NAIST-IS-DD1261027

Doctoral Dissertation

Techniques to Reduce the Overhead and to Improve the Robustness in a Fault Tolerable Reconfigurable Architecture

Tanvir Ahmed

March 12, 2014

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Tanvir Ahmed

Thesis Committee:

Professor Yasuhiko Nakashima	(Supervisor)
Professor Michiko Inoue	(Co-supervisor)
Associate Professor Jun Yao	(Co-supervisor)
Associate Professor Yuko Hara-Azumi	(Co-supervisor)

Techniques to Reduce the Overhead and to Improve the Robustness in a Fault Tolerable Reconfigurable Architecture*

Tanvir Ahmed

Abstract

Nowadays, fault tolerance has been playing a progressively important role in covering increasing soft/hard error rates in electronic devices that accompany the advances of process technologies. Research shows that wear-out faults have a gradual onset, starting with a temporal/transient fault and then eventually leading to a permanent fault. Error detection is thus a required function to maintain execution correctness. Currently, however, many highly dependable methods to cover permanent faults are commonly over-designed by using very frequent checking, due to lack of awareness of the fault possibility in circuits used for the pending executions.

In this dissertation, to address those issues, a technique has been proposed to add check instructions selectively on the data-path, where a metric has been introduced for permanent defects, as operation defective probability (ODP), to quantitatively instruct the check operations being placed only at critical positions. By using this selective checking approach, I can achieve a near-100% dependability by having about 53% less check operations, as compared to the ideal reliable method, which performs exhaustive checks to guarantee a zero-error propagation. By this means, I am able to reduce 21.7% power consumption by avoiding the non-critical checking inside the over-designed approach.

Further, by additionally taking the data importance into account, extra energy savings is possible from the current over-designed fault tolerable system. Partial

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1261027, March 12, 2014.

redundancy is a well used method to cover single event effects (SEEs) on critical data while leaving less important data unprotected. Under a low SEE rate, the method can provide a good cost-effective fault tolerance, while many silent data corruptions (SDCs) may occur under a high fault rate due to incomplete fault coverage.

Thus, a system-level approach is proposed to additionally cover SDCs in a partial redundancy by a light-weighted error prediction. Simulation results under a stress radiation test condition show that with an average 8% cost in energy consumption, which can reduce the SDC rate from 12% to 0.37%, for the work loads those have been studied.

Keywords:

Fault-tolerance, Reconfigurable architecture, Low-power, Soft error, Permanent fault, Partial redundancy

Contents

1 Introduction **1**

1. Motivation and Objective . . . 1
2. Contributions . . . 3
   - 2.1 Selective Check of Data-Path . . . 3
   - 2.2 Improvement of Robustness of Partial Redundancy by Using SDC Rate Prediction . . . 4
3. Organization . . . 5

2 Related Work **7**

1. Permanent Fault Detection . . . 7
2. Soft Error Detection . . . 8

3 Evaluation Framework **11**

1. LAPP/EReLA Architecture . . . 11
2. Level of Redundancy of EReLA . . . 13

4 Selective Check of Data-Path **16**

1. Introduction . . . 16
2. Construction ODP-Aware Data-Path . . . 18
   - 2.1 Selective Check Instruction in Redundant Data-Path . . . 18
   - 2.2 Calculation of Defective Probability . . . 20
   - 2.3 Optimizing of the ODP for Constant Input . . . 23
   - 2.4 Adding Check Instructions According to ODP . . . 25
3. Results . . . 28
   - 3.1 Workloads and Characteristics . . . 28

3.2 Reduction of Check Instructions 31
3.3 Dependability and Energy Savings 34
4. Summary and Discussion 36

5 Improvement of Robustness of Partial Redundancy by Using SDC Prediction 39
1. Introduction 39
2. Proposed Technique 41
2.1 Outline of the Proposal 41
2.2 SDC Rate Prediction 43
2.3 Effectiveness of using SDC Prediction 45
3. Experiment setting: assuming error rates from a radiation stress test 46
3.1 Error rates 46
3.2 Baseline platform and simulator settings 48
4. Results 48
4.1 Overall Reliability Vs. Energy Consumption 48
4.2 SDCs in individual program 51
4.3 Number of Test Blocks 52
5. Summary and Discussion 52

6 Conclusion and Future Work 56
1. Conclusion 56
2. Limitation and Future Work 58
Acknowledgements 60
References 62

List of Figures

3.1 Structure of LAPP. 12
3.2 Modified LAPP Architecture for EReLA. 14
3.3 Different redundancy level for EReLA. 15

4.1 DMR-based permanent fault locating. 17
4.2 Algorithm to selectively add check instruction. 19
4.3 Cost-effective permanent fault locating. 21
4.4 NAND gate error analysis for constant Inputs. 24
4.5 Area required for the constant input operations. 25
4.6 Adding check instruction on a DFG based on Eq. 4.1. 27
4.7 Three kinds of data flow graphs. 29
4.8 Incidence of instructions. 31
4.9 Incidence of instructions with a fixed input. 32
4.10 Ratio of check instructions for different thresholds. 37
4.11 Vulnerability to the second error for different check instructions. . 38
4.12 Energy savings for 10M executions. 38

5.1 Dependability of the traditional partial redundancy under different SEEs. 41
5.2 Example of partial redundancy. 42
5.3 Proposed partial redundancy. 43
5.4 Our predicted *SDC* rate as a function of `err_count`. 45
5.5 Improved error detectability by using Pr_{SDC}. 47
5.6 Breakdown energy consumption under different SEE rate. 51
5.7 SDC reduction in individual benchmarks from original to error prediction based partial redundancy. 54

5.8 Varying the number of test programs. 55

List of Tables

4.1 ODP of the baseline ISA for this study. 22
4.2 Benchmark programs. 28
4.3 Simulator specification. 30

5.1 Evaluation setup 49
5.2 Benchmark functions 50
5.3 Erroneous loop iterations of different redundant technique 50

Chapter 1

Introduction

THIS chapter introduces and motivates the current reliability problems in modern micro-architectures. It contains a list of contributions and a description of the organization of the rest of the dissertation.

1. Motivation and Objective

For decades, the number of on-chip transistors has grown at an exponential speed which was empirically predicted in Moore's Law. The growth of the transistors is achieved by advanced CMOS technology. The advanced CMOS technology provides many advantages in modern microprocessor design such as low energy consumption per transistor, low manufacturing cost, high operating frequency, and high density of transistors. On the other hand, while modern computer systems are utilizing the benefits of continued device scaling, there is a growing concern for the reliability of theses systems. Furthermore, high operating frequency and chip-level overheat accompanying the shrinking of the technology also increase the potential wear-out rates of transistors and inner-connections. These features also add new pressures on the correct states of transistors, especially, when working under an environment with Single Event Effects (SEE) occurrence [28]. As a result, the lifetime of the microprocessors and digital circuits becomes shorter and less predictable [14, 17, 23, 25]. Thus, architecture-level fault-tolerance [24] in microprocessor is becoming essential to maintain the correct calculation by using current scaled transistors which have been demonstrated to be increasingly prone

to faults.

Since the problem of building fault-tolerant systems is not new, one may argue that traditional solutions involving redundancy in space, time, and/or information [7, 18] can be leveraged to neutralize this pending reliability threat. The difference between the traditional reliability problem and the current growing threat caused by device scaling, however, lies in the affected segments of computing markets. Traditionally, hardware reliability mainly concerned high-end niche systems such as transaction processing systems for banks and mission-critical systems for space applications. As the main priority for designing these systems is to meet reliability goals, the budget spent on ensuring reliability is less constrained than mainstream systems and the use of solutions that involve heavy amount of redundancy, such as triple modular redundancy, is acceptable.

On the other hand, the scaling-induced reliability problem is fundamentally different. Modern computer systems have taken advantage of the ever-growing system integration made possible by device scaling, which will still continue in coming years. The reliability problem caused by scaling, therefore, will become a major issue of the computing market and hardware reliability will be a concern for the relatively low end but largely sold systems including personal computers, mobile devices and car control systems. In these market segments, because the budget that can be spent on reliability is much more limited than that of the high-end systems, an effective solution must incur low overheads in area, power, and performance in order to be broadly deployable. This precludes the use of traditional solutions that rely on expensive redundancy. To put it into perspective, the research challenge is to derive a low cost yet effective reliability solution that can be useful to the mass computing market. This dissertation also investigates such cost aware reliability solutions. However, while these previously proposed schemes have either focused on low-cost detection mechanisms or detection and recovery mechanisms for transient faults, we take a holistic system design approach to derive a complete reliability solution including detection, and recovery that handles both permanent and transient faults.

2. Contributions

2.1 Selective Check of Data-Path

Dual modular redundancy (DMR) is usually used to detect the fault in the combinational logic, where all the instructions are duplicated and check to keep a full understanding of defective units in the data-path, which adds pressure to the power utilization, and over-designed by not considering the criticality of fault in different operations. An alternative way is the light-weighted technique, which adds only one check at the end of the data-path. However, it allows the tainted data from the early stage to propagate inside the path. The erroneous data may taint more data, which possibly downgrades the reliability and becomes unpredictable when a second fault occurs. Thus, we have proposed:

- A technique to insert selective checks into the data-path based on the operation defective probability (ODP) in which the ODP has been calculated based on the number of gates used by an operation.

- The ODP calculation also takes the influence of constant input into account, i.e., certain inputs of the circuits can put parts of the circuit into a don't care zone. Potential errors in these gates will not contribute to the final potential error rate.

- The calculated ODPs have been used in the data-path to add check instructions. The branch of the data-graph with ODP larger than the threshold as a potential candidate to contain permanent fault, and check instruction has been added to that branch.

- The result has been shown that this technique achieves near-optimal dependability by reducing the check instructions to 60% and 22% energy can be saved by avoiding the thorough checking.

Published papers related to this work:

- T. Ahmed, J. Yao, Y. Hara-Azumi, S. Yamashita, and Y. Nakashima, "Selective Check of Data-Path for Effective Fault Tolerance", IEICE transaction of Information and Systems (Special Section on Reconfigurable Systems), Vol. E96-D, No. 8, pp. 1592-1601, Aug., 2013.

- T. Ahmed, J. Yao, and Y. Nakashima, "Introducing OVP Awareness to Achieve an Efficient Permanent Defect Locating", in proceedings of the 2012 IEEE/ACM Symposium on Nanoscale Architecture, Nanoarch,'12, pp. 43–49, July, 2012.

- T. Ahmed, J. Yao, and Y. Nakashima, "Achieving Effective Fault-Tolerance in FU Array by Adding AVF Awareness", IPSJ SIG NOTEs, vol-2012, no-5.

2.2 Improvement of Robustness of Partial Redundancy by Using SDC Rate Prediction

In order to achieve the reliability in program executions with the smallest hardware cost or performance loss, partial redundancy is becoming popular. This method categorizes the program into zones of different importance and then treats them accordingly. As an example, loop index calculation in an image processing application is more important than the calculations of the pixels. Thus, the loop index calculations are duplicated whereas the rest of the loop body calculations remain same. In this way, this method guarantees 100% correct loop execution with smallest performance loss and energy consumption. Under partial redundancy, the loop body calculations are executed once, and a fault strike on these calculations cannot be detected, which are treated as silent data corruptions (SDC). Thus, the partial redundancy lacks the ability to tolerate worst-case situations. Under a burst of single event effect (SEEs), it is still possible that the rate of SDCs reaches an intolerable level, even though the program control flow is carefully maintained. Moreover, the data calculation part is more likely to accumulate errors because of its code size, which is usually 4x or 5x larger than the control part. The dependability of the traditional partial redundancy under different SEEs. In order to overcome the problem we have proposed:

- A method to predict the error rate in the next loop execution, and predict the error rate in the next loop execution, and calibrating the redundancy level according to the prediction.

- A self-test program has been used to collect the error samples of past several cycles.

- Results show that with 8% cost, the proposed technique can reduce the silent data corruption from 12% to a 0.37% level under a Stress Radiation Test simulation.

Published papers related to this work:

- T.Ahmed, J. Yao, and Y. Nakashima, "A Tow Order Increase in Robustness of Partial Redundancy Under a Radiation Stress Test by Using SDC Prediction", IEEE Transaction on Nuclear Science (accepted).
- T. Ahmed, J. Yao, and Y. Nakashima, "A Two Order Increase in Robustness of Partial Redundancy Under Radiation Stress Test by Usign SDC Prediction", in proceedings of the 2013 European Conference on Radiation Effects on Components and Systems, RADECS 2013, Oxford, United Kingdom, Sept., 2013.
- T. Ahmed, J. Yao, and Y. Nakashima, "Achieving Near-Optimal Dependability with Minimal Hardware Costs in an FU Array Processor by Soft Error Rate Monitoring", SWoPP'12, IPSJ SIG Notes, 2012-ARC-201(19), Aug., 2012.

3. Organization

The rest of this dissertation is organized as follows:

- Chapter 2 describes the related works to detect and recover the permanent fault, as well as the soft error.
- Baseline architecture for these experiments and the workloads for the evaluation are discussed in Chapter 3.
- Chapter 4 shows the technique to detect the permanent fault by adding selective check inside the data-path, and the experimental results are also discussed.
- Method the improve the robustness of partial redundancy by SDC prediction and the experimental results of this method are discussed in Chapter 5.

- Finally, Chapter 6 concludes this thesis with the conclusion, limitation and future works.

Chapter 2

Related Work

THIS chapter describes the previous works those are either used in, or directly related to this dissertation. In Section 1, related works on permanent fault detection are described, and partial redundant techniques to detect and recover the soft errors on micro-architecture are described in Section 2.

1. Permanent Fault Detection

Different techniques have been proposed to protect microprocessors from electronic errors. Error correcting code (ECC) is applicable on memory to detect and recover bits of errors. However, it can hardly be applied in non regular pattern units such as ALUs and is also unaffordable for pipeline latches due to the delay issue. Redundant exexution on the processor core shows a tolerable performance impact on error detection. A number of redundancy techniques [4, 7, 16] have been proposed to detect and recover from errors. These approaches mostly use redundant multi-threading to detect erroneous execution. In this kind of approach, two or three identical program threads are generated in the micro-architecture and are then executed in the same time frame. The output values are compared to verify the execution correctness. This provides fault detection throughout the entire lifetime of a processor, so no errors will go undetected.

Error detection by duplicated instructions for a super-scalar processor has been proposed in [16], where all the instructions are duplicated and checked thereafter. Similarly, a fault-tolerant framework for a VLIW processor has been

proposed in [3, 4]. This approach also detects the errors by duplication, and the results are checked with additional checkers. Sensitive applications such as multiplication and addition are replicated thrice and compared with a majority voting to select the correct results. This architecture supports real-time fault-tolerance.

In order to meet the challenges of the increasing speed and performance requirements of various applications, reconfigurable hardware [8, 9, 27] has shown its popularity under the demands of flexibility and low non-recurring engineering cost. A fault-tolerant coarse grained architecture [2] has been proposed for reconfigurable architecture. In this architecture, an error correcting code and residue arithmetic have been used on the memory structure and processing units, respectively. However, keeping the reliable execution in the coarse grain architecture by means of residue arithmetic is of relatively high cost, in the area and power consumption overhead, as every functional unit of the array uses additional circuitry for the verification of data correctness.

Thus, in this work our main target was to reduce the checking instructions with a negligible penalty in reliability. In order to achieve the goal, we add selective check instructions on the data-path. Check instructions are added on the data-path based on the operation defective probability (ODP) of an operation. The ODP of an operations is calculated according to the number of gates the operation uses. The branch of the data-flow-graph (DFG) with ODP larger than a threshold is regarded as a potential candidate to contain permanent faults, and a check instruction will be inserted at that branch to help a deterministic verification. Furthermore, our ODP calculation also takes the influence of special inputs into account, i.e., certain inputs of circuits can put parts of the circuits into dont care zones. Potential errors in these gates will not contribute to the final potential error rate and are thus carefully removed from the influence chain, in order to further reduce the number of check operations without affecting the dependability.

2. Soft Error Detection

Likewise, dual modular redundancy [16,18,30] is also used to detect the transient faults in a micro-architecture. However, due to the duplication of the full execution, given by both the time and the spatial redundancy, the energy consumption is doubled, which is not desirable in most circumstances given that energy consumption has already become the most important constraint for microprocessor design. A partial redundancy with an awareness of data importance was thus proposed for this purpose [6, 13, 20] and has gained in popularity, especially in systems with critical requirements for battery life. This technique categorizes the data into zones of different importance and then treat them accordingly. For example, in an image processing application loop index controls all pixels calculations and is therefore regarded to be more important than the pixel data. The corruption of the loop index usually leads to the abnormal end of the program and thus need to be fully covered by error-resilience techniques. The other data including processed pixels can tolerate a limited number of errors. The partial redundancy can thus use an imperfect but low-cost fault-tolerance.

Apart from image processing, applications based on approximate calculations such as decision making from probability exploration, machine learning, computer vision, and so on, may similarly apply different dependability/cost methods for their various types of internal data [13].

Basically, partial redundancy trades dependability for low power, and is thus best suited for approximate programming. A typical approximate programming—EnerJ [20]—has been proposed based on information-flow tracing. In EnerJ, variables and objects are declared as either approximate or precise. The approximate data are then processed, for energy saving purposes, on low-power but less reliable units than the precise data. A follow-up research [6] gives a detailed architecture support for the programming model of EnerJ, where an extra bit is added to the instruction set architecture (ISA) to serve as the precise or the approximate annotation. Compiler supports are also discussed in paper [6] to generate the binary with the new annotation. Their results showed that by treating the data differently, significant energy savings (10% – 50%) can be achieved at a very low accuracy cost. On average, 43% energy can be saved according to the workloads in their study.

Different from the EnerJ system that works on in-core level reliability control, error resilient system architecture (ERSA) [13], was proposed to handle approximate computation in a multi or many core system. The target applications are mainly probabilistic-based ones such as K-mean clustering, Low-Density Parity-Check, and Bayesian network inference. In ERSA, the cores are respectively divided into super reliable and relaxed reliable categories. Super reliable cores are responsible for executing the main threads, in which critical data like control flow are processed. They are also responsible for supervising the relaxed reliable cores. Super reliable cores are very limited in number. Most of the cores are relaxed reliable cores, where only the memory management unit and the instructions cache are reliable. The tasks were distributed among the super reliable and relaxed reliable cores based on their importance. Results showed that, ERSA can still achieve an accuracy of 90% while 2×10^{-4} error/cycle/core was injected into the architecturally-visible register.

Silent data corruptions (SDCs) will be the major QoS drawback for these energy saving systems based on approximate computation. The SDCs arise from either unreliable execution as the low-power execution or the relaxed core intentionally ignores the error for better energy-efficiency. Due to its simple and light-weight design, this technique is not even able to keep track or report on the SDC damage. These SDCs will add uncertainty to the QoS, and execution may become unpredictable especially under a relatively high error rate. Research was recently carried out on error propagation execution [29], indicating that most errors will become derated ones. However, SDCs still give 57% future abort, when the erroneous data are further used as control data or for memory accesses. Reducing the SDCs is our target in this study.

Therefore, in this work our target is to reduce the SDC by calibrating the past execution and predicting the future error rate. In order to achive the target, a small test program is executed along with the program data-flow-graph (DFG) to sample the error rate. A strategy unit is then used to provide a bias between the partial and the full redundancies based on on-the-fly error monitoring of SEE data, together with the instructive program characteristics of the coming program interval.

Chapter 3

Evaluation Framework

The purpose of this chapter is to give an overview of the evaluation framework used for both of the experiments. EReLA framework has been used for the experiments in this dissertation. EReLA is a dependable reconfigurable architecture, which can detect and recover the soft error as well as detect, locate and recover the permanent fault. EReLA has been developed based on **L**inear **A**rray **P**ipeline **P**rocessor, which was previously proposed in the Computing Architecture Lab. LAPP architecture is described in Section 1, and in Section 2, a detail about the EReLA framework with different level of redundancies is given.

1. LAPP/EReLA Architecture

LAPP architecture is depicted in Fig. 3.1. In can be found in Fig. 3.1 that LAPP has two major blocks: (1) pipeline processor, and (2) the functional unit (FU) array. Stage-0 in Fig. 3.1 is a normal VLIW pipeline processor, which executes the normal instructions in a program. Stage-0 has been extended to several stages of FU array by extending the EX and MEM units, which is used to accelerate the parallel loop kernel in a program. Each stage of the FU array contains: (1) an instruction mapping unit (MAP), which maps all the VILW instructions of the loop kernel onto the array, (2) an EAG unit to perform the memory access operations, (3) EX units depending on the size of the VLIW instruction, and (4) a BRC unit to perform the branch operations. A 4-way L0$ cache is used at every stage.

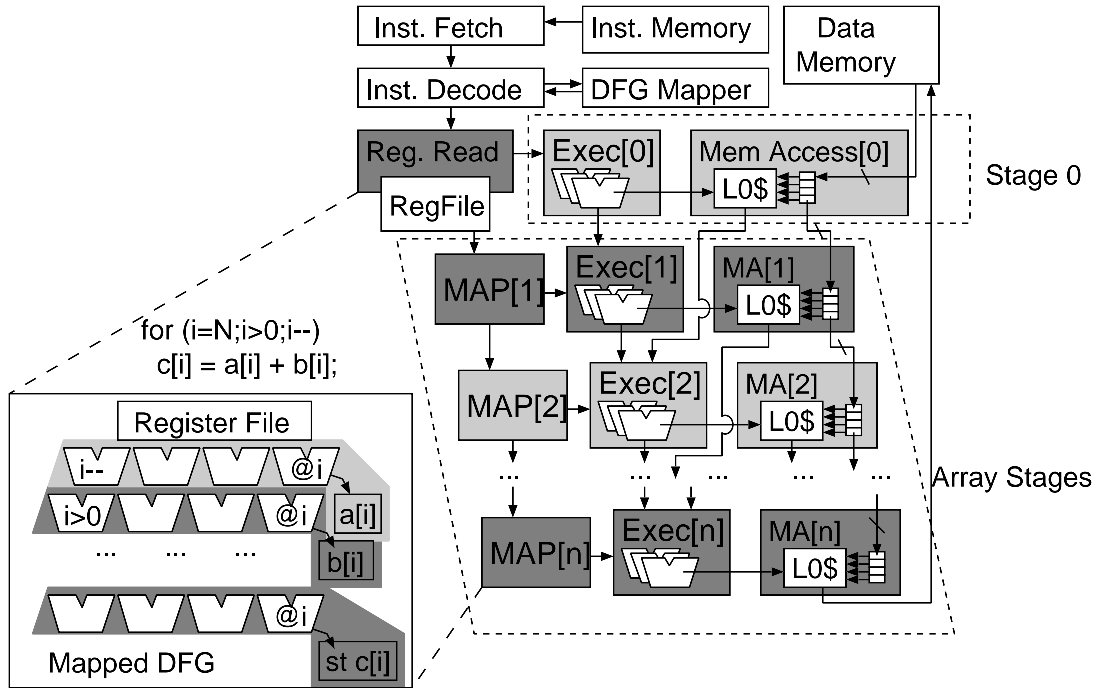


Figure 3.1. Structure of LAPP.

The basic idea for acceleration is to map each instruction in a loop kernel cycle by cycle onto the FUs in each stage. As a result, instructions of different loop iterations can exist simultaneously inside the array and the executions will be accelerated. The FU array takes a two dimensional form, in which each row can map multiple instructions of the same execution cycle, and the vertical direction is used for a full speedup by exploiting parallelism between different loop iterations. In order to improve the usage of the FUs, instructions without dependences, which can be executed in the same cycle should be mapped along the row direction of the FU array. LAPP use FR-V, which is a widely used commercial VLIW architecture and is supported by GCC, as the target ISA.

LAPP achieves high IPC by mapping VLIW instructions onto the FU array efficiently. The LAPP has 3 execution modes: (a) Normal-Execution, (b) Array-Setup, and (c) Array-Execution.

During Normal-Execution, the VLIW pipeline only executes VLIW instructions as a traditional VLIW processor and the array pipeline is halted. However, in Array-Execution, LAPP additionally uses the FU array to exploit parallelism.

Array-Setup is invoked when a loop kernel is detected. In Array-Setup, MAPs start mapping each VLIW instruction in a loop kernel cycle by cycle onto the FUs in each stage. The MAPs also start configuring the interconnection between the FUs by arranging multiplexers to forward values of registers from one stage to its succeeding stages. Further details about mapping are explained in [31]. Data prefetching between the L1$ and level-2 unified cache (L2$) is started at the same time. Using this overlapping time, the overhead of mapping and network configuration can be hidden by the prefetch delay, as the data prefetching is usually longer than mapping and configuration time.

LAPP invokes Array-Execution, after the mapping, network configuration and prefetching are completed. Based on the well mapped VLIW instructions and interconnections, Array-Execution can process these instructions in a highly parallel fashion. Data from L1$ flows into the level-0 Data cache (L0$), which is a kind of local memory for each stage, toward the following stages. The interconnection for forwarding the contents of registers and the L0$ is maintained in Array-Execution. According to this design, the LAPP can produce the result of one loop iteration per clock cycle. The resulting data are stored back into L1$.

LAPP architecture has been used for EReLA framework. In order to achieve high dependability LAPP architecture has been modified such as module for automatic instruction duplication, error propagation chain to carry the error at the end of the array stage, and permanent fault recovery scheme are added. Fig. 3.2 shows the modified Architecture of LAPP. It Can be found in Fig. 3.2 that a chain of flip-flop, multiplexers, and XOR gate has been used to propagate the error information of different stages. In each stage a multiplexers has been used to select the proper signals, which is basically a check instruction. The error flag of each stage is propagate using the flip-flops and XOR gates.

2. Level of Redundancy of EReLA

EReLA uses different level of redundancies, which have different energy requirements and reliability. Three levels of redundancies have been proposed for EReLA. Different levels of redundancies along with an original source program has been depicted in Fig. 3.3.

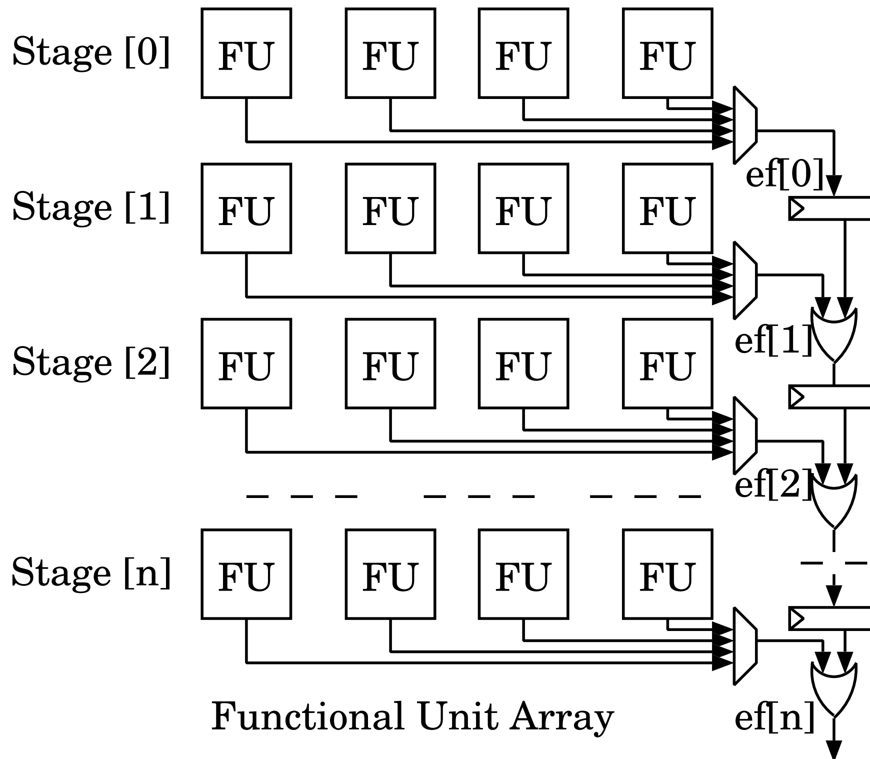


Figure 3.2. Modified LAPP Architecture for EReLA.

Level-1 in EReLA duplicates and checks only the loop index calculation of a program, and the loop body calculations are not duplicated. This level of redundancy is same as the partial redundancy discussed in Chapter 2, and the main objective of this technique to overcome the loop abortion error. Fig. 3.3(b) shows an example of level-1 redundancy. It can be found in the program that the loop index calculation is duplicated and checked in the next instruction. The duplicated instruction in Fig. 3.3 is shown in small letters. The rest of the instructions remain same as in Fig. 3.3(a).

A fault strike on the instructions, those calculate the loop index, can be detected by the check of the next instruction, and the error generated by the check instruction will propagate through the error propagation chain. The error is usually recovered by re-executing the loop. Thus, this level of redundancy ensures 100% loop execution. However, fault strike on the loop body calculations cause silent data corruption (SDC). Experimental results have proofed that Level-

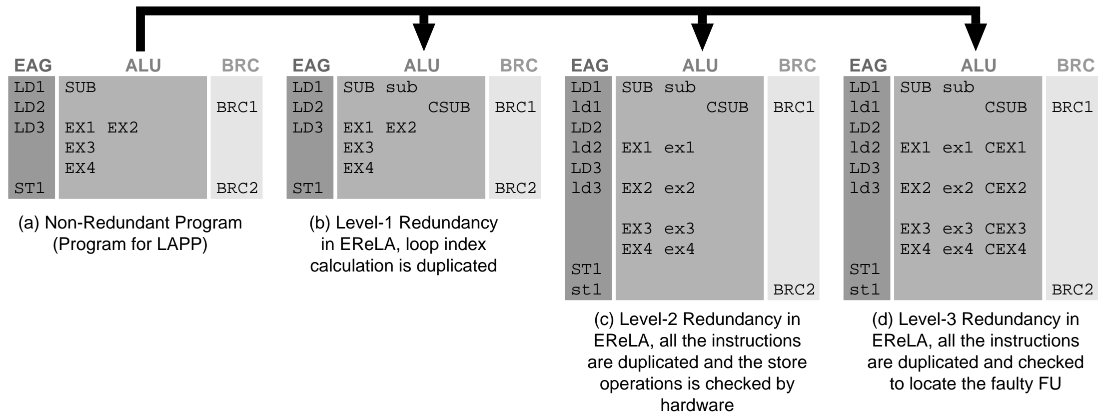


Figure 3.3. Different redundancy level for EReLA.

1 has a very good balancing between reliability and energy consumption while the fault injection rate is very low.

Level-2 redundancy is depicted in Fig. 3.3(c), which is mainly proposed to overcome the limitations of Level-1. Level-2 ensure 100% fault free execution of a program. It can be found in Fig. 3.3(c) that all the instruction of Fig. 3.3(a) is duplicated according to the availability of resources. For example, there is only one load/store unit in and thus the load and store instructions are duplicated in different level. On the other hand, the arithmetic and logic operations are duplicated in the same level. In the EReLA architecture of the store instructions has been implemented with a built in check instructions. Therefore, there is no additional check instruction on the pseudo code. Level-2 also recover the error by re-execution.

Level-3 has been proposed to detect and locate the permanent fault of the array. An example of the Level-3 program is shown in Fig. 3.3(d), where all the instructions are duplicated and checked in the later instruction. While the Level-2 redundancy will report very frequent error Level-3 is used to locate the faulty functional unit of array. With the help of the check instructions faulty pair of the functional units can be located. However, to locate the exact faulty functional unit a technique has been proposed in [10].

Chapter 4

Selective Check of Data-Path

THIS chapter describes the technique to add selective check instructions on the data-path to detect the permanent fault. Calculation and optimization of operation defective probability (ODP), results and summary of this technique are also discussed.

1. Introduction

Error check and correction (ECC) is a conventional fault tolerant technique. Although ECC logic has been effectively working for data-center memory systems [12], it may be not suitable for reconfigurable systems with many FUs, which mainly consist of combinational logic. In order to detect the faults in combinational logic, on-line tests [1,22] and explicit checks [16] have been used. On-line test [22] is not real-time, while the real-time explicit check [16] increases power consumption continuously and visibly. Fig. 4.1(a) gives an example of exhaustive check to keep a full understanding of defective units in the data-path. The thorough check adds pressure to the power utilization limitation, and may be an over-design by not considering the criticality of faults in different operations. An alternative way is a light-weighted technique, which add only one check at the end of the data-path (Fig. 4.1(b)). However, it allows the tainted data from the early stage, such as `I1'` in Fig. 4.1(b), to propagate inside the path. The erroneous data may taint more data, which possibly downgrades the reliability and becomes unpredictable when a second fault occurs.

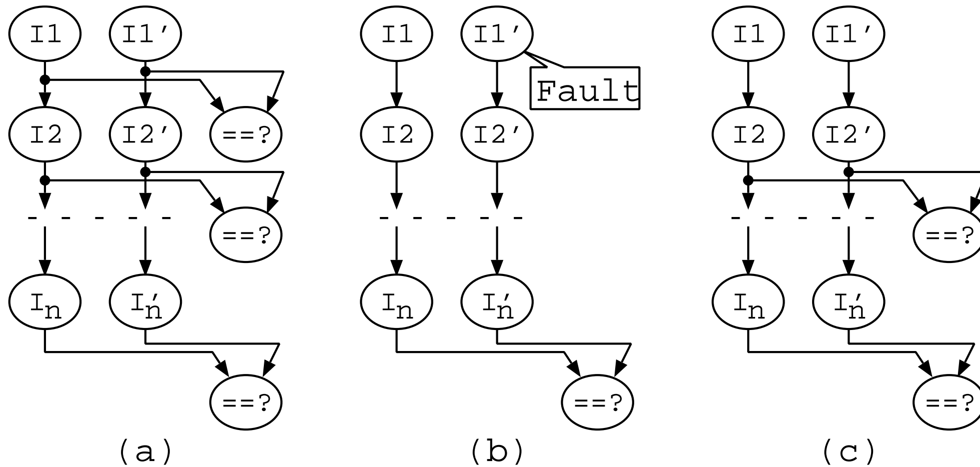


Figure 4.1. DMR-based permanent fault locating: (a) Exhaustive check technique, (b) Light-weighted technique, and (c) Selective check based technique.

Unlike both Fig. 4.1(a) and Fig. 4.1(b), in this research we propose to insert selective checks into the data-path based on the operation defective probability (ODP), as shown in Fig. 4.1(c). The ODP of an operations is calculated according to the number of gates the operation uses. The branch of the data-flow-graph (DFG) with ODP larger than a threshold is regarded as a potential candidate to contain permanent faults, and a check instruction will be inserted at that branch to help a deterministic verification. Furthermore, our ODP calculation also takes the influence of special inputs into account, i.e., certain inputs of circuits can put parts of the circuits into *don't care zones*. Potential errors in these gates will not contribute to the final potential error rate and are thus carefully removed from the influence chain, in order to further reduce the number of check operations without affecting the dependability.

The results show that our proposed technique achieves near-optimal dependability by reducing the checking instructions to 60%. By allowing a slight increase of the error propagation rate (2%), our approach is able to further reduce the number of check instructions to a 37% level, as compared to the exhaustive check method. The removal of non-critical check instructions can contribute to the energy saving in the dependable execution. Under the allowance of 2% error propagation rate execution, 22% energy can be saved by avoiding the thorough

checking.

2. Construction ODP-Aware Data-Path

2.1 Selective Check Instruction in Redundant Data-Path

In this research, we mainly focus on reconfigurable architectures [8,9] which contain a large pool of resources to exploit extreme parallelism. We use LAPP [5,31] as our baseline architecture, which is a reconfigurable architecture containing an array of FUs for accelerating image processing applications. LAPP consists of a large set of combinational units and networks to which ECC protection cannot be applied as easily as in the data-center research [12] with an acceptable cost. Several architectural methods have already been proposed to achieve a relatively effective fault tolerance in combinational logic. Specifically, dual modular redundancy (DMR) with a check after the dual executions can guarantee that no soft error goes undetected [16]. As reconfigurable architectures such as LAPP [5,31] are originally rich in resources, DMR with checks from software level can be applied flexibly and efficiently with the understanding of the DFG inside the architecture.

To detect the permanently defective unit inside the DMR execution, the straightforward method is to exhaustively check all the instruction executions in order to gain the information of all units used, as have been shown in Fig. 4.1(a). After that, on-line test or tuning can locate the defective unit at that erroneous spot. An alternative way is to add one check instruction at the end of data-path (Fig. 4.1(b)). However, it downgrades the dependability as described in Section 1. In this research, we use selective check instructions together with DMR to create correct executions and locate the possible erroneous spots. Fig. 4.2 gives a brief introduction of this method.

Fig. 4.2 gives the algorithm to put the duplicated DFG into the FU array. During the mapping of the DFG, basically, each instruction will be duplicated by `map(i,i')`. Along with the mapping, we also study the vulnerability of each instruction by using `get_vulnerability()`, which provides the possibility of permanent fault inside this hardware unit. Later sections will give a detailed

```
sum = 0; /* Accumulated vulnerability in data-path */
last_need_check = 0; last_inst = NOP;

while (!end(DFG)) {
    /* To map data-flow-graph (DFG) in FU array */
    i = fetch_inst(PC);

    /* Duplicate & selective check in map */
    if (last_need_check)
        map(i, i', check(last_inst));
    else
        map(i, i');

    /* Analyze vulnerability of instruction */
    sum += get_vulnerability(i);
    if (sum > th) {
        /* Critical spot */
        last_need_check = 1;
        sum = 0;
    }
    else
        last_need_check = 0;
    last_inst = i;

    PC++;
} /* end of this cycle */
```

Figure 4.2. Algorithm to selectively add check instruction.

explanation of `get_vulnerability()`. Only when the accumulated vulnerability '`sum`' becomes larger than a predetermined threshold '`th`', a check instruction will be added, following `map(i,i',check(last_inst))`, where '`last_inst`' is the instruction in previous cycle. '`last_inst`' is used in the check because the results of the duplicated instructions will be known in this cycle. By this way, the

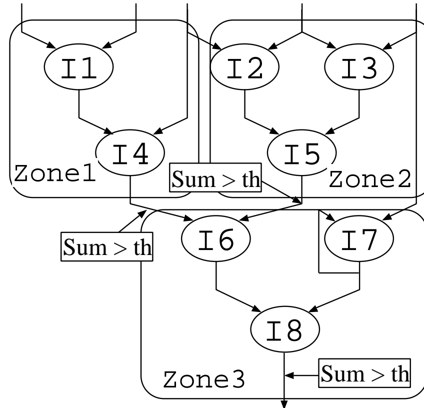
data-path in Fig. 4.1(c) is obtained.

By introducing a selective check according to ODP, the number of check instructions is reduced, as shown in Fig. 4.3(b). With the help of selective check instructions, only the segment with an error report needs to be checked in detail for the defective unit. For example, in Fig. 4.3(b) the `chk-I5` is the first check instruction to report an error, since `I2` is mapped on a defective unit. Thus, only the dependent instructions (in Fig. 4.3(a)), `I2` and `I3`, need to be checked to locate the permanent failure as shown in Fig. 4.3(c). At the same time, due to the previous checks, the instructions inside other segments are judged to be previously mapped inside the correct units. Therefore, they need to be inside the DFG only for the completion of the data-path. No DMR execution is necessary for these instructions, as is shown in Fig. 4.3(c). Consequently, the number of check and redundant instructions is reduced significantly during the permanent defect location. Although the period of locating permanent fault in Fig. 4.1(a) is short, we are not increasing 1/3 power consumption by avoiding exhaustive checking. The burst of power hungry mode as in Fig 4.1(a) is thus avoided.

2.2 Calculation of Defective Probability

Mukherjee and et al. [15] have stated that various programs will respond differently to the same fault rate, according to their different architectural vulnerability factors (AVFs). AVF gives a measure of the probability that a fault will turn into a visible error. Given that the soft error occurs in a certain memory block, it will become an error only when the latter calculation depends on this faulty block. The AVF of a program depends on its working behaviors, especially the memory access intensity and frequency.

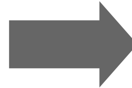
Similarly, we are using the idea of operation defective probability (ODP) in this research to selectively add data verification instructions. We extend the above AVF, which is for soft errors only, to handle permanent faults. In this research, we treat the permanent fault occurrence probability as in proportion to the gate number inside the functional unit. It is expected that the possibility of the wear-out faults as well as the manufacturing faults is increased with the number of gates used inside the functional unit. For example, a 1-bit AND operation requires two 2-input NAND gates, while a 1-bit XOR operation uses four 2-input



(a)

Stage	FU0	FU1	FU2
0	I1	I1′	
1	I2 (x)	I2′	
2	I3	I3′	
3	I4	I4′	
4	I5	I5′	chk-I4
5	I6	I6′	chk-I5
6	I7	I7′	
7	I8	I8′	
8			chk-I8

(b)



Stage	FU0	FU1	FU2
0	I1		
1	I2	I2′	
2	I3	I3′	chk-I2
3	I4		chk-I3
4	I5	I5′	
5	I6		chk-I5
6	I7		
7	I8		
8			

(c)

Figure 4.3. Cost-effective permanent fault locating: (a) DFG, (b) DMR mode with reduced check instructions, (c) Locating a permanent fault after `chk-I5` reports error.

NAND gates. As a result, the lifespan of the XOR will be relatively shorter than the AND unit under a given gate defect ratio. Applying the consideration to arithmetic operations, the defective probability to a permanent defect may be even larger due to the large area and the complex wire interconnections inside the arithmetic operations. For example, a 1-bit full adder takes nine 2-input NAND gates to finish the calculation. A large word-length multiplication uses several stages of adder chains and partial product generators. These units are thereby

vulnerable to soft error and permanent defects because of their large hardware areas and relatively long data paths.

Table 4.1. ODP of the baseline ISA for this study.

Types of Operations	Operations	Number of Gates	Defective Probability (%)
Logic	AND	176	0.10
	OR	176	0.10
	XOR	208	0.12
	SLL/SRL	1,020	0.58
Arithmetic	ADD	892	0.50
	SUB	1,022	0.58
	MUL	5,130	2.90
	SLA	1,005	0.57
Media	MSRL	792	0.45
	BYTE-HALF	219	0.12
	SUML/H	996	0.57
	HALF BYTE	854	0.48
	SAD	2,970	1.69
	UADD	1,320	0.75
	USUB	1,398	0.86
	MUL	2,569	1.46
Memory	LOAD	892	0.50
	STORE	892	0.0

We give ODP to permanent defects in Table 4.1 by studying the typical operations from the FRV Instruction-set architecture [26]. As shown in Table 4.1, these operations can be categorized into four types: logic, arithmetic, media and memory. Although the FUs for these operations may merge and share circuits between operations to achieve the optimized design in both power and area, in this research, we implement these operations into separated units, from the viewpoint that each operation takes an independent and individual path in the FUs

and for every calculation, only the corresponding path is activated. The FUs are implemented into Verilog HDL modules and then synthesized by Design Compiler with a 180nm cell library to obtain the area in the number of gates, as listed in Table 4.1. In addition, we treat AND operation as having a defective probability of 0.1%[1]. The other vulnerable probabilities are thus calculated by $0.1\% \times \frac{\text{Area}_{\text{OP}}}{\text{Area}_{\text{AND}}}$. As discussed above, logic operations are relatively less complex in hardware than other units, and their ODPs are thus relatively small. The arithmetic instructions take medium ODPs, except for the very large multiplication unit whose ODP reaches 2.9%. The media operations are a combination of logic and arithmetic operations and thus tend to show large ODPs. Finally, for LOAD instruction, it has been assumed that the memory is protected with error correcting code (ECC) so that the loaded data can be regarded as error free results. The only vulnerability in LOAD comes from the address calculation part which is the same as the ADD operation. To guarantee that there will be no tainted value by faults to the data storage, in our high dependable LAPP, the STORE operation is originally designed to take checked data before the real commitment, by embedding a check instruction inside it to check both data and address. For this reason, although the address calculation part of STORE still contains 892 gates, the ODP of this address calculation will be updated to 0% as the in-store check determines the correctness of the address and makes the defective possibility to 0%.

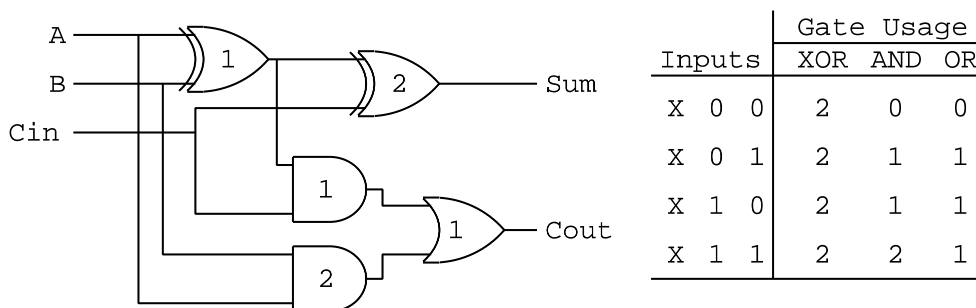
2.3 Optimizing of the ODP for Constant Input

In Section 2.2, we calculated the ODP of a particular operation based on the size of the circuit. Due to the behavior of certain gates such as NAND2, NXOR2, the sensitivity to faults—measured as ODP in our approach—can be further reduced when a constant value is provided to an operation. As an example, given a constant LOW input in an AND gate, it will always have the LOW output. Thus, any fault (stuck at 0 or stuck at 1) on the other input wire of this AND gate is masked and turned into don't care. Similarly, with an input provided to be logic HIGH, an OR gate can ignore the sensitivity of faults in its other input

[1]Although the value may be far larger than practical meanings, we simply use it here to demonstrate how our methods work accordingly to these assumed ODP values.

ports. As a result, compared to operations with variable inputs from prior circuits or register files, an operation with an immediate source operand will demonstrate less sensitivity to faults, since part of its sub-circuits can be logically masked into don't care zones which contribute 0% ODP.

Rather than counting the insensitive inputs, we use don't care gates by the constant value in order to update the ODP with immediate operand in our approach. Fig. 4.4 shows the method to calculate the sensitive gates of an adder while one input is constant. In Fig. 4.4, the gate-level schematic of the adder and a table containing the gate usages for different constant input patterns have been shown. For the pattern $<\mathrm{X}, 0, 0> = <\mathrm{A}, \mathrm{B}, \mathrm{Cin}>$ the input 'A' is a variable and 'B' and 'Cin' are LOW. Two XOR gates are used for calculating the sum. However, two AND gates can be replaced with wires since both have a constant LOW input. Again, similarly for the input pattern $<\mathrm{X}, 1, 0>$ AND gate 1 can be ignored. This illustrates the sensitive gates for a constant input has been reduced. Accordingly, we change the ODP calculation from $0.1\% \times \frac{\mathrm{Area}_{\mathrm{OP}}}{\mathrm{Area}_{\mathrm{AND}}}$ to $0.1\% \times \frac{\mathrm{Area}_{\mathrm{OP}} \times \mathrm{S}(\#\mathrm{imm})}{\mathrm{Area}_{\mathrm{AND}}}$, where S(#imm) represents the ratio of sensitive circuit area under the given #imm inputs.



Inputs			Gate Usage		
			XOR	AND	OR
X	0	0	2	0	0
X	0	1	2	1	1
X	1	0	2	1	1
X	1	1	2	2	1

Figure 4.4. NAND gate error analysis for constant Inputs.

Based on the above assumption, the sensitive gates are calculated for the operations of our baseline ISA. The average of different constant values of the benchmark functions for an operation is included in Fig. 4.5. Specifically, the operations with large areas under variable inputs can be significantly reduced to a level of halved or even smaller circuit areas. The originally small-sized operations such as AND, OR and BYTE-HALF do not show visibly important area reduction, due to their relatively less complex implementation. All these

results thus give a more accurate estimation of the ODP inside a real program, which is able to help reduce further the number of exhaustive checks and lower the resource utilization. The approach to calculate the ODP of operations in data-flow-graph (DFG) will be introduced in the next subsection by using the updated ODPs.

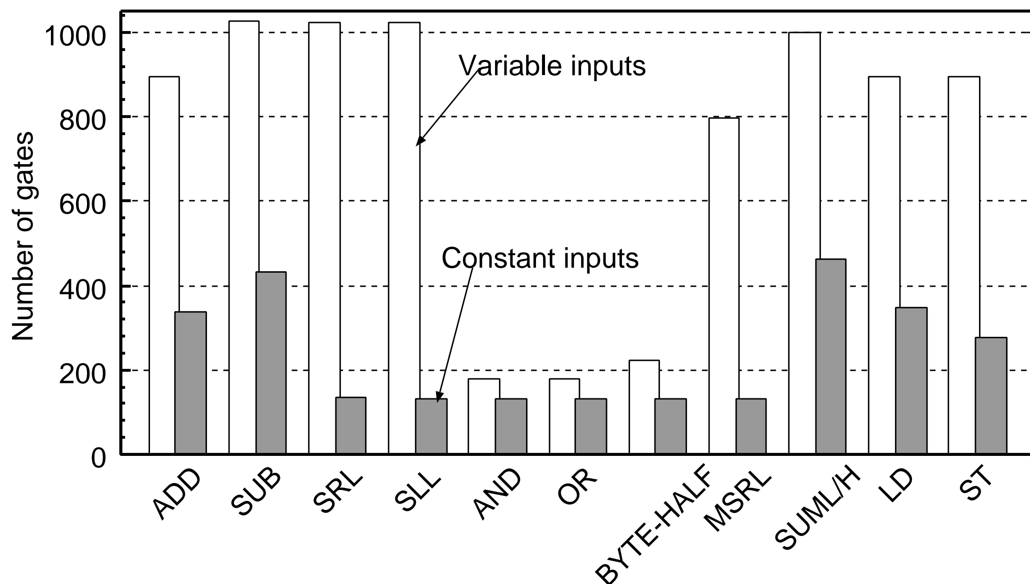


Figure 4.5. Area required for the constant input operations.

2.4 Adding Check Instructions According to ODP

Table 4.1 shows the defective probability of each instruction. Assuming that each operation takes two source operands and produces one result, we can design `get_vulnerability()` in Fig. 4.2 by calculating the probability of the error of the result as follows:

$$1 - \mathrm{Pr(out)} = (1 - \mathrm{Pr(op)}) \prod_{\mathrm{i}=1}^{2} (1 - \mathrm{Pr(s_i)}) \tag{4.1}$$

$\mathrm{Pr(s_1)}$ and $\mathrm{Pr(s_2)}$ are the probabilities of errors in the source operands, while Pr(op) is the error probability coming from the operation itself. It can be imagined that the Pr(op) has a direct connection to the defective probability in Ta-

ble 4.1, augmented with the insensitivity from constant inputs in Section 2.3. Assume that the whole data path starts from several checked inputs such as values from the ECC-ed register file which has 0% probability of error. The output of the first operation will have a probability of error regarding the operation itself. The successive dependent data will inherit this probability of error and adds a new probability when the data goes forward through the data flow graph. Although the values of defective probability in Table 4.1 are actually much larger than a practical probability of error, we are still directly using these values as Pr(op) in the remaining parts of this chapter to introduce the idea. By this means, we are able to tag the results with the probability of permanent error inside the whole data path.

Fig. 4.6 gives a detailed illustration of using ODP in `get_vulnerability()` in algorithm of Fig. 4.2. For simplicity, we assume that the threshold of error probability is 1.0%. The data flow graph starts by taking inputs `R1`, `R2`, `R3` and `R4` from the register file or memory, which are previously checked results and protected by ECC. It ends by committing the final result `R6` into the memory. Basically, every operation will be doubly executed and the final result `R6` will be compared before being written to memory.

In Fig. 4.6, the defective probability of each operation is shown inside the operation. After stage 2, both `R1` and `R2` get error probabilities that exceed the threshold. The check instruction is thus added to make a fast determination whether or not an error has happened there. This also covers Zone1 and Zone2, as shown in Fig. 4.6.

It is possible that the data-path will take backward bypassing data, such as `OP7(R4+=R5)` in Fig. 4.6. Considering that this data path represents a loop body, operation `R4+=R5` takes its first operands from the register file in the first iteration and updates itself afterward. For the first iterations, the vulnerability of the output of `OP7` denoted as $\mathrm{Pr(R4[1])}$, is calculated as previous equation (4.1). From the second iteration using Bayes' theorem the correctness of the output of `OP7`, as $\mathrm{Pr(R4[2]\,correct)}$ is calculated as:

$$\mathrm{Pr(R4[2]\,correct)} = \mathrm{Pr(R5, OP7\,correct)} \times \mathrm{Pr(R4[1]\,correct \mid R5, OP7\,correct)} \tag{4.2}$$

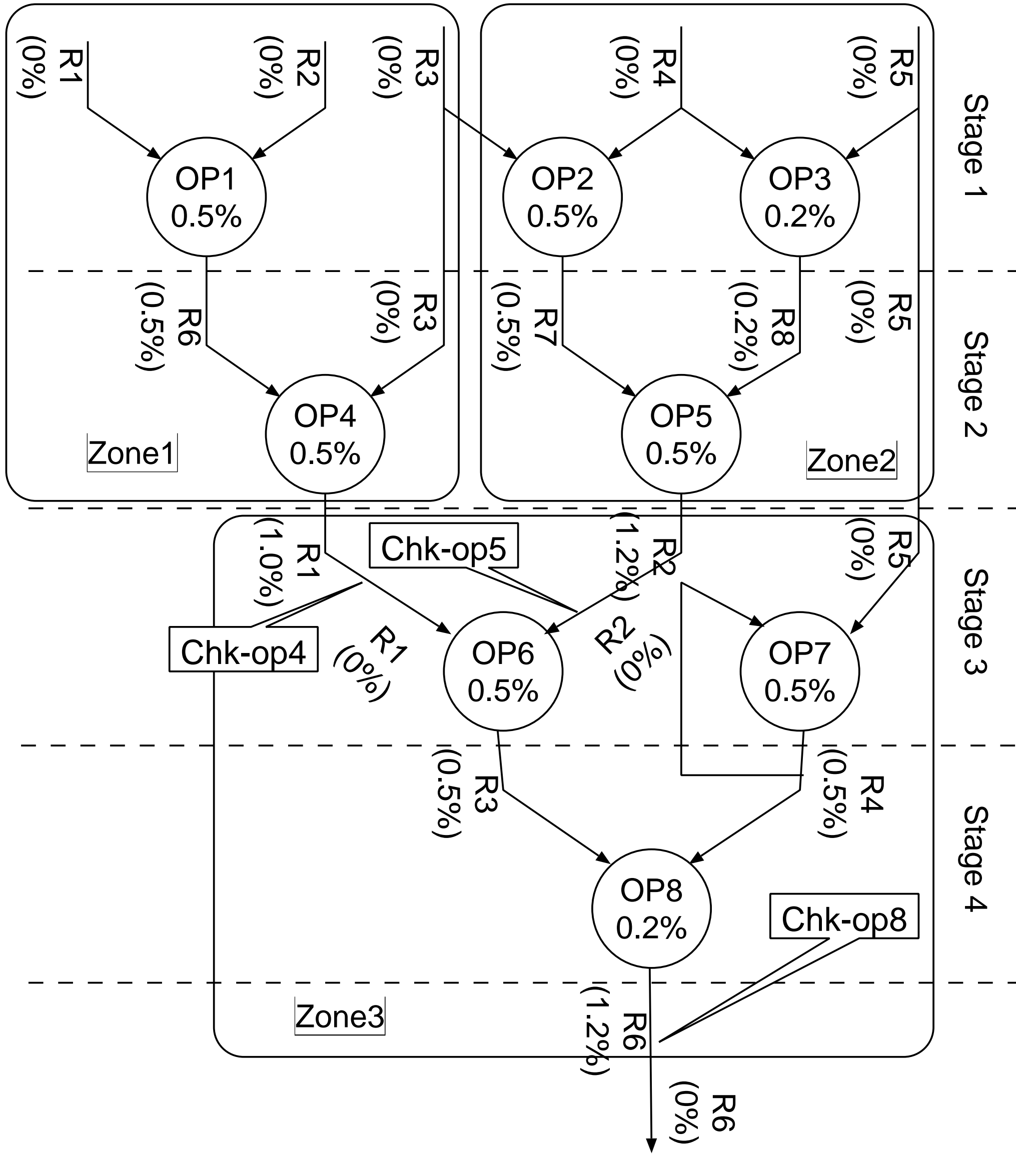


Figure 4.6. Adding check instruction on a DFG based on Eq. 4.1.

However, as `R4[1]` depends on `R4[0]`, `R5` and `OP7`, when `R5`, `OP7` are correct, the only dependency becomes `R4[0]`. For this reason, we can have:

$$\Pr(\text{R4[1] correct} \mid \text{R5}, \text{OP7 correct}) = \Pr(\text{R4[0] correct}) \tag{4.3}$$

Using equation (4.3) in equation (4.2) we can have

Table 4.2. Benchmark programs.

Functions	Number of Instructions	Number of Independent Trees	Types of Program
Expand4k	73	2	
Unsharp	63	1	
Wdifline	57	1	*A*
FI-1	40	1	
Blur	20	1	
FI-3	14	1	*B*
Tone	12	1	
FI-2	68	8	*C*

$$\begin{aligned}\mathrm{Pr(R4[2]\,correct)} &= \mathrm{Pr(R4[0]\,correct)} \times \mathrm{Pr(R5, OP7\,correct)} \\ &= \mathrm{Pr(R4[1]correct)}\end{aligned} \tag{4.4}$$

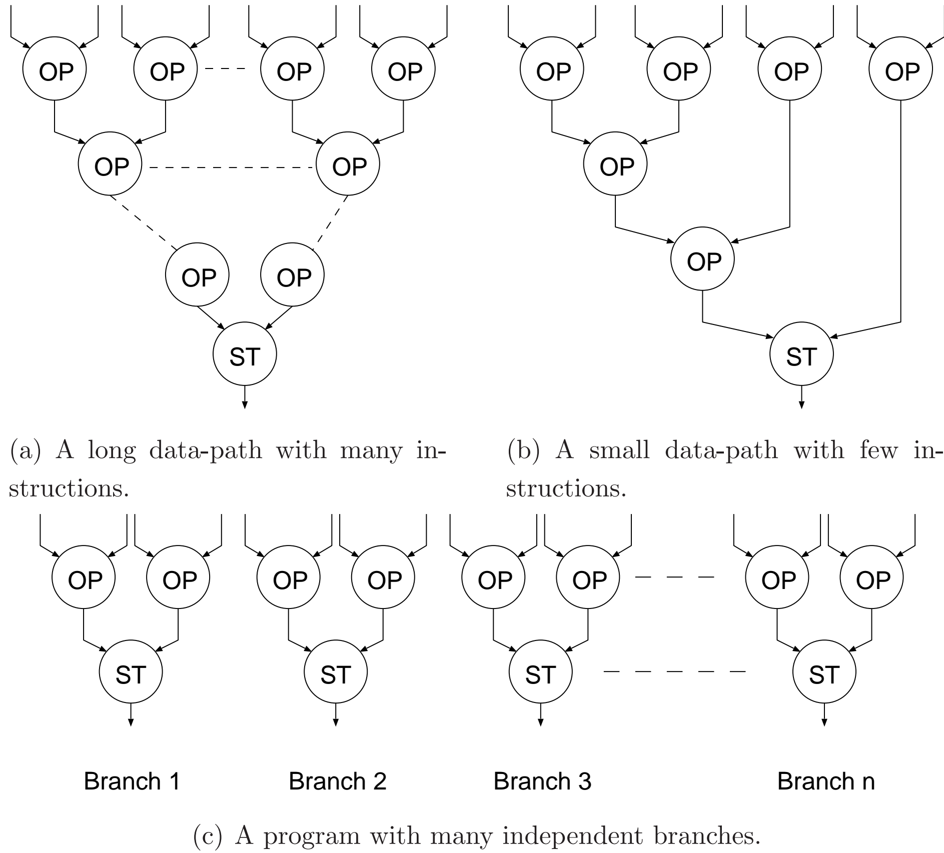
Therefore, we can expect that from the second iteration $\mathrm{Pr(R4[n])} = \mathrm{Pr(R4[n-1])}$. We thus fixed the possibility of faults in loop-back `R4` by the above means.

3. Results

3.1 Workloads and Characteristics

In this section, we present the results of our proposed technique based on ODP-aware selective checking. We tried our techniques on eight image filter functions. The size and the number of independent tree of the functions are described in Table 4.2. Fig. 4.7 show three different types of data flow graph in FU array. There is a long data graph in Fig. 4.7(a), and a small data path in Fig. 4.7(b). Both of them have a final single output to the memory. Fig. 4.7(c) gives a data graph with many independent branches, where all the branches are end up in writing results to memory. The benchmark functions are classified as Type A, Type B and Type C according to Fig. 4.7(a), 4.7(b) and 4.7(c), respectively.

We used a cycle-accurate architectural simulator [5] to get the energy data



(a) A long data-path with many instructions.

(b) A small data-path with few instructions.

(c) A program with many independent branches.

Figure 4.7. Three kinds of data flow graphs.

of the executions of the programs with the selective check instructions. The parameters of the baseline processor in the architectural simulator are listed in Table 4.3. Specifically, power data of each unit in the baseline processor has been obtained by the PrimeTime with a 180nm cell library, under the working condition of a 1.8V supply voltage. Together with the utilization of each unit, which is extracted from the simulator itself, the total energy consumption with the fine-grained power gating scheme can be obtained. Paper [19] has introduced the accuracy of this simulator, verified by the real data from a 180nm-based prototype ASIC.

As can be expected, this study mostly depends on the instruction types and their distribution. Thus, we calculate the incidence rate of the different instructions of the benchmark functions, which are shown in Fig. 4.8. We categorize

Table 4.3. Simulator specification.

Stage0	
Decode Width	max. 8 inst./cycle
General Register	32
Media Register	32
Data transfer speed (with ext. cache)	8bytes/cycle
Instruction Cache	4 ways 16 KB (64byte/line)
L1 Data Cache	4 ways 16 KB (64byte/line)
L1$ → L0$ Data Transfer Rate	16bytes/cycle
Store Buffer	4 entries
50 Stages	
Number of FUs	200 (4 FUs × 50 rows)
Instruction Mapping Speed	4 ways 256B (16bytes/cycle)
Inter L0$ Data propagation rate	16bytes/cycle
L1$→L0$ Data transfer rate	16byte/cycle
L0$→LSU Data transfer rate	4byte/cycle
Store Buffer	1 entry

the instructions by memory accesses and logic/arithmetic/media operations. As can be easily observed from Fig. 4.8, the ratio of instructions in a function varies according to the application characteristics. For example, FI-2 and FI-3 have only memory accesses and logic operations. Differently, Expand4k, Unsharp and Blur have a high ratio of media operations. According to our technique, these functions require more check instructions, since media operations have a higher level of ODP and may be more possibly turned into a defective unit.

Furthermore, we optimized the ODP of the instructions with a constant input to reduce the number of check instructions further as introduced in Section 2.3. Fig. 4.9 shows the ratio of instructions with a constant input inside the functions. Note that most of the address calculations for the memory access operations have one constant input as the offset to the base address. As an example, `LD R1,@(R10,#4)` is a memory access operation, in which, `(R10,#4)` calculates the memory address by taking an immediate input #4. Therefore, there is a rough

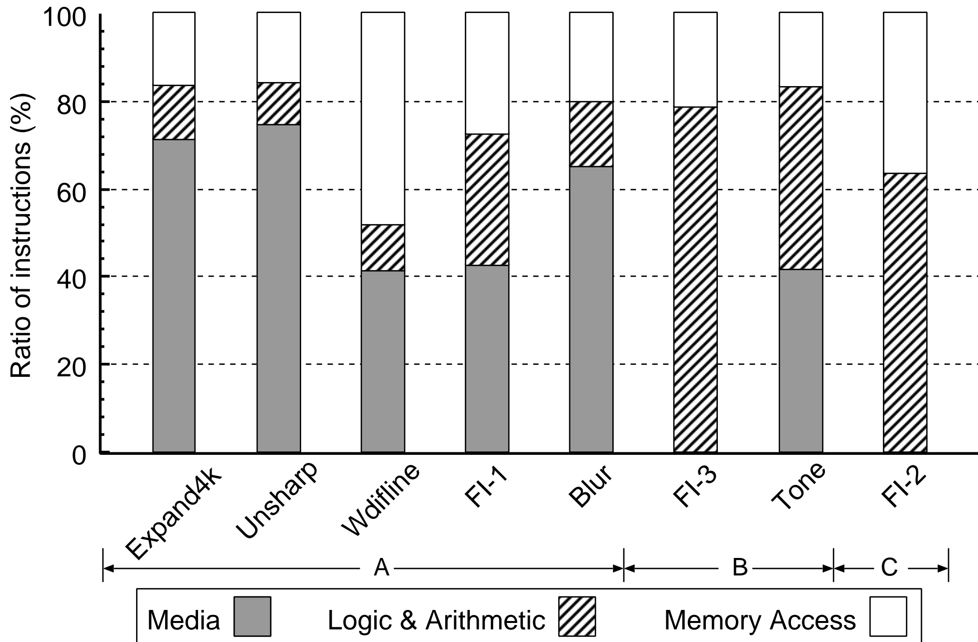


Figure 4.8. Incidence of instructions.

tendency that programs with a large portion of load operations is likely to have a higher ratio of operations with constant inputs. As an example, the difference between FI-2 and Expand4k in Fig. 4.8 and Fig. 4.9 gives a good demonstration of this tendency. Another rough observation is that media operations contribute far less constant inputs than other types. As a result, for Wdifline, the low ratio of constant inputs overwhelms the gaining of high ratio from the memory operations, which results in a medium level of fixed input ratio in this benchmark in Fig. 4.9. Overall, the average result shows that it is possible to update the ODP more precisely in 65% instructions with the insensitivity from constant values, which may have a visible increase in the accuracy in avoiding non-critical checking.

3.2 Reduction of Check Instructions

Fig. 4.10(a) gives the number of check instructions by using algorithm (Fig. 4.2) and ODPs in Table 4.1. Fig. 4.10(b) shows the results of more precisely updated ODP by taking constant inputs into account. For comparison, both figures also include the ratio of check instructions from the light-weighted technique, which

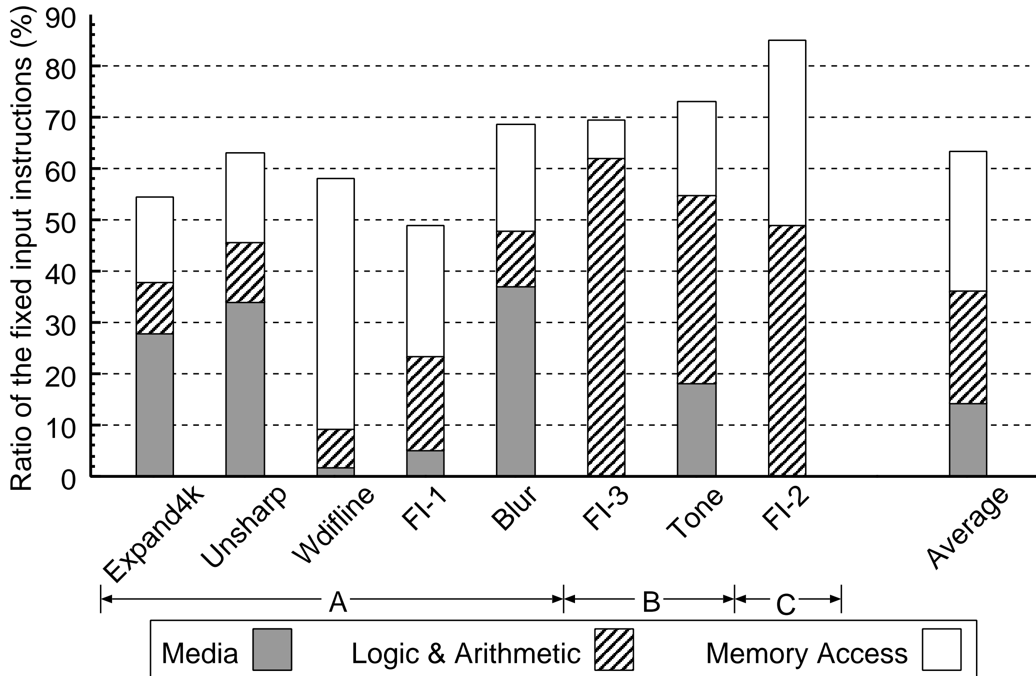


Figure 4.9. Incidence of instructions with a fixed input.

only adds the check instruction at the end of the execution-path. It can be observed from Figs. 4.10(a) and 4.10(b) that the light-weighted method can achieve the smallest number of check instructions. However, we can also find in these figures that the difference of the check instruction ratio is very small between the light-weighted method and our proposal when $\mathrm{ODP_{th}}$=10%, as compared to other $\mathrm{ODP_{th}}$ values. A detailed analysis indicates that the difference between $\mathrm{ODP_{th}}$=10% and the light-weighted method is usually within one check instruction, which helps divide the loop kernel into two branches and accordingly isolates the error propagation so as to increase the dependability. From this view, we can roughly represent the light-weighted method by taking a large $\mathrm{ODP_{th}}$ such as 10% or more. However, although these large $\mathrm{ODP_{th}}$s help reduce the check instructions largely, their corresponding reliability will be traded off, which will be discussed in detail in Section 3.3.

It can be predicted that the number of check instructions will be dominant by the ODP threshold used in our method. However, there are some other parameters that change the final insertion of selective check instructions, such as the

type of instructions, the length of the critical paths, and the number of branches inside the functions. For example, in Fig. 4.10, functions from Unsharp to FI-1 (Type A) contain a lot of media operations. A low ODP threshold 0.1% leads to about 95% selective check instructions, and they will divide the program into 1-instruction zones. However, this represents situations of impractically high error rates. With a threshold 0.5% or higher, the number of selective check instructions can be largely reduced. Specifically, for a threshold of 10%, the number of check instructions can be averagely reduced to a 7% level, in which the added check instructions segment the data path into 5 or 6 zones, each contains 17% instructions.

Secondly, programs with a short critical path (Type B), as Blur, FI-3 and Tone, behave differently from the programs in Type A. The increase ratio of the check instructions is high for low ODP thresholds. However, for the thresholds over 5%, no additional check instructions are required other than checking the final result. On average, these programs have 15% check instructions, and for large thresholds, only the check instruction for the final result covers the whole data path.

Finally, the Type C function, FI-2, which contains many independent data paths, also behaves in the same way as the Type B function. In FI-2, there is one store per each independent data branch, which has been designed to contain a built-in check. These in-store checks have already segmented the whole data-path into small zones, which do not eagerly require additional checks to lower the criticality of the ODP accumulation. For this reason, the number of check instructions stops growing at $\mathrm{ODP_{th}} = 5\%$. The total number of check instructions, including the in-store checks, remains at 8% even when $\mathrm{ODP_{th}} = 10\%$.

According to the Figs. 4.10(a) and 4.10(b), the number of check instructions differs by taking or not taking the influence from the constant input into account. The maximum difference can be found when $\mathrm{ODP_{th}}$=0.5%, averagely. Furthermore, studying the individual programs, we can have the following detailed observations:

1. Under $\mathrm{ODP_{th}}$=0.5%, benchmark programs Unsharp, Wdifline, Blur, Tone and FI-2, show large differences by using the insensitivity from the fixed input in calculating ODP. These benchmark programs contribute to most of

the reductions from Figs. 4.10(a) to 4.10(b), which is 13.2% under ODP_{th}=0.5%. This may come from their relatively high ratios of constant input operations. However, in other than ODP_{th}=0.5%, minor changes can be found between Figs. 4.10(a) and 4.10(b) for these benchmarks.

2. Programs Expand4k and FI-1, show very minor changes under most ODP_{th}s.

3. The best ODP_{th} to distinguish Figs. 4.10(a) and 4.10(b) for benchmark *FI-3* is ODP_{th}=1.0%.

The above observations can be connected to the combination of the program characteristics such as the ratio of the operations with a fixed input, the distribution of operations with a fixed input among all the operation types, and the weight of all input operation types. For example, Expand4k and FI-1 have a small number of operations with a fixed input, they therefore have less changes after taking the fixed input into account. FI-3 has a medium ratio of operations with a fixed input. However, it has a relatively short data-path, which is 14 operations in Table 4.2. In addition, the 14 operations in FI-3 are mainly the logic and arithmetic ones, which have a simpler distribution as the other benchmarks. Therefore, FI-3 shows some differences between Figs. 4.10(a) and 4.10(b) under ODP_{th}=1.0%, which is slightly different from other benchmarks.

3.3 Dependability and Energy Savings

By adding different numbers of check instructions, we are changing the dependability of the data-path by preventing the tainted data from the defective unit propagating inside the data path. The data propagation is usually safe when every instruction is duplicated and checked at the output point of the data-path. However, a prior research [12] on hard error in a data-center also states that the hard error rate will be very high for the servers that have previously experienced a hard fault. Therefore, it is possible that during the propagation of the tainted data, the dependability will go unpredictable when facing a second error before the tainted data is detected. For this reason, we use the metric of error propagation distance to measure the dependability of the data-path. The propagation distance is defined as the delay between the error generation node and the detection node inside the DFG. For a 10-node single branch data-graph, if we have

only one check instruction at the end, the propagation distance will be 9 if an error occurs in the first node.

Accordingly, the reliability is calculated in term of vulnerability to a second error, as follow:

$$E = \sum_{i=1}^{n} P_i \times D_i \tag{4.5}$$

In Eq. 4.5, n is the number of instructions in a segment. This segment refers to a DFG segment between two check instructions added according to the ODP_{th} in this research. P_i is the ODP of an instruction and D is the error propagation distance. According to Eq. 4.5, the vulnerability to a second error is 0% when all the instructions are protected by a check instruction, as the distance D_i is 0. With the decreasing number of check instructions under an increasing ODP threshold, the vulnerability to a second error, as E in Eq. 4.5, will increase.

The reliability of our proposed technique with considering and without considering the fixed input is depicted in Fig. 4.11. Fig. 4.11 clearly shows that the propagation distance decreases sharply from the light-weighted method to ODP_{th}=10% and then to ODP_{th}=5.0%, which indicates that these large ODP_{th}s may suffer more from the occurrence of a second error when the first error is still inside the data-path before detection. However, the decrease of error propagation distance goes rapidly flat when ODP_{th} corsses 1.0%, indicating a saturation in the check instructions. This also matches the expectation that the dependability increase will be exponentially difficult after it reaches a certain level so that balancing cost and efficiency is necessary. In addition, from another view, at ODP_{th}=1.0%, when a 1.0% longer error propagating distance is allowed, the check instructions can be further reduced to a 66% level.

Studying the influence of applying constant inputs in ODP, it can be easily observed that two lines are almost overlapping each other, which indicates that applying the insensitivity of fixed input to decrease ODP does not hurt the reliability. There is almost no difference between ODP_{th}=0.5% and ODP_{th}=0.1%, and for ODP_{th}=0.5%, 39% check instructions can be saved while the influence of the fixed input are not considered. There are 52% check instructions that can be removed from the DFG by considering the influence of fixed inputs for the same ODP_{th}. Another observation, at ODP_{th}=0.1%, is that when a 1.0% longer error

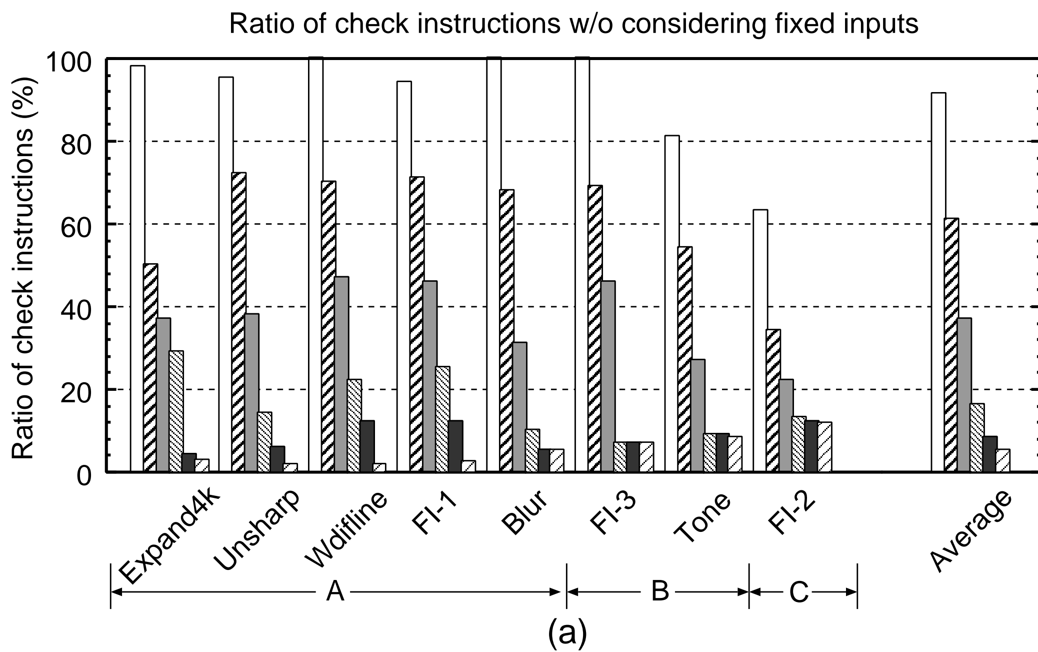
propagating distance is allowed, the check instructions can be further reduced to the 66% level.

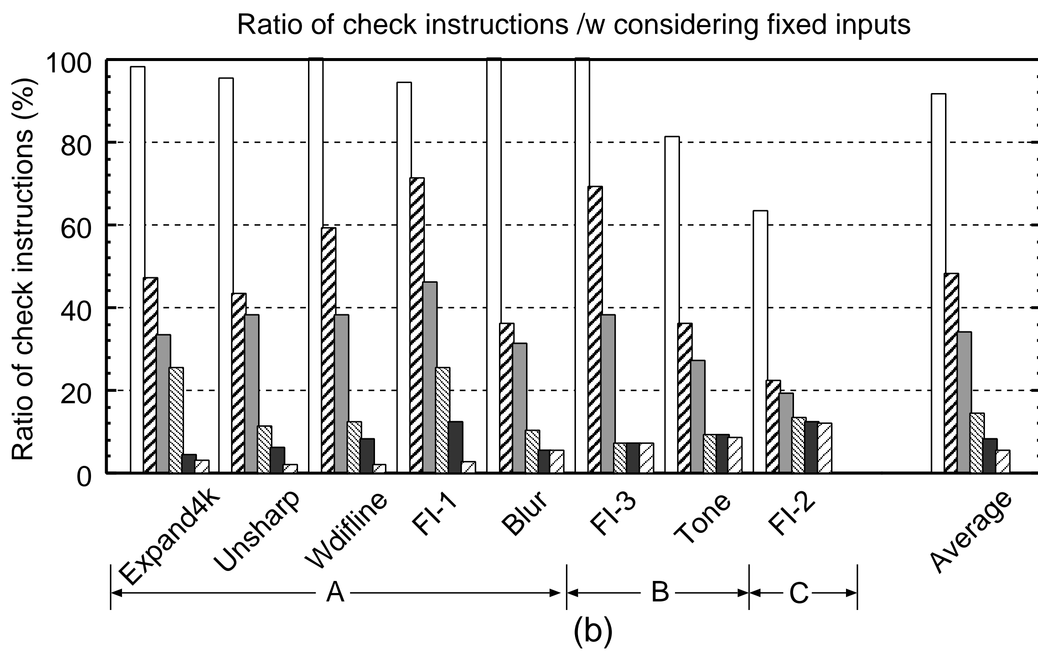
Fig. 4.12 gives the energy saving results by the proposed method, as normalized by the energy of the original exhaustive checking method. Fig. 4.12 also shows that we can save more energy by means of optimized ODP than the normal ODP especially for ODP_{th}=0.5%. Combining with the results in Fig. 4.11, we can achieve the same near-optimal reliability at $ODP_{th} = 0.5\%$ $ODP_{th} = 0.1\%$ by saving 17% more energy. If 1.0% downgrading of reliability is allowed, further 22% energy reduction is possible by using $ODP_{th} = 1.0\%$.

4. Summary and Discussion

In this dissertation, we have presented a technique to remove check instructions from non-critical positions to avoid an exhaustive checking for fault-tolerable execution. The method can efficiently work with a reconfigurable FU array architecture to achieve high dependability with awareness of the fault possibility. In our approach, check instructions are added selectively according to the error probability (ODP) along the data path when the accumulated possibility exceeds a threshold. In addition, we also studied the influence of constant inputs as they can turn parts of the circuit into don't care zones and therefore help reduce the sensitivity to the faults.

Our study of the dependability of the updated data-path has shown that the reliability can still be kept at a near-optimal level when properly removing 52% non-critical check operations. This results in an energy saving of 17% for high dependable execution. With an allowance of downgrading 1.0% reliability, it is possible to reduce the energy further by 22%. In summary, a cost-effective high dependability method has been achieved by using our ODP metric in the FU array processor.





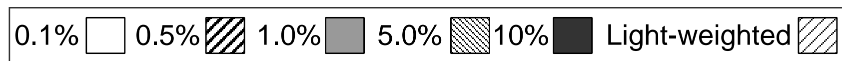


Figure 4.10. Ratio of check instructions for different thresholds.

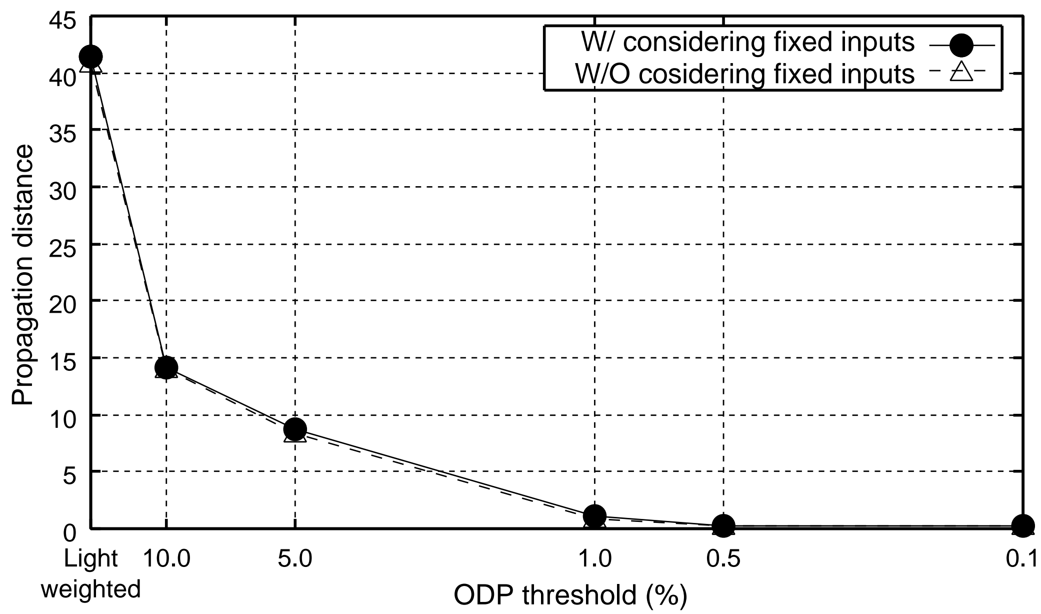


Figure 4.11. Vulnerability to the second error for different check instructions.

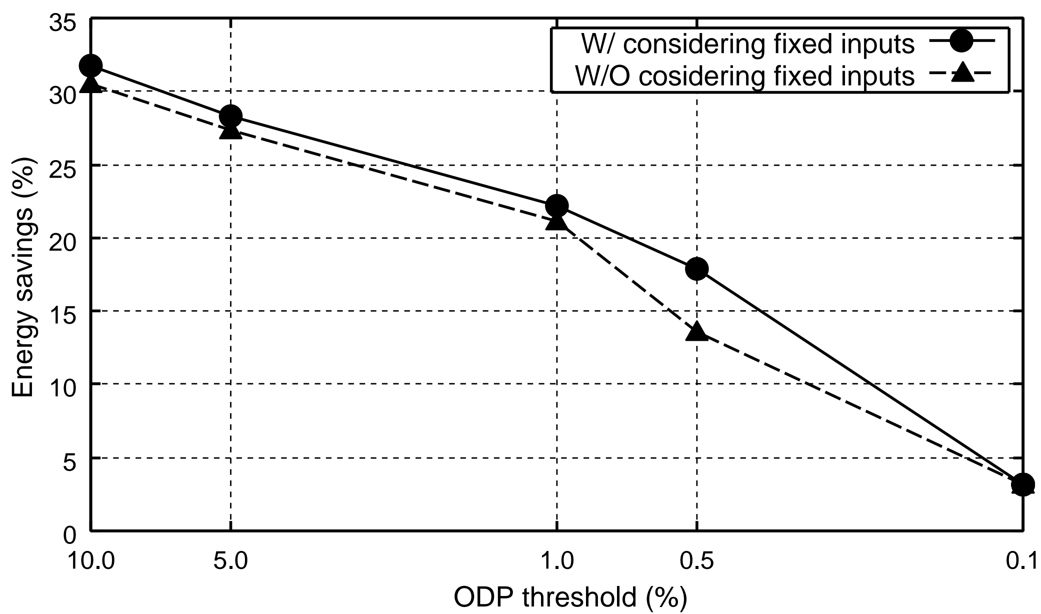


Figure 4.12. Energy savings for 10M executions.

Chapter 5

Improvement of Robustness of Partial Redundancy by Using SDC Prediction

Technique to improve the robustness of partial redundancy by using SDC prediction is discussed in this chapter. Results and discussion of this technique are also included.

1. Introduction

Fully duplicated execution of either the temporal [18] or the spatial [16, 30] redundancy can provide a good fault-tolerance for SEEs. However, the increased chip area, execution time and energy consumption due to the full duplication are usually not preferred in most circumstances, since they interfere with power-efficiency, and power efficiency is the most important requirement of the market. Many techniques have already been used to exploit possible trade-offs between efficiency and dependability. A well-used method is to categorize the data into zones of different importance and then treat them accordingly. The ERSA method [13] has demonstrated a way to put a different fault tolerance on the control and the data paths in an image processing application. For example, the program characteristics in image processing applications can be abstracted into loops, as shown in Fig. 5.2(a), where the loop index controls all pixels calculations

and is therefore regarded to be more important than the pixel data. The corruption of the loop index usually leads to the abnormal end of the program and thus needs to be fully covered by error-resilience techniques. The other data including processed pixels can tolerate a limited number of errors. The partial redundancy can thus use an imperfect but low-cost fault-tolerance. Fig. 5.2(b) demonstrates this idea of partial redundancy by only adding a duplicate execution and a check on the loop index.

Apart from image processing, applications based on approximate calculations such as decision making from probability exploration, machine learning, computer vision, and so on, may similarly apply different dependability/cost methods for their various types of internal data [13].

However, as the calculations and data inside the loop body are not covered by any redundancy (Fig. 5.2(b)), they may suffer from silent data corruptions (SDCs), especially in a worst-case situation. Fig. 5.1 gives a general description of this problem. The full redundancy, i.e. dual modular redundancy, can detect and address every SEE, while its cost is, however, doubled due to the duplication. The partial redundancy cost is minor, since it only covers the most important data such as the control instructions of the program. Under a low SEE rate, the correct execution of the control flow can provide a sharp increase in the dependability, as compared to the non-redundancy execution, in which the execution may abort before completion if SEE hits the control flow. However, when the fault rate increases, SEEs are more likely to accumulate inside the unprotected loop body execution zones and then become SDCs. The relatively large code size of the loop body will increase the SDC rate, and thus lower visibly the dependability, as is seen in Fig. 5.1.

In order to solve the above problems, we propose a method to achieve an improved dependability by calibrating the past executions and predicting the future error rate. A small test program is executed along with the program data-flow-graph (DFG) to sample the error rate. A strategy unit is then used to provide a bias between the partial and the full redundancies based on on-the-fly error monitoring of SEE data, together with the instructive program characteristics of the coming program interval. Under a very high fault injection rate with 10 errors being injected into our baseline microprocessor per second, the SDCs of

the traditional partial redundancy is 12%. By our method, it has been reduced to 0.37%, and a two-order improvement in the error rate, with an additional 8% energy cost.

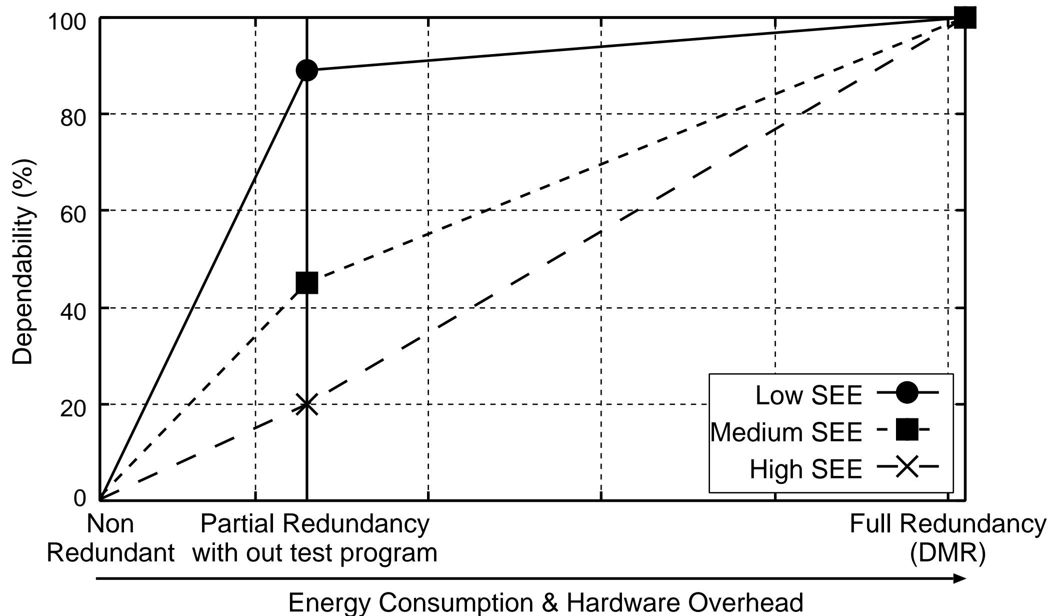


Figure 5.1. Dependability of the traditional partial redundancy under different SEEs.

2. Proposed Technique

2.1 Outline of the Proposal

In this work, our target is to improve the dependability of partial redundancy for all kinds of SEE rates, while maintaining its low cost. Our approach is built on a sampling and prediction of the coming SDC rate. Partial redundancy is still the main working mode in our system. Additionally, for the purpose of robustness, we add a small full-redundant portion as a supporting mode, at the necessary points indicated by the SDC prediction result.

Fig. 5.3 shows the basic flow of the proposed method. For the error sampling we added instruction blocks with a self error detection ability along with the

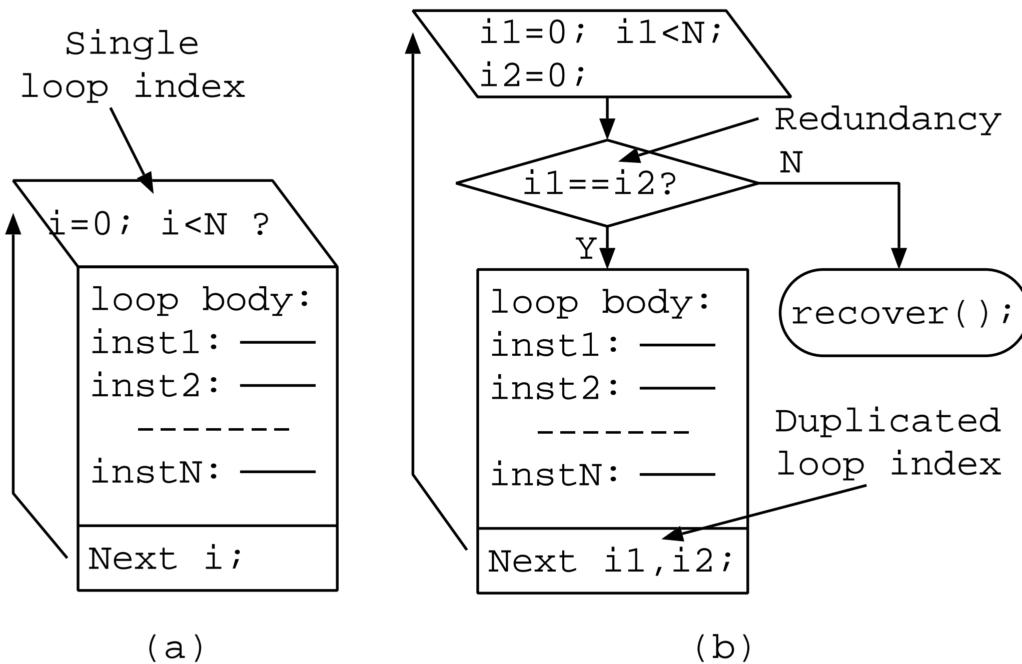


Figure 5.2. Partial redundancy example: (a) A non-redundant loop execution, (b) Partially redundant loop execution by explicitly duplicating the loop index calculation.

program execution to keep a sample of the fault rate. In this work, we used a test program originated from the XOR operation: `C=XOR(A, B); D=XOR(A, C); e_flag = (B!=D);`. When the two XOR operations are correctly executed, the values of `B` and `D` will be identical. The `e_flag` thus indicates that an SEE has struck on any one of the three operations. Note that this error detection is independent of the executing programs. The test program blocks will be inserted into the program execution at a certain frequency. As shown in Fig. 5.3, during period `t1` to `t2`, this test instruction block continuously collects the fault occurrences during program execution. Detected error occurrences are accumulated in a counter (`err_counter`, in Fig. 5.3), and later used as sampling values of the fault rate.

The sampled fault rate will be used to predict the potential SDC rate, based on information from the next program to be executed. The loop execution, as shown in Fig. 5.2, is the typical program behavior that we explored to use for our SDC

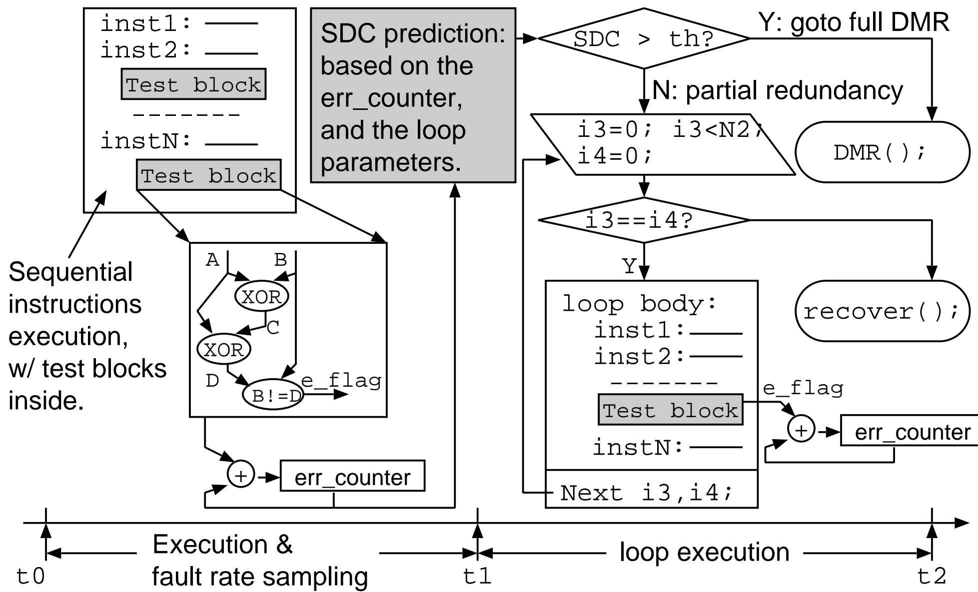


Figure 5.3. Proposed partial redundancy.

prediction approach. Because most loops usually contain a predefined number of iterations, given by the maximum loop index value, and an in-loop instruction number, it is relatively easy to estimate the possible error rate in the program about to be executed. Thus, before starting the loop at time `t3` (Fig. 5.3), the processor will make an SDC rate prediction based on the current fault rate, indicated by the error counter, and the loop parameters. The mechanism of this prediction will be introduced in detail in the next section. If the SDC rate reaches a predefined threshold `th`, the loop execution will select full redundancy, indicated in Fig. 5.3 as function `DMR()` where every instruction is duplicated and the values from the two copies are compared at the end of the loop body graph. Otherwise, the partial redundancy approach, which only duplicates the loop index execution (Fig. 5.2(b)), will still be applied to save energy cost.

2.2 SDC Rate Prediction

Since we are predicting the SDC rate for the coming loop execution with current fault rate, the following equation is used.

$$Pr_{SDC} = [1 - (2 \times err)]^N [1 - (1 - I \times err)^N] \quad (5.1)$$

In Eq. 5.1, I is the number of instructions inside a loop, excluding the loop index computation, and N is the number of loop iterations. *err* is the sampled error rate, calculated from `err_counter` in Fig. 5.3. *err* represents the accumulated number of errors detected by the test blocks in each execution interval. As the partial redundancy approach itself has already considered the error check inside the duplicated loop index execution, the predicted SDC rate should only indicate the possibility of errors that escape the loop index checks and that strike on the data execution in all of the N loop iterations. Therefore, Eq. 5.1 is designed to follow Bayes' theorem. It contains the following parts:

1) $[1-(2\times err)]^N$: Here, *err* gives the cycle-level per unit-sampled error rate, calculated from `err_count` in Fig. 5.3. This part of Eq. 5.1 gives the possibility of correct execution for all the N loop indices by assuming that the error rate is equally distributed inside the processor. The value "2" refers to the two instructions assigned to the loop index. A very high *err* leads to a small $[1-(2\times err)]^N$, which implies that errors can be captured with high possibility inside the duplicated N loop indices, so that the probability of SDC decreases accordingly.

2) $[1 - (1 - I \times err)^N]$: This part is related to the rate of SDCs in the loop body. $(1 - I \times err)^N$ is the possibility of correct execution of the remaining loop body, for all the N iterations. The $[1-(1-I\times err)^N]$ is then given the expected SDC rate during the loop body calculation, which has I instructions. This part has a tendency to increase when *err* increases, which increases the chances of the processor using full redundancy by our baseline algorithm.

Fig. 5.4 shows the change in estimation of Pr_{SDC}, as a function of `err_count` by plotting Eq. 5.1. In Fig. 5.4, the Pr_{SDC} grows normally until point (A), at which point the fault struck the test program. However, after point (A), the Pr_{SDC} decreases due to the decreased $[1 - (2 \times err)]^N$, which means that, to detect the error, we can rely more on the loop index check than on the test program. In Fig. 5.4, the horizontal line "L", which represents the threshold, intersects with the estimated error curve at two points, denoted respectively as (a) and (b). As shown in Fig. 5.3, when the predicted Pr_{SDC} is larger than a predefined threshold `th`, a full redundancy will be scheduled. Therefore, full

redundancy will be applied to the region between (a) and (b) in Fig. 5.4 by using the SDC prediction. The region below (a) and beyond (b) will be covered by the partial redundancy approach.

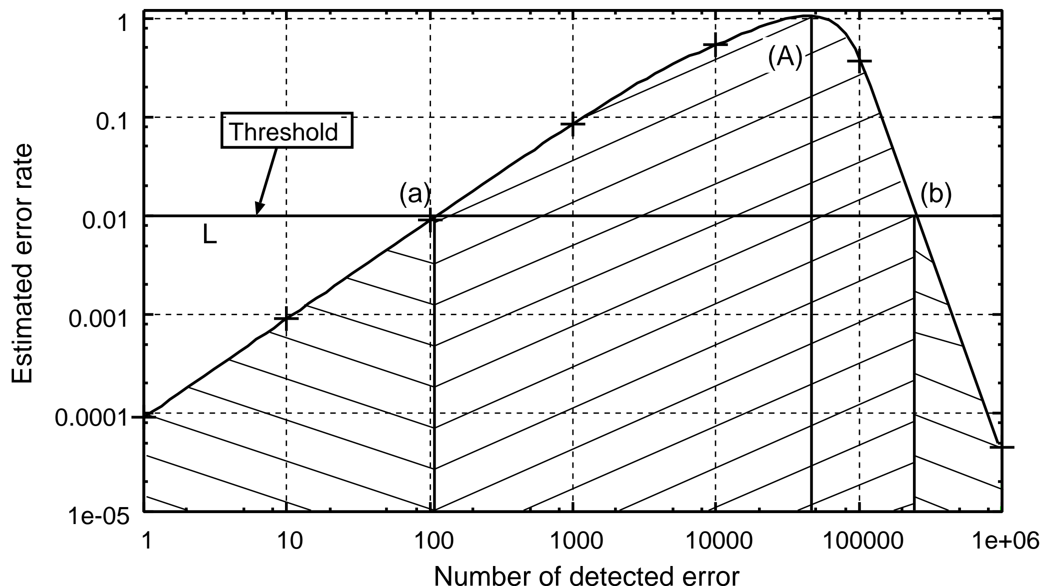


Figure 5.4. Our predicted *SDC* rate as a function of `err_count`.

2.3 Effectiveness of using SDC Prediction

Fig. 5.5(a) and Fig. 5.5(b) show the scenarios of SDC reductions before and after applying our prediction mechanism to the original partial redundancy approach. In the two tables, we use `Y` and `N`, respectively, to represent whether or not there is a fault attack given by SEE. We also use `+` and `-` to denote whether the error detection scheme is able or not able to figure out the fault attack. Therefore, `Y(+)` represents a successful detection of SEE and `N(-)` also gives a good and safe judgment. `Y(-)` is the scenario in which a fault occurs without detection, and which thus becomes an escaped error.

As shown in Fig. 5.5(a), in the traditional partial redundancy, when there is a fault strike on the loop index, it is detected, and the loop is then restarted for the processor to recover from the erroneous state. Any fault striking on the loop body will also be covered by this restart, so that any such fault is marked “Don’t

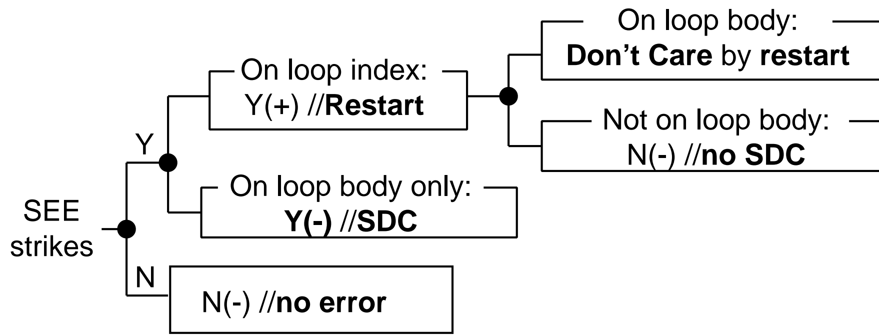
Care". When the SEE only strikes on the loop body, the fault will escape the loop index check and then lead to a `Y(-)`, which then becomes an SDC.

Fig. 5.5(b) gives the scenarios of our proposed partial redundancy. In this figure, unlike Fig. 5.5(a), with an extra SDC prediction we are able to get additional chances for checking by using the approach seen in Fig. 5.3 and in Eq. 5.1. For example, faults that hit the test program will be detected, and a full redundancy, such as DMR, will be used when Pr_{SDC} reaches the `th`. The SEEs on the loop body will be addressed by DMR execution, which leads to a true-positive `Y(+)` prediction scenario and reduces SDCs. An inaccurate (denoted as false-positive) prediction occurs when there are actually no SEEs hitting on the loop body after the DMR is scheduled. This leads to some energy consumption loss while the execution is still safe.

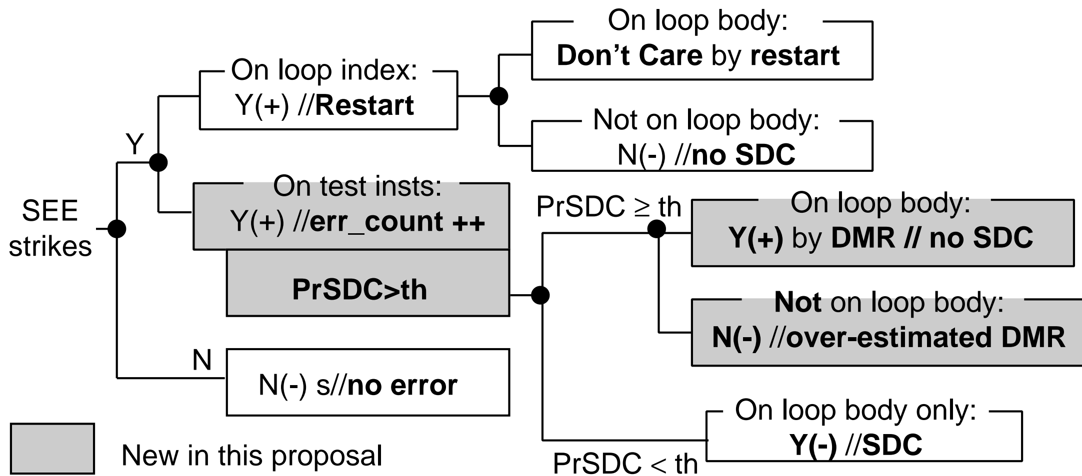
3. Experiment setting: assuming error rates from a radiation stress test

3.1 Error rates

In our research, high, medium, and low error injection rates are applied in our workload simulation. As the purpose of the research is mainly to address SEEs, we used a test chain of 1,080 Flip-Flops (FFs) to obtain the SEE rate under an accelerated radiation. The irradiation test is performed by applying an alpha particle source, 3 MBq 241Am, directly to the test FF chain in a bare chip. The chip was manufactured with 180nm technology, which was originally designed to work at 1.8V supply voltage. We applied, however, a 1.25V supply voltage to accelerate the fault occurrence. A detailed discussion about the experimental setup can be found in [30]. Our experiment has shown that the average erroneous flipping rate in the FF chain is in the order of 10^{-4} flips/second. A preliminary implementation of our baseline processor indicates that the number of unprotected FFs reaches roughly the 10^5-bit level. When the processor works under a 1GHz frequency, the fault rate in each cycle is 10^{-8} flips, which is then adopted as our high error injection rate. Hence, we apply 10^{-10} and 10^{-12} flips/cycle as the medium and low error injection rates, respectively.



(a) Error detection in partial redundancy



(b) Error detection in proposed partial redundancy

Figure 5.5. Improved error detectability by using Pr_{SDC}.

In addition, proton radiation has been tested on an IBM Power6 processor [21], and the results showed that there are ∼1,748 flips in 2 days, while the operating frequency of the processor is 5GHz, and the number of transistors is 790 million. This processor was developed by 65nm technology. Also, a neutron test on 90nm CMOS technology [11] has shown that the fault rate increases at a 1.18x exponential speed for an 10% reduction in voltage, and 8% increment in the fault rate per generation. Based on the above results, then, SEE has been calculated for our platform, and the calculated SEE for a neutron radiation is 10^{-14}. This value is lower than the low SEE injection rate in our setting. From our results

in latter parts, we assume the 10^{-12} SEE rate can represent this 10^{-14} rate from the proton radiation.

3.2 Baseline platform and simulator settings

We use a cycle-accurate simulator, which abstracts the behavior of our reconfigurable architecture to get the performance and energy consumption data. Table 5.1 lists the parameters in the simulator. Specifically, the FU array in the reconfigurable architecture takes 80 rows, which is designed to be sufficient to map the full DMR execution the DFGs we studied. Besides the 4 FUs in each row, an L0 buffer with 4 entries is embedded in each FU array row to help the fast load/store operations from/to the L1 cache. Both the L1 instruction cache and the data cache are defined to be of 4-way 16KB storage size [31].

In this study, we tried our proposed technique on seven image filter functions, as listed in Table 5.2. The purpose of each individual loop and the number of instructions inside the loop body of each are also given in Table 5.2. Note, for example, that Edge, Tone and Blur have fewer than 20 instructions, while Wdifline, Unsharp, FI and Expand4k are relatively large loop bodies with more than 40 instructions. Their behaviors are thus expected to differ even under the same error injection rate, and may therefore respond differently under both the traditional partial redundancy scheme and our biased partial/full redundancy system.

4. Results

4.1 Overall Reliability Vs. Energy Consumption

The error rate data in Table 5.3 are the average numbers of erroneously executed iterations among our workloads under a non-redundancy, original partial redundancy and our SDC-prediction based approaches. Applied to image processing applications, these rates are related to pixel errors, assuming that each iteration processes one pixel.

As shown in Table 5.3, under our SEE injection rates, the non-redundancy mode does not work well, since there are abnormal program ends due to the incor-

Table 5.1. Evaluation setup

Simulator	
Cycle accurate	Yes, f=1GHz
Number of FUs	320 (4 FUs $\times$ 80 rows)
L0 data cache size	4 entries per row
L1 data cache Size	4 ways 16KB (64bytes/line)
L1 Instruction Cache size	4 ways 16KB (64bytes/line)
Error Monitoring Support	
sampling interval	10^{12} cycles
Moving window size	10^{14} cycles
Error Injection Ratio (bit-flip per cycle)	
Low SEE	10^{-12}
Medium SEE	10^{-10}
High SEE	10^{-8}

rectly processed loop indices. This indicates that, even for a low error injection rate, unprotected execution will result in a large loss of data correctness due to the occurrence of critical errors in the program control parts.

The above situation can be largely alleviated by the traditional partial redundancy method. Adding a simple protection of the loop index, we can observe that the error rate is lowered to an order similar to that of the SEE injection rate. Overall, the partial redundancy in Table 5.3 shows that it is able to achieve the 1.24×10^{-7}% error_rate, even under a high fault injection rate. The difference between non-redundancy and traditional partial redundancy is huge. However, Table 5.3 also shows that the erroneous iteration rate increases linearly when more SEEs are injected under the traditional partial redundancy execution. At a high SEE injection rate, the error_rate reaches a 1.24×10^{-7}% level, which means that, for about one second of execution under 1GHz, there may be about one erroneous loop.

By using our error prediction technique, the proposed method has shown its ability to reduce SDCs. One significant improvement is that the difference between the adjacent error_rate values increase more slowly than the error injection rate. For example, under the low injection rate, there is about 30% reduction of

Table 5.2. Benchmark functions

Functions	Usage	Number of operations
Edge	Edge detection	10
Tone	Stereo Matching	12
Blur	Blurring filter	19
Wdifline	Media filter	42
Unsharp	Sharpness adjustment	59
FI	Frame Interpolation	68
Expand4k	Image Expansion	73

Table 5.3. Erroneous loop iterations of different redundant technique

Performance Parameter	Redundancy Technique	Error Injection Ratio		
		Low (10^{-12})	Medium (10^{-10})	High (10^{-8})
Error_rate (%)	Non-redundancy	2.86	4.88	8.65
	Partial redundancy	1.50×10^{-11}	1.23×10^{-9}	1.24×10^{-7}
	Proposed	1.05×10^{-11}	3.68×10^{-10}	3.72×10^{-9}
Energy (%)	Non-redundancy	—	—	—
	Partial redundancy	103.8	104.1	106.2
	Proposed	109.1	111.3	114.3

erroneous iterations when using our error prediction, compared to the traditional partial redundancy. Under the high error rate, the erroneous iteration rate of our method is only 3% of the traditional simple partial redundancy. This indicates that even under a very high error injection rate, the occurrence of each loop iteration with SDC is delayed from 1 second to 30 seconds by our scheme. The quality of service (QoS) of the image processing like programs is thus greatly improved.

The average energy consumption results are shown in Table 5.3, where energy consumption is normalized by the energy consumption of the non-redundant execution when working without error injection. Energy consumption of the non-redundant execution with error injection are not listed, as their early aborts make the data less reliable. The energy increase under the redundancy execution comes from two parts: (1) the power used in the duplication and comparison, and

(2) the time delay when restarting is required after an error is detected. Overall, Table 5.3 shows that partial redundancy required additional 6.2% energy then the non-redundancy mode. The result shows that, for high SEE it has 8% additional energy consumption than traditional partial redundancy.

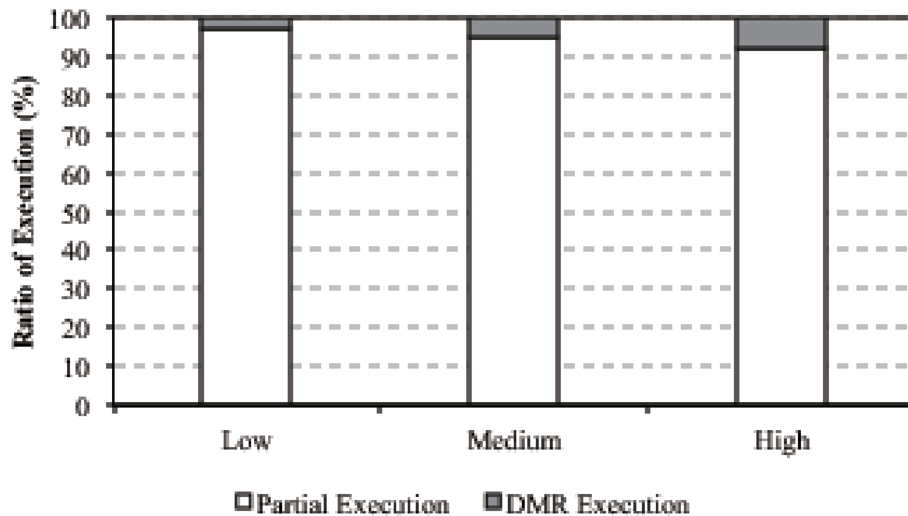


Figure 5.6. Breakdown energy consumption under different SEE rate.

The increase in the energy consumption from the original partial redundancy to our proposal is mainly from the scheduled DMR execution when the sampled SEE rate by Eq. 5.1 exceeds the threshold. Fig. 5.6 shows the breakdown of the partial redundant and DMR executions under our proposal method. It can be observed from Fig. 5.6 that the ratio of DMR execution is increasing with the error rate. For example, DMR execution for low SEE is 3%, and 8% for high SEE. This increase of DMR execution is however far slower than the increase of the SEE injection rate.

4.2 SDCs in individual program

Fig. 5.7 gives the in-loop SDC ratio under both original partial redundancy and our method for individual benchmark loops. The ratio varies among different programs, since small loops tend to dodge SEE injections when the SEEs get

more chances to hit the unmapped units. Accordingly, the SDC rates of the traditional partial redundancy vary visibly, from less than 5% to near 30% among the benchmarks under the low SEE injection. A similar situation can be found under the medium and high SEE injections. However, the different SDC rates are still in the same order under the partial redundancy. According to the analysis based on the average data in Table 5.3, the false-positive alarm has the ability of reducing SDCs at a super linear speed. Therefore, the difference between individual benchmarks decreases after applying our method, as is proved by the data in Fig. 5.7. From another viewpoint, our method reduces a larger number of SDCs for large SDC zones than for the small SDC zones, which makes the final SDC ratios less sensitive to the program characteristics.

4.3 Number of Test Blocks

The above results have been achieved by using a test program in addition to the loop index check to provide more detection ability. Fig. 5.8 explores the ideal number of test programs that can help the program achieve the best dependability within the partial redundancy framework. Accordingly, the lines in Fig. 5.8 give the dependability of test programs 1, 2 and 3 under all the error injection rates. The point at zero in the figure indicates the traditional partial redundancy, which contains only the loop index check. It can be easily observed from the line shapes that under each injection rate, adding a test program has saturated the reduction of erroneous executions. This indicates that the balancing point has been reached. This fits with our original expectation that dependability in the partial redundancy can be improved rapidly for a very small cost, while further improvement in dependability will become exponentially more difficult.

5. Summary and Discussion

In this dissertation, we have presented a technique to achieve a balance between dependability and cost by applying error-prediction based policies to a partial redundant system. The level of redundancy of a program is adaptively monitored according to SDC rate prediction, based on a sampled fault rate and a program behavior analysis. Furthermore, by adding one self-contained test instructions,

we achieved a super linear SDC reduction superior to the traditional partial redundancy method. Overall, our method reduces 12% to 0.37% the SDC ratio of the original partial redundancy, reaching a two-order increase in dependability even under a very high error injection rate of 10^{-8} flips per cycle.

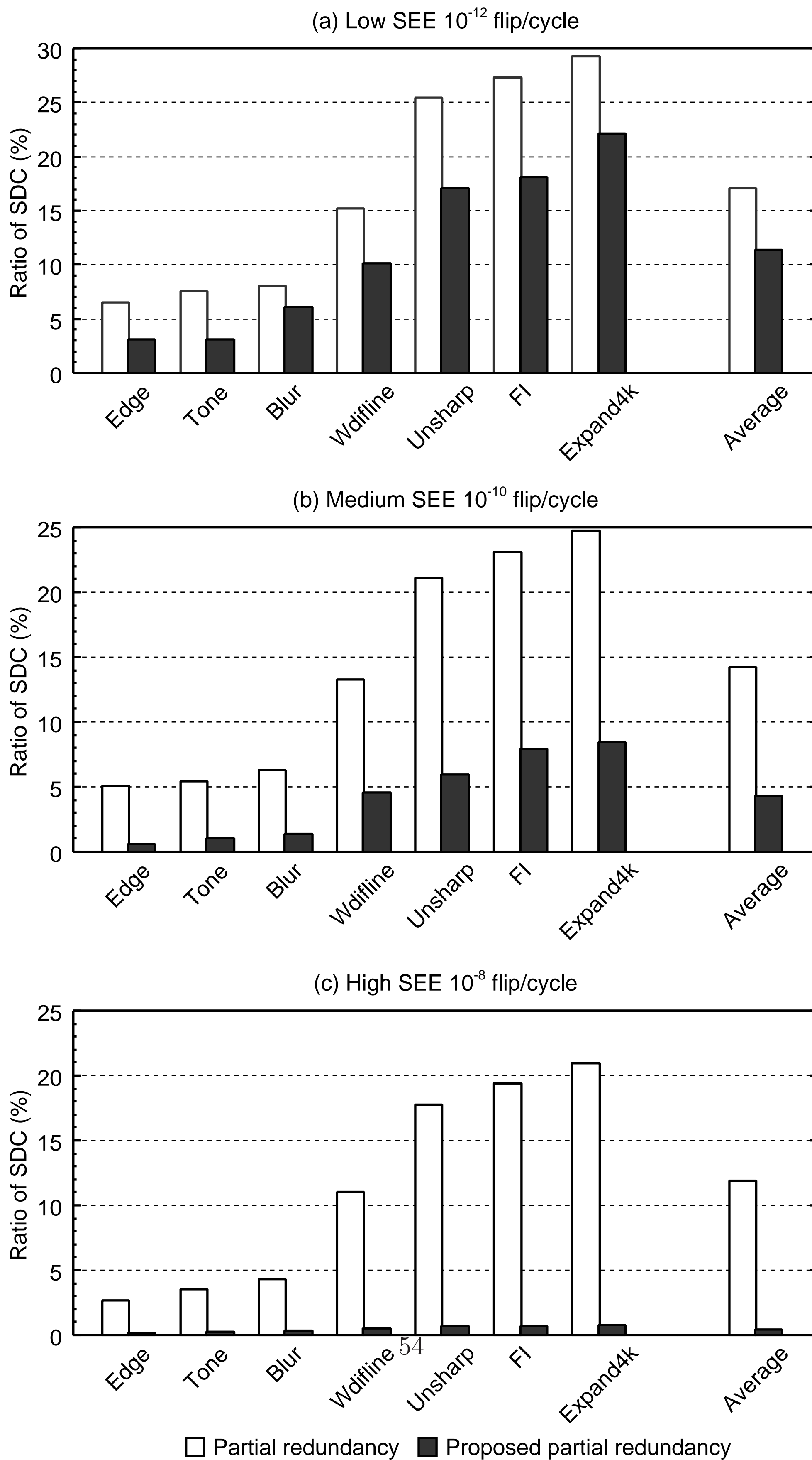


Figure 5.7. SDC reduction in individual benchmarks from original to error prediction based partial redundancy.

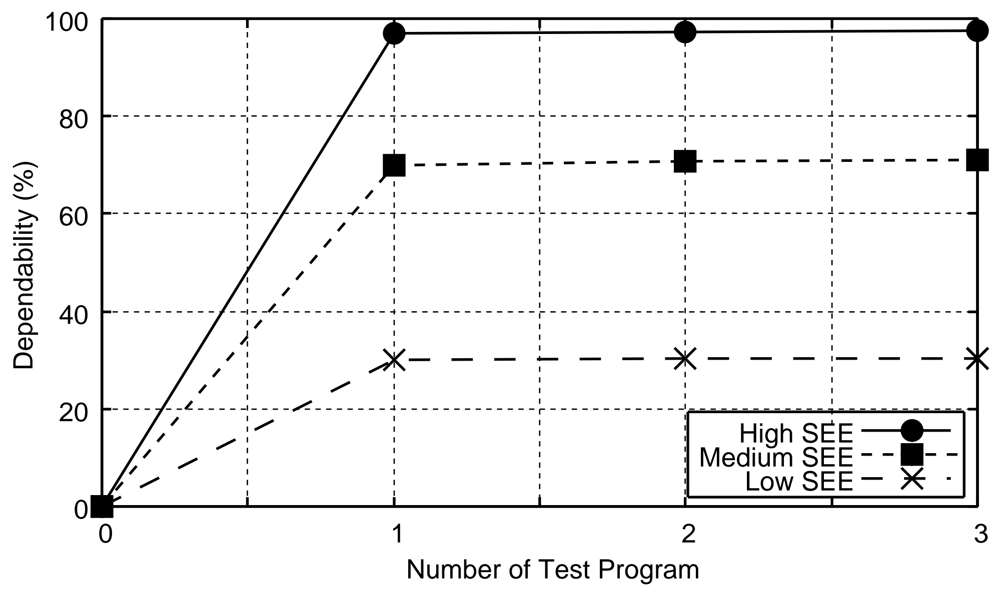


Figure 5.8. Varying the number of test programs.

Chapter 6

Conclusion and Future Work

THIS chapter concludes this dissertation and discusses the possible directions of future works.

1. Conclusion

For a long time in the history of computing, CMOS has continued to scale according to Moore's law, allowing ever-growing system integration and providing continuous performance improvement. On the other hand, as devices shrink perpetually, hardware failure rates are expected to increase due to a wide variety of error sources such as aging, wear-out, infant mortality induced by insufficient burn-in, transient errors caused by alpha particles from the packaging material and cosmic rays, design defects, and others. The pervasiveness of this growing reliability trend demands a low-cost in-field reliability solution that detects, diagnoses, recovers from, and/or repairs around failed components.

This dissertation has proposed techniques to improve the reliability for the reconfigurable architectures based on the following two key observations. First, the overhead of the dual modular redundancy (DMR) with frequent checking to detect the permanent fault is too high. Besides, instructions with a fixed input can increase the probability of fault masking. Second, even though partial redundancy is a good technique to balance the energy consumption and reliability, and the unprotected portion of the program cause very few silent data corruption (SDC) under a low error injection. However, our experimental results have found

that the SDC rate increases while the error injection ratio increased, and due to that the quality of the processed object will decrease.

Based on theses design philosophies, this dissertation has presented two techniques to improve the reliability and to reduce the cost in the redundant solutions. First, in order to detect the permanent fault this dissertation presents a techniques to add selective check instructions on the data-flow-graph that incur low power, while keeping the same reliability. Second, a technique to reduce the SDC in partial redundancy has been presented which is able to reduce the SDC under a very high error injection.

First, we have proposed a technique to remove check instructions from non-critical positions to avoid an exhaustive checking for fault-tolerable execution. The technique can efficiently work with a reconfigurable architecture to achieve high dependability with awareness of the fault possibility. In our approach, check instructions are added selectively according to the error probability (ODP) along the data path when the accumulated possibility exceeds a threshold. In addition, we also studied the influence of constant inputs as they can turn parts of the circuit into don't care zones and therefore help reduce the sensitivity to the faults. Our study of the dependability of the updated data-path has shown that the reliability can still be kept at a near-optimal level when properly removing 52% non-critical check operations. This results in an energy saving of 17% for high dependable execution. With an allowance of downgrading 1.0% reliability, it is possible to reduce the energy further by 22%. In summary, a cost-effective high dependability technique has been achieved by using our ODP metric in the FU array processor.

Second, we have presented a technique to achieve a balance between dependability and cost by applying error-prediction based policies to a partial redundant system. The level of redundancy of a program is adaptively monitored according to SDC rate prediction, based on a sampled fault rate and a program behavior analysis. Furthermore, by adding one self-contained test program, we achieved a super linear SDC reduction superior to the traditional partial redundancy method. Overall, our technique reduces 12% to 0.37% the SDC ratio of the original partial redundancy, reaching a two-order increase in dependability even under a very high error injection rate of 10^{-8} flips per cycle.

2. Limitation and Future Work

While the proposed techniques are shown very effective to serve as the reliable solution for the reconfigurable architectures, there are many directions that can be explored in future work. The following discuss a few of the future research directions.

- First, we have found in our first experiment that adding selective check on the data-path is able to reduce the checking overhead, as well as the energy consumption. In order to evaluate our proposed technique, we use a reconfigurable architecture, and the selective check instructions are added on the data-path during the mapping of the instructions on the functional unit array. An additional module has been used to calculate the ODP of the data-path. The size of the of that module is mostly depend on the odp calculation, which requires floating point operation, and the number of register is almost the same as the number of register used in the ISA. Thus, it occupy a large area of the chip. At the same time, it is not possible to use this technique in a off-the-shelf reconfigurable architecture.

  Therefore, it has been assumed that a compiler support with this technique can overcome the above problems. The compiler will use the information of the hardware such as, the ODP and the optimized ODP of the operations, to calculate the defective probability in the data-path, and will generate the binary. The generated binary can be used at any off-the-shelf commercial reconfigurable architecture, as the there is no necessity of calculating the ODP at run-time.

  Further, the generated binary can also be used to detect and recover the transient fault in a normal pipeline architecture. For example, the processor will execute the binary with reduced check instructions, and a fault detected by any check instructions can be recovered by roll back to the previous checkpoint.

  Additionally, we have found that the instructions with fixed input has a low defective probability. It is also possible that the fixed input instructions can mask the error of previous operations. Thus, a technique can be introduce

in the compiler to group the instructions in such a way that a fixed input instruction can mask the error completely or partially of that group of instructions.

- Second, for the SDC prediction technique, we use the same reconfigurable architecture. That architecture has some limitations such as, it can only process a parallel data-path, and it cannot process a data-path which has dependencies between iterations. Thus, our SDC prediction technique is only evaluated for the parallel data-path. It would be more interesting to evaluate this technique on the other platform such as, a normal pipeline architecture, or a many core architecture, which can process all kinds of data-flow-graph.

  At the same time, the equation for the SDC prediction works only for the parallel data-path and when there is no dependencies between the iterations. The error propagation in such kind of data-flow-graph has different impact depending on the time of the fault strike. If a fault strike at the early stage of the execution that cause many SDCs. On the other hand, the fault strike at the later stage of the execution cause low number of SDCs. Therefore, a nonlinear technique to duplicate the instructions with the time of execution is expected to the reduction of SDCs is a dependent data-flow-graph.

  Besides, the equation for the SDC predictions takes a long time for calculation, which cause additional energy consumption as well as performance loss. A dedicated hardware based on lookup table to predict the SDC of a coming loop is required. In this way, it is able to reduce the prediction time of SDC, which helps to achieve low power and high performance.

Acknowledgements

I was waiting for the last two years to write this section of my dissertation to thank those people who have contributed and supported me during my doctoral study period. First of all, I would like to thank my Supervisor, Prof. Nakashima, for giving me an opportunity to pursue Ph.D. in the Computing Architecture Lab, and supporting me financially. He offered me the opportunity, based on a Skype meeting, and believed in me when I had not enough background and confidence to continue research in the area of computer architecture. His guidance helped me to overcome the difficulties and to build a solid background in computer architecture research and related technologies. Besides, his **EReLA** simulator, has been extensively used throughout my doctoral study, which was very straight forward and easy to learn in a quick time. Thus, it helped me to work faster than the expected time. I would also like to thank Prof. Inoue for the comments on my thesis and serving as a committee member of my doctoral thesis committee.

Prof. Yao is the person who has helped me since my first day in Japan. He picked me up from the airport while I came to Japan. Not only that, he always supported me in the last three years more as a friend than a supervisor. In last three years he always forced me to think critically and big. It would not be possible for me to write this dissertation without his continuous guidance. A special thanks to Prof. Yao. The other committee member, Prof. Hara gave valuable comments my research. Prof Yamashita from Ritsumikan University gave many valuable comments on my first journal submission that helped to improve the quality of the paper. Thank you, Prof. Hara and Prof. Yamashita.

There is no shortage of fellow students to thank. First of all, I met Dr. Yoshimura who was a doctoral student when I joined in the Computing Architecture Lab. He taught me how to use the simulators and tools for hardware development. A special thanks to him. Other than him, I met many students who were being friendly and made my life enjoyable. Thanks to all the past members, especially Naveen and Moritaka, as well as the current members. Secondly, all the past EReLA members, Otani and Yamanaka, and current FUSA members, Kwang and Koike, are greatly thanked for their friendship and Japanese language support. I will miss all of them.

Outside the Computing Architecture Lab. but at NASIT, I found many good

friends. They always cheered me up and always respond to my phone calls and went with me to the restaurants when I wanted to try different foods. Their efforts helped me to forget about my family and concentrate on my research. Thank you Nishanth, Salik, Kartik and Abhinav for the happy moments that I spend with you. At NAIST, I also met two Bangladeshi couples, who were always inviting me at their places to try Bangladeshi foods. Thank you Hasan, Mehnaz and their families for those delicious foods and enjoyable discussion.

During my doctoral study, one of my friends has been always with me from Barcelona, Spain. She always inspired me with her motivational speech, and always cheered me up with her funny things when I was sad. Most importantly She never felt irritated on my stupid jokes. A big thanks goes to Maya for her continuous inspiration till now.

Finally, thanks to my family in Bangladesh for their support. Thanks, mom and dad for your patience and believing in me. Although I'm not sure they ever knew exactly what I was doing. Thanks, mom and dad, for your emails and your conversation. I'm looking forward to seeing you again soon.

References

[1] S. Arslan and G. Shah. A Flexible in-Field Test Controller. In *Proceedings of the 2011 IEEE 14th International Multitopic Conference*, INMIC '11, pages 71 –75, dec. 2011.

[2] M.M. Azeem, S.J. Piestrak, O. Sentieys, and S. Pillement. Error recovery technique for coarse-grained reconfigurable architectures. In *Proceedings of the 2011 IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, DDECS '11, pages 441 –446, april 2011.

[3] Yung-Yuan Chen, Shi-Jinn Horng, and Hung-Chuan Lai. An integrated fault-tolerant design framework for VLIW processors. In *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003*, pages 555 – 562, nov. 2003.

[4] Yung-Yuan Chen and Kuen-Long Leu. Reliable data path design of VLIW processor cores with comprehensive error-coverage assessment. *Microprocessors and Microsystems*, 34(1):49 – 61, 2010.

[5] Naveen Devisetti, Takuya Iwakami, Kazuhiro Yoshimura, Takashi Nakada, Jun Yao, and Yasuhiko Nakashima. LAPP: A Low Power Array Accelerator with Binary Compatibility. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 854–862. IEEE Computer Society, 2011.

[6] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the 17th international conference on Architectural Support for Programming Languages and Operating Systems*, pages 301–312. ACM, 2012.

[7] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks.*, DSN '02, pages 409 – 415, 2002.

[8] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Proceedings of the 2001 conference on Design, automation and test in Europe*, DATE '01, pages 642–649, 2001.

[9] Scott Hauck and André DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, Amsterdam, nov 2007.

[10] Yohei Hazama, Jun Yao, Takashi Nakada, and Yasuhiko Nakashima. A DMR based Parmanent Error Locating Method for a Dependable FU Array. *IEICE Tech. Rep.*, 111(328):47–52, Nov 2011.

[11] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-μm to 90-nm generation. In *Proceeding of IEEE International Electron Devices Meeting, 2003. Technical Digest*, IEDM '03, pages 21.5.1–21.5.4, 2003.

[12] 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 *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 111–122. ACM, 2012.

[13] L. Leem, Hyungmin Cho, J. Bau, Q.A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *Proceedings of Design, Automation Test in Europe*, DATE '10, pages 1560 –1565, 2010.

[14] J. W. McPherson. Reliability challenges for 45nm and beyond. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 176–181, 2006.

[15] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual*

IEEE/ACM International Symposium on Microarchitecture, 2003., MICRO-36, pages 29 – 40, dec. 2003.

[16] N. Oh, P.P. Shirvani, and E.J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63 –75, mar 2002.

[17] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital integrated Circuits- A Design Perspective.* Prentice Hall, 2ed edition, 2004.

[18] E. Rotenberg. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *Proceedings of 29th Annual International Symposium on Fault-Tolerant Computing*, pages 84–91. IEEE, 1999.

[19] Mitsutoshi Saito, Shunsuke Shitaoka, Devisetti V.R. Naveen, Suguru Oue, Kazuhiro Yoshimura, Jun Yao, Takashi Nakada, and Yasuhiko Nakashima. Development of High Power-Efficient Processor with Linear FU Array Accelerator [in japanese]. *IEICE TRANSACTIONS on Information and Systems (Japanese Edition)*, J95D:1729–1737, Sept. 2012.

[20] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, PLDI '11, pages 164–174. ACM, 2011.

[21] P.N. Sanda, J.W. Kellington, P Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones. Soft-error resilience of the IBM POWER6 processor. *IBM Journal of Research and Development*, 52(3):275–284, 2008.

[22] Smitha Shyam, Kypros Constantinides, Sujay Phadke, Valeria Bertacco, and Todd Austin. Ultra low-cost defect protection for microprocessor pipelines. *SIGARCH Comput. Archit. News*, 34(5):73–82, October 2006.

[23] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of the 2004 International*

Conference on Dependable Systems and Networks, DSN'04, pages 177 – 186, june-1 july 2004.

[24] Jayanth Srinivasan, S.V. Adve, Pradip Bose, and J.A. Rivers. Lifetime Reliability: Toward an architectural solution. *IEEE Micro*, 25(3):70 – 80, May-June 2005.

[25] J. H. Stathis. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development*, 46(2.3):265 –286, march 2002.

[26] A. Suga and K. Matsunami. Introducing the FR500 embedded microprocessor. *Micro, IEEE*, 20(4):21 –27, jul/aug 2000.

[27] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E.M. Panainte. The MOLEN polymorphic processor. *IEEE Transactions on Computers*, 53(11):1363 – 1375, nov. 2004.

[28] Fan Wang and V.D. Agrawal. Single Event Upset: An embedded tutorial. In *21st International Conference on VLSI Design, 2008*, VLSID '08, pages 429 –434, Jan. 2008.

[29] Xin Xu and Man-Lap Li. Understanding soft error propagation using efficient vulnerability-driven fault injection. In *Proceedings of 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '12, pages 1–12, june 2012.

[30] Jun Yao, S. Okada, M. Masuda, K. Kobayashi, and Y. Nakashima. DARA: A low-cost reliable architecture based on unhardened devices and its case study of radiation stress test. *IEEE Transaction on Nuclear Science*, 59(6):2852–2858, 2012.

[31] Kazuhiro Yoshimura, Takuya Iwakami, Takashi Nakada, Jun Yao, Hajime Shimada, and Yasuhiko Nakashima. An instruction mapping scheme for FU array accelerator. *IEICE Transactions on Information and Systems*, 94(2):286–297, February 2011.

Achievements

Journal Articles

1. T. Ahmed, J. Yao, Y. Hara-Azumi, S.Yamashita, and Y.Nakashima, "Selective Check of Data-Path for Effective Fault Tolerance", IEICE transaction of Information and Systems (Special Section of Reconfigurable System), Vol. E96-D, No. 8, pp. 1592–1601, Aug. 2013.

Conference (Peer Reviewed)

1. T. Ahmed, J. Yao, and Y. Nakashima, "A Two-Order Increase in Robustness of Partial Redundancy under Radiation Stress Test by Using SDC Prediction", in proceedings of the 2013 Conference on Radiation Effects on Components and Systems, RADECS 2013, pp. 1–7, paper no. 7, Oxford, UK, Sept., 2013.

2. T. Ahmed, J. Yao, and Y.Nakashima, "Introducing OVP Awareness to Achieve an Efficient Permanent Defect Locating", in proceeding of the 2012 IEEE/ACM Symposium on Nanoscale Architecture, Nanoarch '12, pp. 43–49, Amsterdam, The Netherlands, July, 2012.

Conference (Not Peer Reviewed)

1. T. Ahmed, J.Yao, and Y. Nakashima, "Achieving Near-Optimal Dependability with Minimal Hardware Costs in an FU Array Processor by Soft Error Rate Monitoring", SWoPP'12, IPSJ SIG Notes, 2012-ARC-201(19), pp. 1–6, Aug., 2012.

2. T. Ahmed, J. Yao, and Y. Nakashima, "Achieving Effective Fault Tolerance in FU array by Adding AVF Awareness", IPSJ SIG Notes, Information Processing Society of Japan, 2012, vol-2012, no-5, pp. 1–4.