler. As illustrated in Figure 4.10, a block is placed in one vault, spread across one row in eight banks (either the odd or the even in our example), each bank on a different die. Since the row size is 512B, one entire row in a bank is effectively a row in the 8×8 block, which stores eight 64B cachelines. In our 3D-Stacked DRAM example, a pool is composed of 64 blocks spread across both the odd and even banks of 8 vaults (*i.e.* parallelism of 16). Since this parallelism (16) is not enough to form a pool, blocks of 4 consecutive rows in a bank belong to the same pool of 64 blocks, as shown in Figure 4.10. Then, a page has a cacheline in every bank of every vault, in 4 consecutive rows (same column).

The above organization defines how T0 of Odd-ECC is mapped to 3D-stacked DRAM. T0 offers all 64 pages in a pool to the user without any protection.

4.4.1 T1 ECC

Similar to 2D-DRAM, we dedicate 8B of T1 to each 64B cacheline out of which seven bytes are used to store the T1 ECC described in the previous section and one byte is left unused. The 8B T1 ECC is stored alongside the data, occupying one additional column[19] in a row on one bank. As described in the previous sections, the column address for accessing T1 protected cacheline needs to be re-coded to find the starting column address, which is now a multiple of 72B rather than a multiple of 64B in T0. Applying the method explained in the general Odd-ECC description, the stride of 72 is calculated by copying column address bits 8:6 to bits 5:3, as shown in Table 4.4.

4.4.2 T2 ECC

As shown in Figure 4.10, T2 codes are placed in the eighth row of every block. For 3D-stacked DRAM this means that T2 codes are always placed on the eighth die. Similar

[19]Column size is 8B.

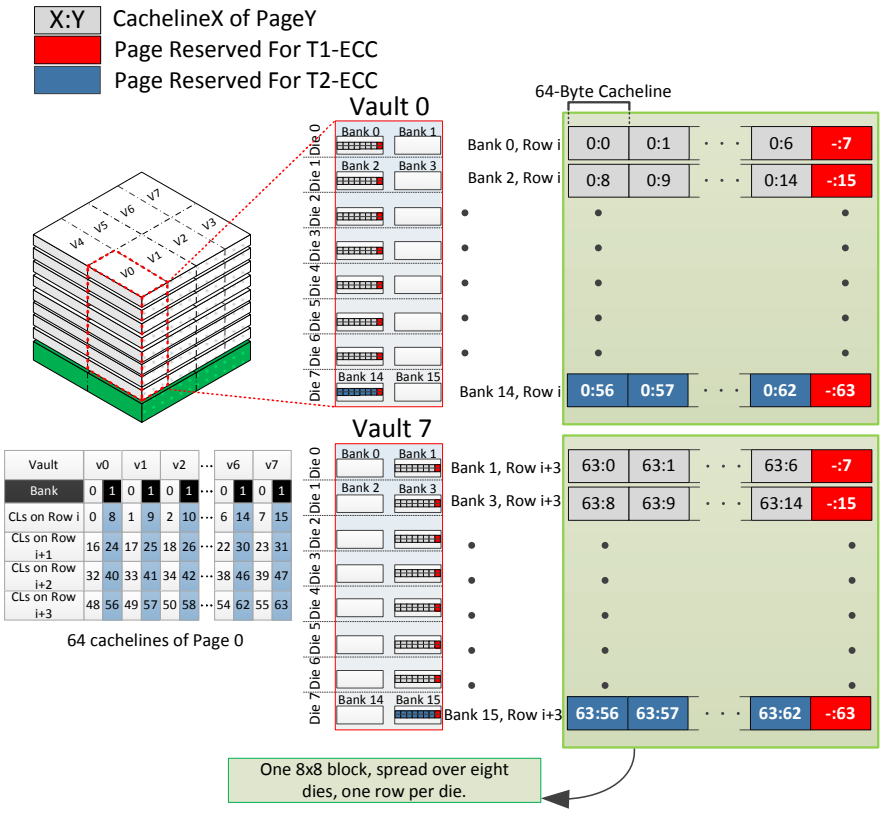


Figure 4.10: Page and cachelines mapping for 3D-stacked memory. With 8 dies, 8 vaults and 2 banks per vault, cachelines of one page occupy 4 rows on each bank (i.e. A pool boundary is 4 rows over all banks)

to Parity Helix [73], in 3D-stacked Odd-ECC, T2 code is the bit-wise parity of the seven, 64B cachelines, stored on a same row and column of seven banks in one vault, one bank on every die. Simply stated, a 64B parity stored on the eighth die, protects seven 64B cacheline stored on the same row and column of the other seven dies above. This way, T2 codes can be used to reconstruct the entire cacheline, even in case of a single die failure [20].

In Odd-ECC, T2 parity is covering only the data of the 7 protected cachelines and not their additional 8B T1 ECC. Consequently, for each T2 parity stored in the last row of a block, there are eight leftover bytes. The leftover bytes can be used to provide additional protection for the T2 code.

The corresponding T2 code for a T2 protected cacheline is on the same bank position

[20] In the current design, T2 does not protect against vault faults.

, same column, same row but on the eighth die. Thus, address of T2 code can be found directly by using a fixed T2-MASK. vector, changing the three die bits of the cacheline address to "111". In our example 3D-stacked memory system the T2-MASK vector is `0x1C00`.

T2 is added on top of T1. Hence, large granularities multi-bit faults over the whole cacheline can be detected using the strong T1 CRC code. Contrary to 2D-DRAM where T2 was protecting 8B segments of cachelines, in 3D-stacked DRAM T2 is constructed over the whole cacheline. Therefore, there is no need for locating the place of the fault in the cacheline. For a given cacheline, when CRC code detects a fault, uncorrectable by T1, the memory controller will read the T2 code and all other six cachelines, contributed in computing the T2. This is done by simply changing the die bits in the cacheline address. Regenerating a faulty cacheline is done by computing the bit-wise parity of respective T2 and the six other cachelines similar to Parity Helix [73].

4.4.3 Address Mapping

Similar to 2D-DRAM, employing Odd-ECC on 3D-stacked DRAM only requires changes in the way 12 bits of a pool address are mapped to the memory, *i.e* six bits page address (17:12) and six bits of cacheline address (11:6).

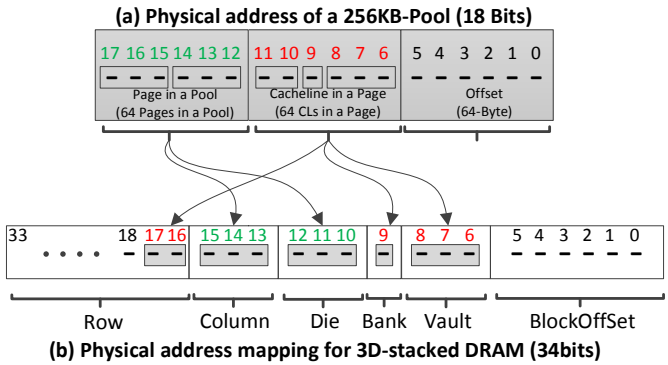


Figure 4.11: 3D-stacked address mapping based on the memory system configuration of Table 4.7.

Figure 4.11(b) depicts how each bit of a pool's physical address is mapped to our example 3D-stacked DRAM configuration. Each pool occupies 4 consecutive rows on every bank and vault. The cachelines of each page are interleaved across all vaults and banks (bits 9:6) on one die on the same column spanning over 4 rows (bits 17:16). The pages of a pool are spread across all vaults, dies (bits 12:10), and columns (bits 15:13) occupying 4 complete rows. This mapping, is a special case of default address mapping of HMC,*i.e. Row::Column::Die::Bank::vault::BlockOffSet*, which exploits the available bank parallelism inside the cube to avoiding bank conflicts [131]. Odd-ECC 3D-stacked

address mapping exploits maximum channel and bank parallelism and therefore should not affect the performance.

4.5 Evaluation

Using the same applications studied in Section 4.1 and guided by their MTTF analysis, we select combinations of fault tolerance levels (*i.e.* T0, T1, T2) for each data region to be supported by the proposed Odd-ECC scheme. These Odd-ECC setups are then compared with flat fixed memory fault tolerance levels in terms of the overall MTTF of the application at hand, as well as in terms of overheads in memory capacity, application performance and energy costs.

4.5.1 Experimental Setup

Performance and energy consumption of different memory protection options are evaluated using an in-house cycle accurate, execution driven multi-core simulation platform based on Pin [119] coupled with DRAMSim-2 [140], which is modified to implement Odd-ECC address mapping. DRAMSim-2 was also modified to support memory bursts of eight and ten. OS virtual to physical address translation is emulated in our simulator by keeping track of the virtual addresses and protection levels for each address.

For the common case reads and writes of a T1 protected cacheline, we consider an extra burst. The 8 Bytes of T1 can be accessed in half a cycle, while the other half cycle is enough for the detection mechanism based on [126]. For the common case writes, each T2 protected cacheline requires two reads and two write operations for updating the parity. Nevertheless, since writes are not on the critical path, T2 writes have limited performance impact. Prior works have proposed using a parity cache to reduce the performance impact of updating ECC codes [71, 72, 74]. However, since our goal is to compare between different protection setups, our evaluation does not consider this optimization.

Table 4.8 summarizes different system parameters used in our evaluation. For 2D and 3D-stacked DRAM memory systems, we used the same memory configurations as given in Table 4.5 and Table 4.7, respectively. 2D-DRAM timing and current parameters were obtained from DDR3-1600 data-sheet. For 3D-stacked DRAM we considered the timing parameters used in [139] and for energy measurement we used current value from DDR4-2400. CACTI 6.5 [141] is used for obtaining the cache latency.

In the evaluation we used the same sets of applications as in our sensitivity analysis. A brief description of these applications is shown in Table 4.9. The results presented are obtained from the simulation of one billion instructions after a warm-up of one hundred million instructions.

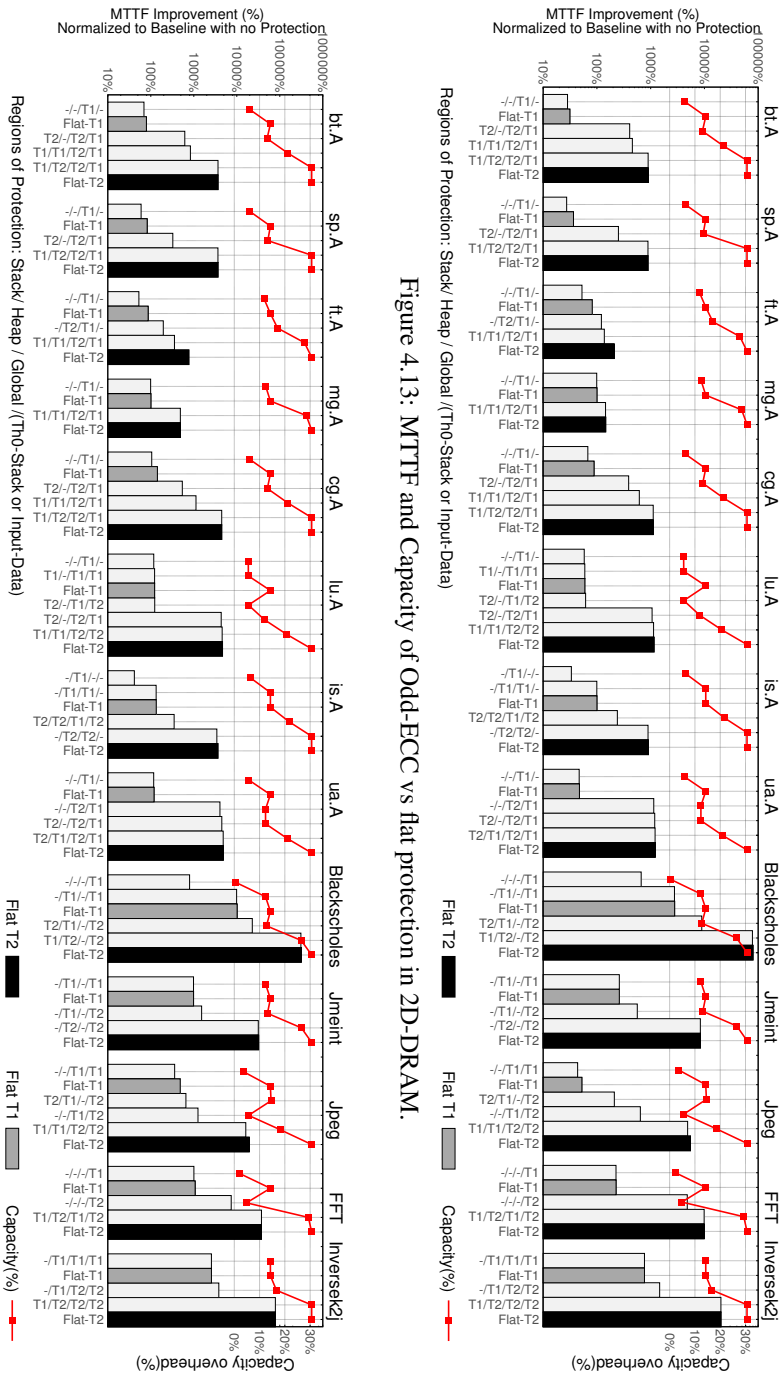


Figure 4.13: MTTF and Capacity of Odd-ECC vs flat protection in 2D-DRAM.

Figure 4.14: MTTF and Capacity of Odd-ECC vs flat protection in 3D-stacked DRAM.

Table 4.8: System parameters for our evaluation.

Processors		
Core	8 cores, 3.2Ghz , OoO 4-way superscalar	
LLC	8MB, 16-way Lookup latency 6, Read latency 10	
Memory Systems		
	2D-DRAM	3D-Stacked DRAM
Frequency/channel	800 Mhz	1.2 GHz
tRCD/ tRP/ tCL/ tRAS	11/11/11/28	17/17/17/34
Row Buffer Policy	Closed-page	Close-paged

Table 4.9: Applications used for Odd-ECC evaluation and their data regions sizes.

Application	**Description**	**Workload**	**Region Size(MB)**			
			Stack	Heap	Global	Th0-Stack/ InputData
bt	Block tri-diagonal solver	Class A- 8Th	0.008	60.3	44.4	0.003
sp	Scalar penta-diagonal solver	Class A- 8Th	0.005	60.8	46.5	0.003
ft	Discrete 3D Fast Fourier Transform	Class A- 8Th	0.007	68.0	322.9	0.004
mg	Multi-grid on a sequence of meshe	Class A- 8Th	0.044	67.4	453.0	0.008
cg	Conjugate gradient	Class A- 8Th	0.006	63.1	47.0	0.003
lu	Lower-Upper Gauss-Seidel solver	Class A- 8Th	0.062	67.6	43.0	0.16
is	Integer Sort	Class A- 8Th	0.006	56.3	68.0	0.003
ua	Unstructured adaptive mesh	Class A- 8Th	0.127	54.8	37.1	0.008
Blackscholes	Financial market model	100K points	0.003	8.5	2.0	3.4
Jmeint	Triangle intersection detection	100K pairs	0.002	19.1	2.0	6.8
JPEG	JPEG encoding	512x512 pixels	0.003	11.9	2.0	2.9
FFT	Radix-2 Colley-Tykey Fast Fourier	524,288 F.P	0.004	19.3	2.0	4.0
Inversek2j	Inverse kinematics for 2-joint arm	100k coordinates	0.004	13.7	2.0	1.5

4.5.2 Experimental Results

We first present the MTTF and capacity overhead for the different protection schemes of each application. The baseline considered is the non-protection setup (T0). In addition, we consider two reference cases: flat-T1 (*i.e.* all memory regions are protected by T1) and flat-T2 (*i.e.* all memory regions are protected by T2).

The results are depicted in Figures 4.13 and 4.14 for the 2D and 3D setups, respectively. For each application the results of different fault tolerance setups are shown. The setups are represented in terms of the fault tolerance level provided to the different regions. For example T2/-/T2/T1 represents the setup for T2 protection of the Stack, non-protected (T0 denoted as "-") for Heap, T2 for Global, and T1 for the Stack of Thread 0. For each setup we show a bar representing the MTTF improvement compared to the baseline. The values for the bars are presented in a logarithmic scale

on the y-axis on the left. In addition, for each setup we show a point representing the capacity overhead, which values are presented in the linear scale on the y-axis on the right. For each application, the results presented are ordered by increasing MTTF. Note that although more experiments where performed, only the most promising Odd-ECC setups are presented.

The first observation is that although the MTTF values of 2D-DRAM and 3D-stacked DRAM are different, their trend is similar. The difference in absolute MTTF values is due to the different fault rates and fault models listed in Tables 4.2 and 4.3. Thus our comments focus on the 2D results but apply to the 3D-stacked results as well.

Odd-ECC can be used to achieve, compared to a flat memory fault tolerance scheme, either of the following two objectives: (1) similar MTTF with a reduced memory capacity overhead, or (2) higher MTTF for similar memory capacity constraint.

Considering the above, we can observe that for 4 out of the 13 applications Odd-ECC does not yield significant gain. For these applications (*ft*, *mg*, *is* and *Inversek2j*) there is no Odd-ECC setup that has substantially better MTTF-capacity tradeoff compared to a flat scheme.

For *Blackscholes* and *Jmeint* we can observe that there is an Odd-ECC setup with better MTTF than the flat-T1 and even less memory capacity overhead. For example, in *Blackscholes* we achieve a 1.5× MTTF improvement and 10.8% memory overhead reduction for the T2/T1/-/T2 setup, which can be explained by the very low SDC rate of the global region, as shown in Figure 4.2.

For 5 of the 13 applications (*bt*, *sp*, *cg*, *lu* and *FFT*) we observe that both goals are achieved for some Odd-ECC setup when compared to the flat-T1. For example, in *FFT* on the one hand, the -/-/-/T1 achieves similar MTTF (-0.6%) with the flat-T1 with a reduction of 84.2% in the memory capacity overhead. On the other hand, the -/-/-/T2 setup increases MTTF by a factor of 8.7× compared to flat-T1, with a 66.1% lower memory capacity overhead. High SDC rate of Input-data region explains this outcome.

Finally, for *ua*, and *Jpeg* we can observe not only both goals are achieved when compared to flat-T1, but also there is a reduction of the memory capacity overhead for similar MTTF when comparing with flat-T2. For example, for *ua*, when compared to flat-T2, T2/-/T2/T1 achieves a similar MTTF (-1.8%) with 59.6% reduced memory capacity overhead. For the same application *ua*, when compared to flat-T1, on the one hand -/-/T1/- achieves similar MTTF (-0.5%) with a 59.6% reduction in the memory capacity overhead. On the other hand, -/-/T2/T1 achieves an increase in the MTTF of 7.0× and a reduction of 13.5% in capacity overhead, which again shows the impact of high SDC rate in the global region, under both single and multi-bit faults.

Our goal in this work is to reduce the storage costs of ECCs in memory while still offering similar MTTF. When achieving this goal, Odd-ECC setups have lower performance and energy overheads than the flat protection schemes with equivalent MTTF. In general, performance and energy overheads of a protection scheme depend significantly on the memory architecture. In 3D stacked memories the overhead in IPC and energy is on average 1.2% and 2.2%, respectively, for flat-T1, and 6.4% and 14.9% for T2. Flat-T2 has higher overheads because every write access requires an extra read

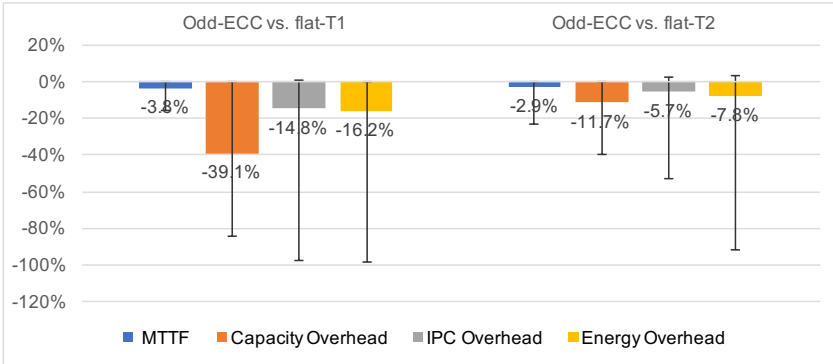


Figure 4.14: Improving capacity when using Odd-ECC in 2D DRAM DIMMs. Odd-ECC setups versus flat-T1 and flat-T2 with the same MTTF constraints. Average, minimum, and maximum increase in MTTF, and in capacity, IPC, and energy overheads (negative numbers indicate reduction).

modify and write of the respective parity. For 2D DRAMs, the performance and energy overheads are higher due to use of single channel. In 2D-DRAMs, flat-T1 introduces on average a performance and energy overhead of 7.7% and 6.7%, respectively, while flat T2 has a 23.6% overhead in performance and 47.% in energy. As shown, Odd-ECC setups with the same MTTF as a flat protection scheme are faster and more energy efficient.

4.5.3 Summary

In order to summarize the benefits of Odd-ECC we present two charts, one for each goal that can be achieved by Odd-ECC: improving the memory capacity with similar MTTF (Figures 4.14 and 4.15) and improving the MTTF with similar memory capacity overhead (Figure 4.16 and 4.17). Notice that the former can be done as compared to both flat-T1 and flat-T2 while the latter only when compared to flat-T1 since we cannot improve the MTTF over flat-T2.

In each chart we show the average results over all applications for the Odd-ECC protection setups against the corresponding flat protections. We breakdown the results into change in MTTF, capacity overhead, IPC overhead, and energy overhead. The error bars represent the extreme maximum and minimum results for particular applications.

In terms of the capacity goal, we can observe that the Odd-ECC protection setups in both 2D and 3D-stacked DRAMs are able to keep a similar MTTF (reduction by 3.8-11.1% and 2.9-7.4% as compared to flat-T1 and flat-T2, respectively) and achieve a reduction in the memory capacity overhead of 39.1% and 11.7% on average when compared to flat-T1 and flat-T2, respectively. It is important to notice that the MTTF is not significantly affected as we gain capacity and at the same time we observe a reduction in both the IPC and energy overheads.

When attempting to improve flat-T1 MTTF within the same ECC storage constraints, we can observe that Odd-ECC protection setups are able to improve considerably the

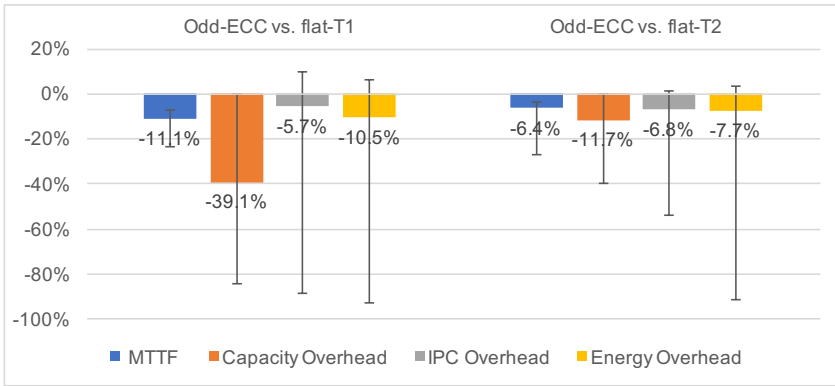


Figure 4.15: Improving capacity when using Odd-ECC in 3D-stacked DRAM. Odd-ECC setups versus flat-T1 and flat-T2 with the same MTTF constraints. Average, minimum, and maximum increase in MTTF, and in capacity, IPC, and energy overheads (negative numbers indicate reduction).

MTTF. In 2D DRAMs that is 266% on average and up to 900%, and in 3D-stacked DRAMs 770% on average and up to 1400%. Nevertheless, this MTTF improvement comes with a performance and energy cost (195% and 344%-429% higher than the flat-T1 overheads, respectively). It is relevant to notice that this cost is always lower than a flat-T2 memory protection approach and is due to the fact that for improving MTTF over flat-T1, only a subset of the data regions require T2 protection.

4.6 Related Work

In Odd-ECC we use application sensitivity analysis to gain insight about error resiliency of different data regions. In the past, several approaches have been taken based on the same observation. Mehrara and Austin [50] explored the possible benefits of providing different level of protections for different application memory regions. However, they did not propose a mechanism to support different memory protection schemes. In Flikker [59] application's data are divided into critical and non-critical regions and the authors propose a technique to trade DRAM refresh rate with reliability in order to reduce refresh energy consumption for mobile devices. Luo *et al.* [49] use application's sensitivity analysis as a motivation for heterogeneous-reliability memory systems, where DIMMs with different protection levels are connected to different memory channels.

Several works on fault tolerance for both 2D- and 3D-DRAMs have been presented in the past. We discuss the most relevant and recent ones below.

In the 2D-DRAM domain, typical chipkill-level schemes use 4-symbol SSC-DSD codes and require activation of 36×4 DRAM chips. In order to improve the energy cost, many alternative schemes have been proposed. Among them, some offer the flexibility

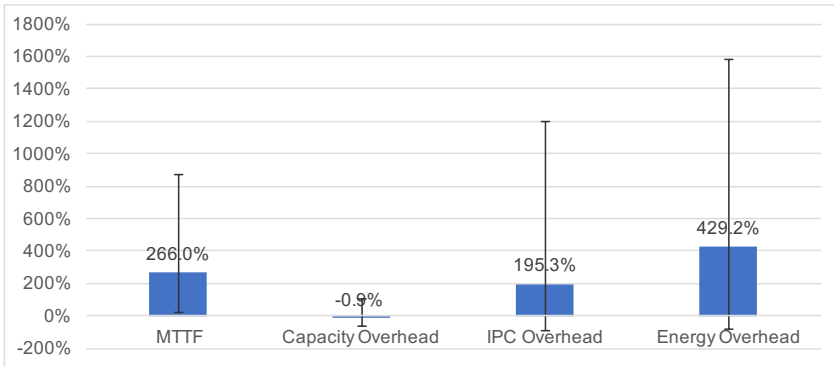


Figure 4.16: Improving MTTF when using Odd-ECC in 2D DRAM DIMMs. Odd-ECC setups versus flat-T1 and flat-T2 with the same capacity constraints. Average, minimum, and maximum increase in MTTF, and in capacity, IPC, and energy overheads (negative numbers indicate reduction).

to use different levels of protection. VECC [60] dynamically maps data and ECC information in separate locations inside the same memory. VECC can be applied to non-ECC $\times 8$ DIMMs with 18.75% overhead. However, VECC requires major modifications in the memory system, operating system and last level cache. Another restriction of VECC is that for $\times 8$ chips it requires the activation of 16 chips. Clean-ECC [61] offers fault protection flexibility by exploiting different granularity memory accesses in special sub-ranked $\times 4$ ECC-DIMMs and has a fixed capacity overhead of 12.5%. In addition to the access granularity prediction mechanism, Clean-ECC requires modification of the memory interface and cache management to handle mixed granularity accesses. MAGE [62] uses a combination of SSC-DSD and erasure codes to adaptively change the protection level based on applications access granularity. MAGE can be employed on different memory system configurations, requiring at least two 128-bit channels for $\times 8$ chips with ECC-DIMMs. ARCC [63] describes a mechanism to adaptively increase the protection strength by combining cachelines at page granularity. Bamboo ECC [64] proposes a single-tier, strong error correction code that can gracefully degrade the protection level through redundant bit steering on $\times 4$ DRAMs with ECC-DIMMs. RAIM-5 [75] is new design that allows selective protection of subset of the memory, allowing the user to pay storage cast only when needed, in multi-channel memory systems. Compared to these techniques, to the best of our knowledge, Odd-ECC is the only scheme that offers three different protection levels applied on non-ECC $\times 8$ DIMMs, requiring access to only 8 chips in a rank on a 64-bit channel.

Although the above techniques provide flexible reliability, they impose fixed capacity overhead on the system. On the contrary, in Odd-ECC capacity overheads are proportional to the protection level offered at page granularity. Another technique that aims at capacity efficiency is ECC-Parity [65], which is an optimization of chipkill-level protection

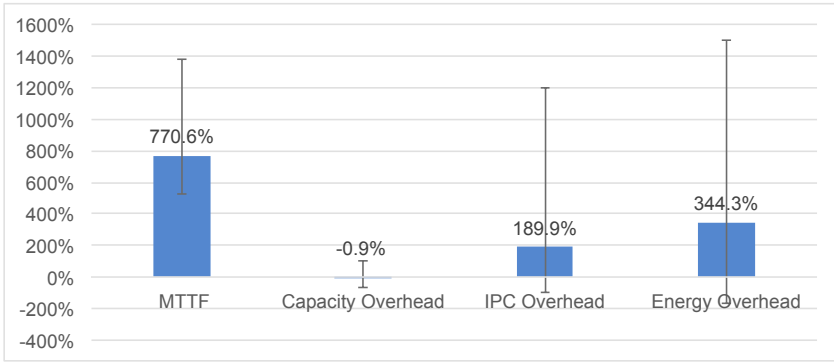


Figure 4.17: Improving MTTF when using Odd-ECC in 3D-stacked DRAMS. Odd-ECC setups versus flat-T1 and flat-T2 with the same capacity constraints. Average, minimum, and maximum increase in MTTF, and in capacity, IPC, and energy overheads (negative numbers indicate reduction).

schemes, assuming multi-channel memory systems. ECC-Parity decouples error detection and correction bits and instead of saving the actual ECC bits on each channel, the bit-wise parity of ECC is stored in a different channel to save capacity and energy. Capacity and reliability trade-off for DRAM memories has also been addressed in CREAM (Capacity- and Reliability-Adaptive Memory) [76], in which a hardware mechanism is employed to offer multiple data protection levels. CREAM enables the use the extra storage of ECC-DIMMs when no protection is required, while allowing the required address transition to be transparent to the OS by using a DRAM module bridge chip. By modeling page faults, CREAM shows significant performance improvement when extra capacity is provided for running applications.

The T1 and T2 protection levels used in Odd-ECC are related to prior-art. In particular, LOT-ECC [66] uses a two tier protection scheme where the first tier is responsible for local error detection and second tier is used for error correction. Tier one is a 7bit checksum of 57bit segments of one cacheline and tier two is the bit-wise parity of different 57bit segments of the same cacheline. In comparison, Odd-ECC T1 protects the entire 64B cacheline with single-bit fault correction and strong CRC-16 detection, and T2 is the bit-wise parity of 8B segments of seven different cachelines. Moreover, flat T2 protection in Odd-ECC has capacity overhead of 30.6% using 8 chips on a non-ECC DIMM, which is comparable to LOT-ECC 26.5% capacity overhead using 9 chips in an ECC-DIMM.

Another technique is Multi-ECC [67], which uses a first tier Reed-Solomon code with erasure correction capability and checksum error localization codes over 256 adjacent cachelines on the same column. Multi-ECC uses ECC-DIMMs and requires 256 cacheline reads for correcting multiple-bit faults.

E3CC [68] stores the ECC-bits on the columns next to the data, and proposes a new

Biased Chinese Remainder Mapping (BCRM) to resolve the non-power-of-two address mapping required for this placement. In Odd-ECC, we use the same placement for T1, however, by splitting the row into 64B boundaries, we eliminate the need for complicated address mapping.

In addition to VECC, COP [69] and Frugal-ECC [70] are two other techniques that combine data compression with protection to provide chipkill-level protection using non-ECC DIMMs.

The reliability of 3D-stacked DRAMs has been the topic of some recent works. Citadel [71] uses a combination of techniques to improve the reliability of a 3D-stacked DRAM where each die is connected to one channel[21]. A two tier ECC-scheme is used where T1 is a strong CRC-32 error detection and T2 is a three dimensional parity correction code, stored in an additional ECC-die. The error correction process in Citadel takes a long time and is not efficient for correcting transient faults. Citadel can tolerate up to complete bank faults. Another technique which is proposed for the same 3D-stacked structure, *i.e.* one channel per die, is RATT-ECC [72] which uses a strong Reed-Solomon codes as tier one, and a RAID5 like parity correction code, across channels and banks as tier 2. Another work that uses bit-wise parity for correcting large granularity faults is Parity-Helix [73], where a parity line is constructed using one cacheline from each channel, each die, similar to a spiral. Parity-Helix can protect against complete bank, channel and die faults. E-RAS [74] decouples error detecting and parity-based correcting codes, storing them is separate regions. Parities are constructed using cachelines from all channels similar to RAID5, providing protection against complete bank and channel faults. While Odd-ECC is similar to some of these technique in the way that it uses parity based correction codes, none of these techniques support different levels of DRAM protection. To the best of our knowledge, Odd-ECC is the first scheme that can provide different protection levels on 3D-stacked DRAMs, using eight dies. Except for Parity-Helix, other proposals require additional ECC-die, effectively requiring special stack fabrication. Moreover, in all described techniques reading and updating error correcting codes require accesses to multiple channels, which implies a centralized memory controller for the stack. In particular, the channel and die protection coverage of Parity-Helix depends on accessing the parity line, constructed over cachelines in different channels. However, Odd-ECC complies with current specification of HMC and HBM in the way it considers that each vault (channel) has a separate memory controller.

4.7 Conclusion

Odd-ECC offers a flexible memory protection scheme, which enables a system to dynamically select the level of fault tolerance for different application data. The Odd-ECC address mapping allows the OS to store ECC bits in separate physical pages made unavailable to the user. The same mapping guarantees that the ECC bits are physically aligned with the data they protect, so as to ensure efficient memory accesses on both

[21] This structure is supported by HBM [138].

conventional 2D DRAMs and 3D-stacked DRAMs. We analyzed the reliability of various applications and observed that faults in different data regions may have different impact on application's reliability. Odd-ECC allows such regions to be protected differently thus, offering the fault tolerance level required for the expected MTTF goal and paying a proportional cost in capacity, performance, and energy. Compared to a flat Tier-1 (T1) memory fault tolerance scheme that provides multi-bit detection and single-bit correction, Odd-ECC reduces the capacity overheads by 39% and performance and energy overheads by 6-15% to 10-16%, respectively, without compromising reliability (MTTF). Compared to a flat Tier 2 (T2) memory protection scheme that supports additional multi-bit correction, Odd-ECC reduces capacity overheads by 12% and performance and energy overheads by 6-8%. Finally, within the particular capacity constraints of a flat T1 scheme, Odd-ECC improves MTTF by 260% to 770% increasing T1 performance and energy overheads by 195% and 430%-340%, respectively, which are still lower than the overheads of a flat T2 scheme.

5 Conclusion

The design of adaptive fault-tolerant system components have been addressed in this thesis. It has been observed that previous adaptive fault tolerance techniques are mainly employed to sustain a fixed level of reliability, by properly reacting to changes in systems conditions. Besides this primary usage, in this thesis we suggested that the flexibility of adaptive fault-tolerant components can be exploited to offer better trade-off between reliability and its overheads. The effectiveness of this approach was then demonstrated by setting and pursuing different objectives for three main components of a multi-core system, i.e. micro-processors, a Network-on-Chip (NoC), and DRAM main memories. In the following sections, we briefly summarize the content of this thesis, present our main findings and contributions in each part and, finally, provide a guideline for possible future related research topics.

5.1 Summary

In the first part of this thesis, covered in Chapter 2, we studied the design of a reliable micro-processors architecture that can adapt to permanent faults through coarse and/or fine grain reconfigurations. Our main objective was to find the most efficient granularity of substitutable units (SUs) so that, for a given fault-rate, we have the best trade-off between the achieved reliability and incurred overheads, i.e. in terms of area, performance and power. To this end, first a probabilistic reliability analysis of a generic reconfigurable design was presented and different SU granularities were evaluated in terms of reliability and their overheads. Subsequently, we used a CMP with multiple identical processors as our use-case, and explored the reliability and overheads of our proposed adaptive

fault-tolerant architecture considering different reconfiguration options. Furthermore, we explored the advantages of pipelining reconfigurable interconnects by measuring the execution time and energy consumption and comparing it to the design with non-pipelined wires. Finally, in order to find the most efficient granularity mix, a design space exploration was performed.

In the second part of this thesis, covered in Chapter 3, the main objective was to design and evaluate a service-oriented NoC that enables us to trade service-isolation for reliability. To this end, we described RQNoC, a service-oriented Network-on-Chip (NoC) that can adapt to permanent faults. We characterized the network resources based on the particular service they support and designed a mechanism to bypass them in presence of permanent faults, allowing the respective traffic class to be redirected. We proposed two alternatives for service redirection, each having different advantages and disadvantages. The first one, Service Detour, used longer alternative paths through resources of the same service to bypass faulty network parts, keeping traffic classes isolated. The second approach, Service Merge, used resources of other services providing shorter paths but allowing traffic classes to interfere with each other. The remaining network resources that are common for all services employed additional mechanisms for tolerating faults. Links tolerated faults using additional spare wires combined with a flit-shifting mechanism and the router control was protected with Triple-Modular-Redundancy (TMR). We evaluated these two alternative techniques in terms of reliability as well as hardware and performance overheads.

In the third and final part of this thesis, covered in Chapter 4, we focused of main memory DRAMs and the objective was to have a fault tolerance mechanism that allows us to trade capacity for reliability. To this end, we introduced Odd-ECC: a new DRAM data mapping scheme that offers adaptive ECC for each physical page of an application's data inside DRAM memories. The main motivation was the observation that an application may have different sensitivity to faults that appear in different parts of its data. This observation suggests that using a fixed protection scheme for all data regions could potentially be a case of reliability over-provisioning. In Odd-ECC, we capitalized on this observation and provided a mechanism to dynamically select the memory protection level of each allocated page of a program on demand depending on the criticality of the respective data. Briefly, in Odd-ECC 4KB pages of physical memory are grouped in pools of 64 pages, where each pool can have a particular protection level. Then, depending on the protection offered, a number of pages in a pool is reserved, i.e. marked unavailable for the user by the OS, to store the ECCs. In our study we selected an Odd-ECC configuration with three different protection levels, namely *Tier Zero* (T0), *Tier One* (T1), and *Tier Two* (T2), where T0 provides no protection, T1 provides lower level protection (single-bit correction, multi-bit detection), and T2 provides higher level protection (adding multi-bit correction to T1). We showed how Odd-ECC can be applied to conventional 2D as well as to 3D-stacked DRAMs and evaluated the reliability, capacity, performance and energy consumption of several Odd-ECC setups, comparing them with flat protection cases.

The following section summarizes the contributions of this thesis for the three aforementioned parts.

5.2 Contributions

This thesis addressed several issues regarding the design of adaptive fault-tolerant system components and made the following contributions and findings.

5.2.1 Adaptive Fault-Tolerant Micro-Processors

The first part of the thesis dealt with design of a reliable micro-processors architecture that can adapt to permanent faults. The main contributions of this part of the thesis were the following:

- **Analytically calculating the fault tolerance for a generic array of reconfigurable components with substitutable units.**

  We presented an analytical method for calculating the probability of constructing a certain number of fault-free components via different reconfiguration options, i.e. CG and CG+FG, given the fault density.

- **Evaluating the area overheads of different reconfiguration options.**

  We measured the area overheads of reconfigurability using reconfigurable micro-processors with substitutable parts as a use-case component.

- **Evaluating the performance impact of pipelining reconfigurable interconnects in an adaptive fault-tolerant micro-processors array at different fault densities.**

  As a part of our work, we investigated possible performance benefits of pipelining reconfigurable interconnects which connect stages of adaptive micro-processors. We retrieved post place and route results in order to consider the actual delay of the reconfigurable interconnects.

- **Performing a Design Space exploration for finding the most efficient reconfiguration mix of an adaptive micro-processors array.**

  Finally, using the obtained analytical calculations and overhead values from our use-case, we evaluated various reconfiguration scenarios, measuring the average number of fault-free components as well as the probability of delivering a minimum number of fault-free components in a given silicon area. In so doing, we identified the most fault-tolerant designs for a particular fault density.

Our results showed that employing fine-grain logic in addition to the coarse-grain reconfiguration, i.e., mixed-grain, can increase availability, tolerating $3\times$ more faults than mere component redundancy. Furthermore, the design-space exploration revealed that different fault densities require different granularities of substitutable units to maximize fault tolerance. With growing number of permanent faults[1], coarse-grain with granularity of $\frac{1}{8}$ and mixed-grain with granularity of $\frac{1}{16}$ offered the best availability figures. Finally,

[1] In a fixed silicon area with known number of transistors.

we observed that in adaptive processors with no pipelined interconnects the frequency and execution time both increased by almost 3.5×, compared to the baseline. Adding pipeline registers showed significantly better performance, where for the fault-free case the overhead of operating frequency and execution time were 1.41× and 1.7× compared to the baseline.

5.2.2 Adaptive Fault-Tolerant NoC

In the second part of the thesis, our focus was the design of a reliable service-oriented NoC architecture. The main contributions of this part were as follows.

- **Modifying the architecture of a service-oriented NoC to provide service level reconfiguration for tolerating permanent faults.**

  The architecture of the baseline service-oriented NoC is modified to support different service redirection alternatives, i.e. Service Detour and Service Merge.

  - *Service Detour:* We extended previous detour techniques, allowing the network to selectively detour traffic per service rather than detouring all packets to avoid faulty network parts, substantially improving NoC reliability as well as significantly increasing performance.
  - *Service Merge:* Within a router, multiple services are merged, still preserving their priorities, in order to bypass the damaged data-path of one or multiple services.

- **Evaluating the proposed fault tolerance techniques in terms of performance, power and area.**

  The above fault tolerance techniques are evaluated in terms of network performance, i.e. latency (in cycles), throughput (flits/cycle/node) and also the percentage of successful packet transmission. Moreover, area, power and frequency overheads are reported after implementing the proposed techniques.

- **Evaluating reliability improvements.**

  Analytical models of our techniques are created and network connectivity is measured under different fault densities.

The proposed RQNoC network designs were implemented in 65nm technology and evaluated in terms of performance, area, power consumption and fault tolerance. Service Detour required 25% more area and consumed 7% more power compared to a baseline network, not tolerant to faults; its packet latency and throughput were close to the fault free performance at low fault densities, but fault tolerance and performance dropped substantially above 4 network faults. Service Merge required 51% more area and 27% more power than the baseline and had a 9% slower clock. Compared to a fault free network, a Service Merge RQNoC with up to 32 faults had increased packet latency up

to 1.5-2.4× and reduced throughput to 70% or 50%. However, it delivered substantially better fault tolerance having a mean network connectivity of above 90% even with 32 network faults versus 41% of a Service Detour network. Combining Service Merge and Service Detour improved fault tolerance further sustaining a higher number of network faults and reduced packet latency.

5.2.3 Adaptive Fault-tolerant Main Memories

The third part of the thesis was dedicated to design of a mechanism that supports adaptive ECCs within DRAM main memories. Concisely, the main contributions were as follows.

- **Analyzing applications sensitivity to faults.**

  We extensively studied a variety of applications, analyzing the impact of faults on their different data regions to their reliability (MTTF). Moreover, we defined a methodology to explore the impact of protecting data regions with ECCs of different strength, on the applications reliability.

- **Proposing a new scheme for adaptive ECCs in DRAM modules.**

  We presented Odd-ECC, a new memory mapping scheme able to support different levels of fault tolerance for data stored in memory, decided dynamically on demand for each allocated physical page.

- **Applying Odd-ECC to 2D, as well as 3D-stacked DRAM modules.** We showed how Odd-ECC is applied to both conventional 2D DRAM DIMMs as well as to 3D-stacked DRAMs.

- **Evaluating the reliability and performance impact of different Odd-ECC setups.**

  We evaluated Odd-ECC using the analyzed applications and compare the achieved MTTF, the ECC capacity, performance and energy overheads versus flat memory protection schemes.

  - Odd-ECC can be used to reduce capacity overheads (besides slightly reducing performance and energy costs) maintaining similar reliability to a flat protection scheme.
  - Alternatively, Odd-ECC can significantly improve reliability under the same capacity constraints as a flat protection scheme at the cost of some performance and energy overheads.

We showed that compared to a flat T1 memory protection scheme Odd-ECC reduces the capacity overheads by 39% and performance and energy overheads by 6-15% to 10-16%, respectively, without compromising reliability of the application(MTTF). Compared to a flat T2 memory protection Odd-ECC reduces capacity overheads by 12% and performance and energy overheads by 6-8%. Finally, within the particular capacity

constraints of a flat T1 scheme, Odd-ECC improves MTTF by 260% to 770% increasing T1 performance and energy overheads by 195% and 430%-340%, respectively, which are still lower than the overheads of a flat T2 scheme.

5.3 Proposed Research directions

In this thesis we showed how adaptive fault tolerance techniques can be exploited to have better trade-off between reliability and its overheads. There are several directions for future research that can improve and complement the work presented here. In the following, we identify and list some of them:

Adaptive Fault-Tolerant Micro-Processors: It would be interesting to explore extending the proposed techniques for more complex (for example, out-of-order) microarchitectures. Moreover, run-time management of mixed-grain reconfigurable designs that can react based on detected faults and the acceptable performance degradation is another interesting topic. Analyzing the timing dependability, i.e. the response time of the system from the fault detection point until system restoration, remains an open issue which requires design of low-overhead online-testing mechanisms to accurately detect, locate and isolate permanent faults. Finally, the probabilistic analysis can be further extended to consider different distributions of faults other than uniform, e.g. cluster faults.

Adaptive Fault-Tolerant NoC: Exploring different policies for merging/detouring services based on performance requirements would be an interesting topic. The techniques to improve the reliability inside the service-oriented NoC can also be exploited for energy efficiency purposes. For example, in the fault free mode, RQNoC can be configured to disable some service lanes and use the service merge feature to work in a degraded mode, in a manner similar to adaptive bandwidth networks [142], offering significant reduction in power consumption. Moreover, adaptive routing algorithms can be used to increase the packet delivery in networks with irregular topologies, where some nodes are disabled.

Adaptive Fault-tolerant Main Memories: The proposed Odd-ECC schemes can be extended to support additional protection levels. In the current form, Odd-ECC is applicable to non-ECC $\times 8$ DRAM DIMMs. However, with some modifications it should be possible to extend its usage to other modules such as DIMMs with $\times 4$ devices or ECC-DIMMs with 9 chips and 72B channel. Another interesting topic could be to explore the possibility of having multiple protection levels inside the same pool of pages, thus, avoiding the need to have separate pools per protection level. In the context of 3D-stacked memories, one interesting research direction could be exploring fault tolerance techniques for the interconnects and the memory combined.

These research directions are just a few of many possible ways. Of course, each of these tracks requires more inspection and deeper investigation.

Bibliography

[1] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. Bulletproof: A defect-tolerant cmp switch architecture. In *In Proceedings of the 12th International Symposium on High Performance Computer Architecture*, pages 3–14, 2006.

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

[3] Peter A Lee and Thomas Anderson. *Fault tolerance: principles and practice*, volume 3. Springer Science & Business Media, 2012.

[4] Matti A Hiltunen and Richard D Schlichting. A model for adaptive fault-tolerant systems. In *European Dependable Computing Conference*, pages 1–20. Springer, 1994.

[5] Eltefaat Shokri, Herbert Hecht, Patrick Crane, Jerry Dussault, and KH Kim. An approach for adaptive fault tolerance in object-oriented open distributed systems. *International Journal of Software Engineering and Knowledge Engineering*, 8(03):333–346, 1998.

[6] Myron Hecht, Herbert Hecht, and Eltefaat Shokri. Adaptive fault tolerance for spacecraft. In *Aerospace Conference Proceedings, 2000 IEEE*, volume 5, pages 521–533. IEEE, 2000.

[7] KH Kane Kim and Thomas F Lawrence. Adaptive fault-tolerance in complex real-time distributed computer system applications. *Computer Communications*, 15(4):243–251, 1992.

[8] Jack Goldberg, Ira Greenberg, Raymond Clark, ED Jensen, and Kane Kim. Adaptive fault-resistant systems. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1994.

[9] Kypros Constantinides, Onur Mutlu, and Todd Austin. Online design bug detection: Rtl analysis, flexible mechanisms, and evaluation. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 282–293. IEEE, 2008.

[10] Kypros Constantinides, Onur Mutlu, Todd Austin, and Valeria Bertacco. Software-based online detection of hardware defects mechanisms, architectural support, and evaluation. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 97–108. IEEE Computer Society, 2007.

[11] Kypros Constantinides, Onur Mutlu, Todd Austin, and Valeria Bertacco. A flexible software-based framework for online detection of hardware defects. *IEEE Transactions on Computers*, 58(8):1063–1079, 2009.

[12] Yanjing Li, Samy Makar, and Subhasish Mitra. Casp: Concurrent autonomous chip self-test using stored test patterns. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 885–890. IEEE, 2008.

[13] Yanjing Li, Onur Mutlu, Donald S Gardner, and Subhasish Mitra. Concurrent autonomous self-test for uncore components in system-on-chips. In *VLSI Test Symposium (VTS), 2010 28th*, pages 232–237. IEEE, 2010.

[14] Y Li, E Cheng, S Makar, and S Mita. Self-repair of uncore components in robust systems-on-chips. *SELSE*, 2013.

[15] M.L. Fair et al. Reliability, availability, and serviceability (ras) of the ibm eserver z990. *IBM Jour. of Research & Development*, 48(3-4):519–534, 2004.

[16] David Bernick, Bill Bruckert, Paul Del Vigna, David Garcia, Robert Jardine, Jim Klecka, and Jim Smullen. Nonstop advanced architecture. In *DSN*, 2005.

[17] C. LaFrieda et al. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *DSN*, pages 317–326, 2007.

[18] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. Resilient high-performance processors with spare ribs. *IEEE Micro*, 33(4):26–34, 2013.

[19] Smitha Shyam, Kypros Constantinides, Sujay Phadke, Valeria Bertacco, and Todd Austin. Ultra low-cost defect protection for microprocessor pipelines. In *ASPLOS XII*, pages 73–82, 2006.

[20] Bogdan F. Romanescu and Daniel J. Sorin. Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults. In *PACT*, pages 43–51, 2008.

[21] S. Gupta, Shuguang Feng, A. Ansari, and S. Mahlke. Stagenet: A reconfigurable fabric for constructing dependable cmps. *IEEE Trans. on Computers*, 60(1):5–19, 2011.

[22] Andrea Pellegrini, Joseph L. Greathouse, and Valeria Bertacco. Viper: virtual pipelines for enhanced reliability. In *ISCA '12*, pages 344–355, 2012.

[23] I. Wagner, V. Bertacco, and T. Austin. Shielding against design flaws with field repairable control logic. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 344–347, 2006.

[24] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *Design Test of Computers, IEEE*, 22(4):306–315, 2005.

[25] Wei-Je Huang and E.J. McCluskey. Column-based precompiled configuration techniques for fpga. In *FCCM*, pages 137–146, 2001.

[26] A. P. Shanthi and R. Parthasarathi. Exploring fpga structures for evolving fault tolerant hardware. In *NASA/DoD Conference on Evolvable Hardware, 2003. Proceedings.*, pages 174–181, July 2003.

[27] Mingjie Lin, S. Ferguson, Yaling Ma, and T. Greene. Haft: A hybrid fpga with amorphous and fault-tolerant architecture. In *2008 IEEE International Symposium on Circuits and Systems*, pages 1348–1351, May 2008.

[28] AJ Yu and Guy G Lemieux. Fpga defect tolerance: Impact of granularity. In *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, pages 189–196. IEEE, 2005.

[29] Atin Mukherjee and Anindya Sundar Dhar. Choice of granularity for reliable circuit design using dynamic reconfiguration. *Microelectronics Reliability*, 63(Complete):291–303, 2016.

[30] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5):15–31, 2007.

[31] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc: Qos architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2):105–128, 2004.

[32] Srinivasan Murali, David Atienza, Luca Benini, and Giovanni De Michel. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of the 43rd annual Design Automation Conference*, pages 845–848. ACM, 2006.

[33] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Minxuan Zhang, and Zuocheng Xing. Addressing transient and permanent faults in noc with efficient fault-tolerant deflection router. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(6):1053–1066, 2013.

[34] Muhammad Ali, Michael Welzl, and Sven Hessler. A fault tolerant mechanism for handling permanent and transient failures in a network on chip. In *Information Technology, 2007. ITNG'07. Fourth International Conference on*, pages 1027–1032. IEEE, 2007.

[35] Ritesh Parikh and Valeria Bertacco. udirec: unified diagnosis and reconfiguration for frugal bypass of noc faults. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 148–159. ACM, 2013.

[36] Mohammad Fattah, Antti Airola, Rachata Ausavarungnirun, Nima Mirzaei, Pasi Liljeberg, Juha Plosila, Siamak Mohammadi, Tapio Pahikkala, Onur Mutlu, and Hannu Tenhunen. A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 18. ACM, 2015.

[37] Alessandro Strano, Davide Bertozzi, Francisco Trivino, José L Sánchez, Francisco J Alfaro, and José Flich. Osr-lite: Fast and deadlock-free noc reconfiguration framework. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 86–95. IEEE, 2012.

[38] Konstantinos Aisopos, Andrew DeOrio, Li-Shiuan Peh, and Valeria Bertacco. Ariadne: Agnostic reconfiguration in a disconnected network environment. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 298–309. IEEE, 2011.

[39] Samuel Rodrigo, Jose Flich, Antoni Roca, Simone Medardoni, Davide Bertozzi, J Camacho, Federico Silla, and Jose Duato. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 25–32. IEEE, 2010.

[40] David Fick, Andrew DeOrio, Gregory Chen, Valeria Bertacco, Dennis Sylvester, and David Blaauw. A highly resilient routing algorithm for fault-tolerant nocs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 21–26. European Design and Automation Association, 2009.

[41] Yinhe Han and Binzhang Fu. A new fault-tolerant routing based on turn model. In *Proceedings of the 3rd Workshop on Diagnosttic Services in Network-on-Chips, DSNOC'09*, pages 102–103. IEEE Computer Society, 2009.

[42] Arseniy Vitkovskiy, Vassos Soteriou, and Chrysostomos Nicopoulos. Dynamic fault-tolerant routing algorithm for networks-on-chip based on localised detouring paths. *Computers & Digital Techniques, IET*, 7(2), 2013.

[43] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, and Timothy Mark Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *Proceedings of the second ACM/IEEE international symposium on networks-on-chip*, pages 13–22. IEEE Computer Society, 2008.

[44] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester. Vicis: a reliable network for unreliable silicon. In *Proceedings of the 46th Annual Design Automation Conference*, pages 812–817. ACM, 2009.

[45] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Mazin S Yousif, and Chita R Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 4–15. IEEE Computer Society, 2006.

[46] Marios Evripidou, Chrysostomos Nicopoulos, Vassos Soteriou, and Jongman Kim. Virtualizing virtual channels for increased network-on-chip robustness and upgradeability. In *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pages 21–26. IEEE, 2012.

[47] Mohammad Reza Kakoee, Valeria Bertacco, and Luca Benini. Relinoc: A reliable network for priority-based on-chip communication. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[48] Cheng Liu, Lei Zhang, Yinhe Han, and Xiaowei Li. A resilient on-chip router design through data path salvaging. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 437–442. IEEE Press, 2011.

[49] Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 467–478. IEEE, 2014.

[50] Mojtaba Mehrara and Todd Austin. Exploiting selective placement for low-cost memory protection. *ACM Transactions on Architecture and Code Optimization (TACO)*, 5(3):14, 2008.

[51] Vilas Sridharan and David R Kaeli. Using pvf traces to accelerate avf modeling. In *Proceedings of the IEEE workshop on silicon errors in logic-system effects*, pages 23–24, 2010.

[52] Vilas Sridharan and David R Kaeli. Eliminating microarchitectural dependency from architectural vulnerability. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 117–128. IEEE, 2009.

[53] Siva Kumar Sastry Hari, Sarita V Adve, Helia Naeimi, and Pradeep Ramachandran. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In *ACM SIGPLAN Notices*, volume 47, pages 123–134. ACM, 2012.

[54] Shubhendu S Mukherjee, Christopher Weaver, Joel Emer, Steven K Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 29–40. IEEE, 2003.

[55] Dong Li, Jeffrey S Vetter, and Weikuan Yu. Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 57. IEEE Computer Society Press, 2012.

[56] Li Yu, Dong Li, Sparsh Mittal, and Jeffrey S Vetter. Quantitatively modeling application resilience with the data vulnerability factor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 695–706. IEEE Press, 2014.

[57] Xuanhua Li and Donald Yeung. Application-level correctness and its impact on fault tolerance. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 181–192. IEEE, 2007.

[58] Dong Li, Zizhong Chen, Panruo Wu, and Jeffrey S Vetter. Rethinking algorithm-based fault tolerance with a cooperative software-hardware approach. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2013.

[59] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. Flikker: saving dram refresh-power through critical data partitioning. *ACM SIGPLAN Notices*, 47(4):213–224, 2012.

[60] Doe Hyun Yoon and Mattan Erez. Virtualized and flexible ecc for main memory. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 397–408. ACM, 2010.

[61] Seong-Lyong Gong, Minsoo Rhu, Jungrae Kim, Jinsuk Chung, and Mattan Erez. Clean-ecc: High reliability ecc for adaptive granularity memory system. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 611–622. ACM, 2015.

[62] Sheng Li, Doe Hyun Yoon, Ke Chen, Jishen Zhao, Jung Ho Ahn, Jay B Brockman, Yuan Xie, and Norman P Jouppi. Mage: adaptive granularity and ecc for resilient and power efficient memory systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11. IEEE, 2012.

[63] Xun Jian and Rakesh Kumar. Adaptive reliability chipkill correct (arcc). In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 270–281. IEEE, 2013.

[64] Jungrae Kim, Michael Sullivan, and Mattan Erez. Bamboo ecc: Strong, safe, and flexible codes for reliable computer memory. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 101–112. IEEE, 2015.

[65] Xun Jian and Rakesh Kumar. Ecc parity: A technique for efficient memory error resilience for multi-channel memory systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1035–1046. IEEE Press, 2014.

[66] Aniruddha N Udipi, Naveen Muralimanohar, Rajeev Balsubramonian, Al Davis, and Norman P Jouppi. Lot-ecc: localized and tiered reliability mechanisms for commodity memory systems. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 285–296. IEEE Computer Society, 2012.

[67] Xun Jian, Henry Duwe, John Sartori, Vilas Sridharan, and Rakesh Kumar. Low-power, low-storage-overhead chipkill correct via multi-line error correction. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 24. ACM, 2013.

[68] Long Chen, Yanan Cao, and Zhao Zhang. E3cc: A memory error protection scheme with novel address mapping for subranked and low-power memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):32, 2013.

[69] David J Palframan, Nam Sung Kim, and Mikko H Lipasti. Cop: To compress and protect main memory. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 682–693. ACM, 2015.

[70] Jungrae Kim, Michael Sullivan, Seong-Lyong Gong, and Mattan Erez. Frugal ecc: Efficient and versatile memory error protection through fine-grained compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 12. ACM, 2015.

[71] Prashant J Nair, David A Roberts, and Moinuddin K Qureshi. Citadel: Efficiently protecting stacked memory from tsv and large granularity failures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):49, 2016.

[72] Hsing-Min Chen, Carole-Jean Wu, Trevor Mudge, and Chaitali Chakrabarti. Ratt-ecc: Rate adaptive two-tiered error correction codes for reliable 3d die-stacked memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(3):24, 2016.

[73] Xun Jian, Vilas Sridharan, and Rakesh Kumar. Parity helix: Efficient protection for single-dimensional faults in multi-dimensional memory systems. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pages 555–567. IEEE, 2016.

[74] Hyeran Jeon, Gabriel H Loh, and Murali Annavaram. Efficient ras support for die-stacked dram. In *Test Conference (ITC), 2014 IEEE International*, pages 1–10. IEEE, 2014.

[75] Ruohuang Zheng and Michael C Huang. Redundant memory array architecture for efficient selective protection. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 214–227. ACM, 2017.

[76] Yixin Luo, Saugata Ghose, Tianshi Li, Sriram Govindan, Bikash Sharma, Bryan Kelly, Amirali Boroumand, and Onur Mutlu. Using ecc dram to adaptively increase memory capacity. *arXiv preprint arXiv:1706.08870*, 2017.

[77] Nidhi Aggarwal et al. Configurable isolation: building high availability systems with commodity multi-core processors. In *ISCA*, 2007.

[78] S. Shamshiri and Kwang-Ting Cheng. Modeling yield, cost, and quality of a spare-enhanced multicore chip. *IEEE Trans. on Computers*, 60(9):1246–1259, 2011.

[79] Nikos Hardavellas et al. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[80] Michael D. Powell, Arijit Biswas, Shantanu Gupta, and Shubhendu S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *ISCA*, pages 93–104, 2009.

[81] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design Test of Computers, IEEE*, 23(6):484–490, 2006.

[82] Georgios Smaragdos, Danish Anis Khan, Ioannis Sourdis, Christos Strydis, Alirad Malek, and Stavros Tzilis. A dependable coarse-grain reconfigurable multicore array. In *21st Reconfigurable Architectures Workshop (RAW'14)*, 2014.

[83] Ioannis Sourdis, Danish Anis Khan, Alirad Malek, Stavros Tzilis, Georgios Smaragdos, and Christos Strydis. Resilient chip multiprocessors with mixed-grained reconfigurability. *IEEE Micro*, 36(1):35–45, 2016.

[84] Scott Hauck and Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation*. Morgan Kaufmann, 2010.

[85] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Computers*, 20(12):1469–1479, 1971.

[86] EEMBC. http://www.eembc.org, 2009.

[87] Neil HE Weste and Kamran Eshraghian. Principles of cmos vlsi design: a systems perspective. *NASA STI/Recon Technical Report A*, 85:47028, 1985.

[88] Cristian Grecu, Andre Ivanov, Res Saleh, Egor S Sogomonyan, and Partha Pratim Pande. On-line fault detection and location for noc interconnects. In *On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International*, pages 6–pp. IEEE, 2006.

[89] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Computing Surveys (CSUR)*, 46(1):8, 2013.

[90] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. Online reconfigurable self-timed links for fault tolerant noc. *VLSI design*, 2007, 2007.

[91] Antonis Psathakis, Vassilis Papaefstathiou, Nikolaos Chrysos, Fabien Chaix, Evangelos Vasilakis, Dionisios Pnevmatikatos, and Manolis Katevenis. A systematic evaluation of emerging mesh-like cmp nocs. In *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, pages 159–170. IEEE, 2015.

[92] John D. Owens, William J. Dally, Ron Ho, D. N. (Jay) Jayasimha, Stephen W. Keckler, and Li-Shiuan Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, September 2007.

[93] F Gilabert, María Engracia Gómez, Simone Medardoni, and Davide Bertozzi. Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 165–172. IEEE Computer Society, 2010.

[94] DeSyRe Project official website : http://www.desyre.eu/.

[95] Adán Kohler, Gert Schley, and Martin Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(6):883–896, 2010.

[96] Ronald L Rivest and Charles E Leiserson. *Introduction to algorithms*. McGraw-Hill, Inc., 1990.

[97] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428. European Design and Automation Association, 2009.

[98] Li-Shiuan Peh and Natalie Enright Jerger. *On-Chip Networks*. Morgan and Claypool Publishers, 1st edition, 2009.

[99] Stavros Tzilis and Ioannis Sourdis. A runtime manager for gracefully degrading socs. In *in Int'l Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014.

[100] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang. Fon: Fault-on-neighbor aware routing algorithm for networks-on-chip. In *SOC Conference (SOCC), 2010 IEEE International*, pages 441–446. IEEE, 2010.

[101] Andrew DeOrio, David Fick, Valeria Bertacco, Dennis Sylvester, David Blaauw, Jin Hu, and Gregory Chen. A reliable routing architecture and algorithm for nocs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(5):726–739, 2012.

[102] Sheng Li, Ke Chen, Ming-Yu Hsieh, Naveen Muralimanohar, Chad D Kersey, Jay B Brockman, Arun F Rodrigues, and Norman P Jouppi. System implications of memory reliability in exascale computing. In *Int. Conf. for HPC, Netw., Stor. & Analysis*, page 46, 2011.

[103] Vilas Sridharan and Dean Liberty. A study of dram failures in the field. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11. IEEE, 2012.

[104] Vilas Sridharan, Jon Stearley, Nathan DeBardeleben, Sean Blanchard, and Sudhanva Gurumurthi. Feng shui of supercomputer memory positional effects in dram and sram faults. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–11. IEEE, 2013.

[105] Bharan Giridhar, Michael Cieslak, Deepankar Duggal, Ronald Dreslinski, Hsing Min Chen, Robert Patti, Betina Hold, Chaitali Chakrabarti, Trevor Mudge, and David Blaauw. Exploring dram organizations for energy-efficient and resilient exascale memories. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 23. ACM, 2013.

[106] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 415–426. IEEE, 2015.

[107] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. In *ACM SIGPLAN Notices*, volume 50, pages 297–310. ACM, 2015.

[108] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014.

[109] Sai Ankireddi and Tony Chen. Challenges in thermal management of memory modules. *Electronics Cooling*, 14(1):24, 2008.

[110] Daniel A. Reed and Jack Dongarra. Exascale computing and big data. *Commun. ACM*, 58(7):56–68, June 2015.

[111] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 267–278. ACM, 2009.

[112] Dingyou Zhang and James J.-Q. Lu. *3D Integration Technologies: An Overview*, pages 1–26. Springer International Publishing, Cham, 2017.

[113] Hsing-Min Chen, Akhil Arunkumar, Carole-Jean Wu, Trevor Mudge, and Chaitali Chakrabarti. E-ecc: Low power erasure and error correction schemes for increasing reliability of commodity dram systems. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 60–70. ACM, 2015.

[114] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

[115] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design and Test*, 34(2):60–68, 2017.

[116] IBM Support. Linux Native Memory issues for WebSphere Application Server. `http://www-01.ibm.com/support/docview.wss?uid=swg27039764&aid=1`, 2013. Online; Accessed: 2017-05-06.

[117] The OpenMP Organization. OpenMP Application Programming Interface Version 4.5. `http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`, 2015. Online; Accessed: 2017-05-05.

[118] Linux Foundation Events. Virtual Memory and Linux. `http://events.linuxfoundation.org/sites/events/files/slides/elc_2016_mem_0.pdf`, 2016. Online; Accessed: 2017-05-07.

[119] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200. ACM, 2005.

[120] The GNU Project Debugger. GDB 7.12. `https://www.gnu.org/s/gdb/`, 2016. Online; Accessed: 2017-05-07.

[121] Julian Seward, Nicholas Nethercote, and Josef Weidendorfer. *Valgrind 3.3-advanced debugging and profiling for gnu/linux applications*. Network Theory Ltd., 2008.

[122] Shubhendu S Mukherjee, Joel Emer, and Steven K Reinhardt. The soft error problem: An architectural perspective. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 243–247. IEEE, 2005.

[123] Swarup Kumar Sahoo, Man-Lap Li, Pradeep Ramachandran, Sarita V Adve, Vikram S Adve, and Yuanyuan Zhou. Using likely program invariants to detect hardware errors. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 70–79. IEEE, 2008.

[124] Georgios Stefanakis. Characterizing and exploiting application behavior under data corruption. 2015.

[125] Altera Corporation. Error Correction Code in SoC FPGA-Based Memory Systems. `https://www.altera.com/en_US/pdfs/literature/wp/wp-01179-ecc-embedded.pdf`, 2012. Online; Accessed: 2017-04-28.

[126] Jaewoong Sim, Gabriel H Loh, Vilas Sridharan, and Mike O'Connor. Resilient die-stacked dram caches. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 416–427. ACM, 2013.

[127] Marc Casas et al. Fault resilience of the algebraic multi-grid solver. In *ACM Int. Conf. on Supercomputing*, pages 91–100, 2012.

[128] Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, and Zhichun Zhu. Mini-rank: Adaptive dram architecture for improving memory power efficiency. In

Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, pages 210–221. IEEE Computer Society, 2008.

[129] Moinuddin K Qureshi and Gabe H Loh. Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 235–246. IEEE Computer Society, 2012.

[130] Mark W Kellogg, Timothy J Dell, Erik L Hedberg, and Claude L Bertin. Programmable burst length dram, April 20 1999. US Patent 5,896,404.

[131] Hybrid Memory Cube Consortium. HMC Specification 2.1. `http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf`, 2015. Online; Accessed: 2017-05-08.

[132] Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W Lee. A fully associative, tagless dram cache. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 211–222. ACM, 2015.

[133] David A Patterson, Garth Gibson, and Randy H Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[134] Bruce Jacob, Spencer Ng, and David Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[135] Abdallah M Saleh, Juan J Serrano, and Janak H Patel. Reliability of scrubbing recovery-techniques for memory systems. *IEEE transactions on reliability*, 39(1):114–122, 1990.

[136] Andy A Hwang, Ioan A Stefanovici, and Bianca Schroeder. Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design. In *ACM SIGPLAN Notices*, volume 47, pages 111–122. ACM, 2012.

[137] J Thomas Pawlowski. Hybrid memory cube (hmc). In *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pages 1–24. IEEE, 2011.

[138] JEDEC Standard. High bandwidth memory (hbm) dram. *JESD235*, 2013.

[139] Paul Rosenfeld. *Performance exploration of the hybrid memory cube*. PhD thesis, 2014.

[140] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.

[141] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14. IEEE Computer Society, 2007.

[142] George Michelogiannakis and John Shalf. Variable-width datapath for on-chip network static power reduction. In *Networks-on-Chip (NoCS), 2014 Eighth IEEE/ACM International Symposium on*, pages 96–103. IEEE, 2014.