



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



**COMPILER SUPPORT FOR ENHANCING
RELIABILITY OF NETWORK-ON-CHIP
ARCHITECTURES**

MUHAMMAD ADITYA SASONGKO

MASTER THESIS

Department of Computer Engineering

ADVISOR

Prof. Dr. Haluk Rahmi Topcuoğlu

ISTANBUL, 2017



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



COMPILER SUPPORT FOR ENHANCING
RELIABILITY OF NETWORK-ON-CHIP
ARCHITECTURES

MUHAMMAD ADITYA SASONGKO

(524112924)

MASTER THESIS

Department of Computer Engineering

ADVISOR

Prof. Dr. Haluk Rahmi Topcuoğlu

ISTANBUL, 2017

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Muhammad Aditya SASONGKO, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled “**Compiler Support for Enhancing Reliability of Network-on-Chip Architectures**”, on 18.01.2017 and has been found to be satisfactory by the jury members.

Jury Members

Prof. Dr. Haluk Rahmi TOPÇUOĞLU (Advisor)

Marmara University

Prof. Dr. Can ÖZTURAN (Jury Member)

Boğaziçi University

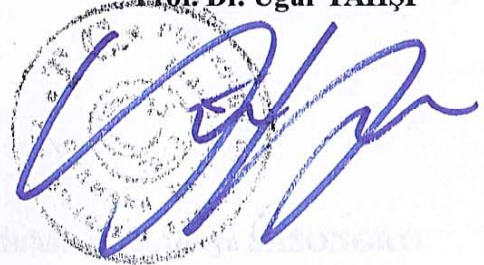
Asst. Prof. Dr. Ömer KORÇAK (Jury Member)

Marmara University

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Muhammad Aditya SASONGKO be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program on ..30.01.2017..... (Resolution no: 2017/03.02..).

Director of the Institute
Prof. Dr. Uğur YAŞI



ACKNOWLEDGMENT

I would like to express my gratitude to my thesis supervisor, Prof. Haluk Rahmi Topcuoğlu, for his support, guidance and encouragement throughout my graduate study that motivated me to finish my work.

I also would like to thank Prof. Can Özturan and Asst. Prof. Ömer Korçak for participating in the committee of my thesis defense and giving me feedback.

I want to express my gratitude to my colleagues from the Multicore Computing Research Laboratory of Marmara University, Sanem Arslan Yılmaz and Zuhale Öztürk, who have supported me during my research.

I gratefully acknowledge the financial support of The Scientific and Technological Research Council of Turkey (TUBITAK) through a research grant (Project Number 113E530).

Finally, I would like to thank my family for their encouragement, patience and support throughout my education.

January, 2017

Muhammad Aditya SASONGKO

TABLE OF CONTENTS

ACKNOWLEDGMENT.....	i
TABLE OF CONTENTS.....	ii
ÖZET.....	iv
ABSTRACT	v
ABBREVIATIONS	vi
LIST OF FIGURES	vii
LIST OF TABLES.....	ix
1. INTRODUCTION.....	1
2. FAULT TOLERANCE TECHNIQUES FOR NETWORK-ON-CHIP ARCHITECTURES	6
2.1 Soft Errors in NoCs	6
2.2 Fault Tolerance for NoCs.....	7
2.3 Fault Tolerance through Partial Protection.....	11
2.4 Compiler-based Fault Tolerance Techniques	12
3. COMPILER ENHANCED RELIABILITY FRAMEWORK FOR NoC ARCHITECTURES	15
3.1 Compiler Part of Our Framework.....	15
3.1.1 Polyhedral Model.....	16
3.1.2 Design of Compiler Support.....	20
3.2 Tracking and Protecting Critical Communications on Hardware Level..	24
4. EXPERIMENTAL STUDY.....	28
4.1 Setup for Experiments.....	28
4.1.1 Sniper Simulator.....	28
4.1.2 Configuration of Simulated Architecture	29
4.1.3 Compared Fault Tolerance Schemes.....	29
4.1.4 Benchmarks.....	30
4.1.5 Criticality Annotations with Compiler Support.....	31
4.1.6 Fault Injection Framework	37
4.1.7 Measurement Metrics	37
4.2 Experimental Results and Discussion	38
4.2.1 Evaluating Performance and Fault Tolerance of The Schemes.....	38
4.2.2 Evaluating Impact of Annotation Sizes and Shapes.....	43
4.2.3 Evaluating Impact of Route Length Upper Bound	48

5. CONCLUSIONS AND FUTURE WORK	54
6. REFERENCES.....	56

ÖZET

Yonga-üstü-Ağ Mimarilerinin Güvenilirliğini Artırmak için Derleyici Desteği

Yonga-üstü-Ağların (YüA) küçülen boyutları sistem tasarımlarında güvenilirliğin önemini arttırmaktadır. Nano-ölçekli boyutlar, sistemlerin radyasyon parçacıkları ve elektromanyetik parazitler nedeniyle oluşabilecek hatalara karşı hassasiyetini arttırmıştır. Bu hassasiyet, YüA mimarileri üzerinde çalıştırılan çok iş parçacıklı uygulamalar için bir tehdit oluşturmaktadır. Artıklama gerektiren geleneksel hata tolerans yöntemleri tüm sistemi göz önüne aldığı anda, sınırlı performans ya da enerji bütçesinin olduğu durumlarda, ciddi yük oluşturmaktadır. Bu çalışmamızda, YüA mimarilerinde performans ve enerji maliyetlerini minimize ederek güvenilirlik problemini çözmek için derleyici desteği ve donanım seviyesinde hata tolerans yöntemi içeren yeni bir güvenilirlik sistemi sunmaktayız. Bu yöntemde, uygulama içerisinde kritik veriler belirlenerek, sadece kritik verilerin YüA üzerinde transferinde ek koruma sağlanmaktadır.

Derleyici desteği, verilen bir programda programcıların kritik olarak gördükleri verilerin işaretlemesini sağlamaktadır. Derleyici seviyesinde geriye dönük program dilimleme yardımı ile, işaretlenen verilere direkt ya da dolaylı olarak değerlerini sağlayan değişken veya dizi elemanlarının tespit edilmesi sağlanmaktadır. Sunduğumuz sistem, kritik veriyi taşıyan iletişimler için kullanılan korunmuş bağlantı oranını maksimize eden bir rotalama algoritması ile YüA seviyesinde ek koruma sağlamaktadır. Gerçeklenen deneysel çalışmalarımızda, önerdiğimiz kısmi güvenilir yöntem, ayırım gözetmeden tüm verilerin iletimini koruyan yöntemle karşılaştırıldığında, kritik verilerin iletiminde yaklaşık olarak eşit seviyede hata toleransı sağladığı, fakat bu geleneksel yöntemden daha az enerji tüketimi ve daha yüksek performans ortaya koyduğu doğrulanmıştır.

January, 2017

Muhammad Aditya SASONGKO

ABSTRACT

Compiler Support for Enhancing Reliability of Network-on-Chip Architectures

Smaller feature sizes in the Networks-on-Chip (NoCs) have raised reliability as an important issue. The nanoscale sizes have increased the vulnerability of the whole system to errors due to radiation particles or electromagnetic interferences. This vulnerability poses a threat to the execution of the multithreaded applications in NoC architectures. Traditional fault tolerance schemes which require redundancies can be overburdening when they cover the entire system and while facing a limited amount of performance or energy budget. In order to tackle the reliability problem in NoC architectures while minimizing performance and energy costs, we propose a new reliability framework in this work which incorporates a compiler-support and a hardware-level fault-tolerance scheme to identify the critical data and provide extra protection only to critical data transmissions.

The compiler support in our framework enables programmers to annotate portions of data that are considered as critical. Then, it performs backward program slicing to mark other variables or array elements which directly/indirectly assign values to the annotated data. The output of the compiler support is the final set of critical data due to annotation and program slicing. Our framework assigns extra protection in NoC layer with a routing algorithm that maximizes the proportion of protected links for communications which carry the critical data. The experimental study validates that our scheme provides almost equal level of fault tolerance for critical data transmissions with the scheme that protects all data transmissions indiscriminately through fault-tolerant routing, while having less energy consumption and better performance than more conservative schemes.

January, 2017

Muhammad Aditya SASONGKO

ABBREVIATIONS

CFE : Control Flow Error

CFG : Control Flow Graph

CMP : Chip Multi Processors

COTS : Commercial-of-the-Shelf

ECC : Error-Correcting Code

ECC-SECDED : Error-Correcting Code-Single Error Correction Double Error-Detection

FEC : Forward Error Correction

GCC : GNU Compiler Collection

IC : Integrated Circuit

MAF : Maximal Aggressor Fault

NIC : Network Interface Card

NoC : Network-on-Chip

SCoP : Static Control Part

SEU : Single Event Upset

SIHFT : Software Implemented Hardware Fault Tolerance

SM : Signature Monitoring

SoC : System-on-Chip

SPMD : Single-Program Multiple-Data

TMR : Triple Modular Redundancy

HBH : Hop-by-Hop

SEC : Single Error Correction

LIST OF FIGURES

Figure 3.1. The workflow of the compiler-enhanced reliability framework.....	15
Figure 3.2. Iteration domain.....	17
Figure 3.3. Access Function	18
Figure 3.4. Data dependence between statement 1 and 2 on variable s.....	20
Figure 3.5. Example of criticality pragma.....	23
Figure 3.6. A partially protected 8x8 NoC which has 27 protected links.....	25
Figure 3.7. XY route on partially protected NoC	26
Figure 3.8. Fault-tolerant route with $x=0$	27
Figure 3.9. Fault-tolerant route with $x=2$	27
Figure 4.1. Manual criticality annotations on 1-dimensional and 2-dimensional result arrays	32
Figure 4.2. Manual criticality annotations on 1-dimensional result array of gesummv application for the second experiment	32
Figure 4.3. Manual criticality annotations on 2-dimensional result array of covariance application for the second experiment	33
Figure 4.4. SCoP of covariance benchmark.....	34
Figure 4.5. SCoP of gesummv benchmark	35
Figure 4.6. Total Packet Latency of Compared Schemes	38
Figure 4.7. Application Execution Time of Compared Schemes.....	39
Figure 4.8. Percentage of Protected Links on Critical Communications of Compared Schemes	40
Figure 4.9. Proportion of Erroneous Bytes on Critical Communications.....	40
Figure 4.10. Proportion of Protected Links on All Communications of Compared Schemes	42
Figure 4.11. Proportion of Erroneous Bytes on All Communications.....	42
Figure 4.12. Energy Consumption of Compared Schemes	43
Figure 4.13. Total Packet Latency of Compared Cases	44
Figure 4.14. Application Execution Time of Compared Cases	44
Figure 4.15. Percentage of Protected Links on Critical Communications of Compared Cases	45
Figure 4.16. Proportion of Erroneous Bytes on Critical Communications.....	46
Figure 4.17. Percentage of Protected Links on All Communications of Compared Cases	46
Figure 4.18. Proportion of Erroneous Bytes on All Communications.....	46
Figure 4.19. Energy Consumption of Compared Cases	48
Figure 4.20. Total Packet Latency over Different Length Constraints	49
Figure 4.21. Application Execution Time over Different Length Constraints	49
Figure 4.22. Percentage of Protected Links on Critical Communications over Different Length Constraints	50
Figure 4.23. Proportion of Erroneous Bytes on Critical Communications over Different Length Constraints	51
Figure 4.24. Percentage of Protected Links on All Communications over Different Length Constraints.....	51

Figure 4.25. Proportion of Erroneous Bytes on All Communications over Different Length Constraints.....	52
Figure 4.26. Energy Consumption over Different Length Constraints.....	53

LIST OF TABLES

Table 3.1. Syntax of criticality pragma	22
Table 4.1. Applications considered in experimental study	31
Table 4.2. Results of program slicing of criticality for the second experiment	36

1. INTRODUCTION

In recent decades, components embedded on computer chips get smaller in size, while at the same time chip density keeps increasing as the number of transistors packed into manufactured chips increases. Probability of power dissipation gets higher as power consumption and clock frequency increase in modern architectures. As current computer architectures seem to hit a *power wall*, one solution to keep improving computation performance and throughput is to deploy multiple processing units concurrently rather than to keep increasing the performance of a single processor.

By following this strategy, increasingly high demand for fast computation has opened way to chip multi processors (CMP) and then many-core architectures, to be manufactured and deployed. Since the number of processing units in a single chip keeps increasing, buses, which are used as means of communication between cores on the chip, began to become a significant bottleneck in computer performance since the contention in communication between cores on a chip gets higher as the number of cores embedded on a single chip keeps increasing in modern chip products [1]. Therefore, a different and more scalable interconnected system was required. This new system which adopts the communication strategy used in computer network is called Network-on-Chip (NoC) [2, 3].

Network-on-Chip is an architecture which comprises multiple processing nodes that are connected with interconnected network and is built on a single chip. The processing nodes in this system communicate with each other by packing their message into packets, routing, and transmitting them between each other. Using the same strategy in computer networks, there is also a separation between computation and communication layers in a NoC system. A NoC normally consists of processors, Network Interface Card (NIC), routers, and link wires. Similar to how it functions in computer network, NIC in Network-on-Chip also packages data or message from a sending processing node into packets and deliver them to router to be sent and routed to receiver node. The communication between a pair of sender and receiver is packet-based, and there is no dedicated link for it.

As mentioned in [4], it was predicted that the amount of errors due to mistakes in manufacture and errors that are generated during run time would increase significantly in nanoscale systems. This reliability issue that has become relevant in most modern

computer chips has also become a serious design issue in NoCs. The increased chip density and the reduction of noise margin due to reduction of power supply voltage for minimizing power dissipation make the entire architecture become more prone to transient errors [5].

In order to tackle problems related to reliability, several techniques have been attempted. One of the used solutions is by hardening the hardware components that need to be protected through shielding. The hardening can be performed during manufacturing process of the devices. However, shielding can significantly increase production costs [6]. A type of fault protection method that is cheaper than shielding is hardening-by-system. Several examples of this type of method are redundancy of devices for error detection or correction, error detection algorithms, and memory scrubbing. This method can be implemented on Commercial-of-The-Shelf components (COTS). A drawback of this method is that it requires additional resources. Another method that is perhaps the most cost-efficient in terms of manufacturing cost is Software Implemented Hardware Fault Tolerance (SIHFT) [7-9]. This approach is cost-efficient because it doesn't require any additional hardware resource at all. However, this method introduces redundancy in higher level of abstraction than the redundancy introduced by COTS, since the redundancy in SIHFT is in programming language code level while the redundancy in COTS is in hardware level. Therefore, the performance of SIHFT is likely to be worse than the performance of COTS approaches which use redundancy in lower and finer-grained level.

Due to the redundancy that it introduces, implementation of fault tolerance is costly in terms of energy consumption and use of additional areas. Under constrained budget, providing fault tolerance to all parts of NoC architecture might become too expensive. As stated in [10], reducing the requirement of 100% correctness in the utilization of various on-chip components and channels can decrease the costs incurred by fabrication process. Furthermore, as shown in [11-14], protection schemes that protect only certain components of targeted architectures are able to reduce hardware costs compared to the traditional schemes which provide protection equally to all targeted components.

To maximize reliability while minimizing performance and energy costs, a strategy has to be used to decide which components of targeted architectures are to be protected

among all of the available components. One option is to protect the components which are utilized by critical instructions or data. A work in [15] shows that instructions in a same program can have varying levels of criticality. Critical instructions are defined as those which are more likely to propagate the faults that they have to visible output, while the errors in less critical or non-critical instructions are more likely to be masked. Therefore, in order to minimize performance and energy costs while maintaining reliability within tolerable level, a fault tolerance scheme which is able to identify critical parts of program execution and assign necessary protection to those parts is needed.

The work in this thesis aims at achieving the goal of creating a fault tolerance scheme that is cost-effective. The targeted architecture of this method is Networks-on-Chip. In order to address reliability issue in NoCs, we introduce a new framework for protecting only the communications that carry critical data. In our framework, fault tolerance in hardware level is provided by using a partially protected NoC which consists of *protected links* and *unprotected links*. The fault tolerance techniques for the protected links can be in the form of hardening of wires in the links, hop-to-hop retransmission [16-18], or protection by coding scheme [16, 19-20]. However, it is also possible to interpret the classification between *protected links* and *unprotected links* in a different way. *Unprotected links* might not actually be left with no protection at all, but they can be assigned with less costly protection scheme compared to *protected links*. In the experimental study, *unprotected links* is protected with hop-by-hop Single Error Correction (SEC) coding scheme which is implemented only to flits which carry packet headers (header flits). This header flit-only protection follows the example in [16] where header flits are protected on switch-to-switch basis for end-to-end error detection scheme. On the other hand, the same coding scheme is applied to protect every single flit that crosses *protected links*. Furthermore, our method is equipped with a compiler support which was created by modifying Pluto parallelizer [21-23]. The compiler support lets programmers mark data in input applications that they consider critical with annotation. Clan library [24] was modified to support a new pragma that allows manual annotation by programmers. The criticality which is introduced by the annotation will propagate from the annotated data to other variables or array elements which assign values to the variables and array elements which are already marked as critical. The output of the compiler support is the final set of critical data that has to be assigned with extra

protection in hardware layer.

In the hardware layer, any packet transmission which carries the critical data will be directed to cross a route which contains as many protected links as possible given a maximum constraint length of routes. The default constraint length is the length of shortest paths. From here on, any packet transmission which carries critical data is referred to as critical communication. The rest of packet transmissions which don't carry critical data will follow XY-routing algorithm which is known to be deadlock-free. Furthermore, in order to ensure that the entire proposed scheme is deadlock-free, the generated routes for packet transmissions which contain critical data will not include the turns that are disabled in west-first routing algorithm. The entire routing algorithms used in the proposed scheme will not create any cycle which causes deadlock, since west-first routing algorithm is already proven to be deadlock-free [25], and the turns that are disabled in the west-first routing algorithm are also disabled in the XY-routing algorithm.

An experimental study is carried out to evaluate the proposed scheme. This experimental study consists of three phases. Firstly, the introduced scheme is compared with an XY-routing scheme which uses the hardware layer of the introduced scheme and with another fault tolerant scheme that treats all data transmissions as critical communications. The experiment results show that, on average, the proposed scheme improves the *proportion of protected links on critical communications* by approximately 1.76 times and reduces the *proportion of erroneous bytes on critical communications* by 18% over the XY-routing scheme on the same partially protected NoC hardware. Our proposed scheme also improves total packet latency by approximately 6%, execution time by approximately 3%, and energy consumption by approximately 2% over the more conservative fault tolerant scheme which generates routes that maximize the use of *protected links* for all data transmission. When compared with another conservative scheme which uses hop-by-hop Single Error Correction (SEC) coding scheme [16, 26], our scheme improves total packet latency by approximately 7%, execution time by approximately 14%, and energy consumption by approximately 12%. Furthermore, the second experiment shows that the proportion of protected links and the total packet latency of all communications are directly proportional with the size of the final set of data that is marked by the compiler support as critical. Finally, in the third experiment, it is shown that higher upper bound for the length of fault-tolerant routes causes critical

communications to be more fault tolerant. It is due to the ability to include more protected links to the routes. However, as the upper bound of length increases, the total packet latency is also shown to increase. Nevertheless, the increase of total packet latency doesn't seem to have significant impact on the entire application execution time and energy consumption, since the execution time and energy consumption of most applications in the experiment don't exhibit significant increment over increasing length constraints.

To sum up, the main contributions of this work are as follows.

- A compiler-enhanced reliability framework is proposed which includes a compiler support that enables programmers to mark a portion of data. The compiler layer of the framework also identifies array elements and scalar variables which assign values to the selected data and mark them as critical.
- The proposed framework assigns extra protection in NoC layer for critical communications which carry the critical data. The extra protection is provided by running a routing algorithm which maximizes the proportion of *protected links* in the routes generated for the critical communications.
- An experimental study is carried out which compares our scheme with an oblivious scheme and two more conservative schemes. The experimental study also evaluates our scheme under different annotations and route length constraints.

The rest of the thesis has the following organization. In Chapter 2, we present an overview on soft error problems in NoCs. Then, related work on fault tolerance techniques for NoCs are also presented in the same chapter. In Chapter 3, we present details of our compiler-enhanced fault-tolerance framework for NoC architectures. Chapter 5 explains the setup and discusses the results of experiment in this work. Finally, chapter 6 summarizes the work in this thesis and concludes the results.

2. FAULT TOLERANCE TECHNIQUES FOR NETWORK-ON-CHIP ARCHITECTURES

In this chapter, we present an overview on soft errors in NoCs. Then, existing fault tolerance schemes which target Networks-on-Chip architectures will be reviewed. Furthermore, as the fault tolerance framework proposed in this thesis offers partial protection by protecting only hardware components which are used by critical data, some existing publications that have also proposed partial protection are summarized in this chapter. Since our framework uses compiler to direct fault protection in hardware level, it will also be necessary to review several publications which implement compiler-based fault tolerance.

2.1 Soft Errors in NoCs

Nanometer technologies which have been widely used in modern computer chips have allowed transistors in very large quantity to be packed in a single die. As the sizes of transistors shrink, voltage level used on the chips must also be decreased in order to keep power dissipation within tolerable degree. This reduction of supply voltage level and decrease of transistor dimensions cause the increase of sensitivity to soft errors [27]. Furthermore, the reduction of silicon die sizes also increases the vulnerability of the entire system on-chip to cross talks, high field effects, and critical leakage currents which might cause transient and permanent errors [18].

From all components of a NoC architecture, there are three components that are considered to be the most vulnerable to soft errors. These components are router input buffers, crossbars, and network links [28].

Radiation effects are the main cause of transient faults in switch logic [29]. Electrically-charged particles due to sun activity can cause transient current pulse when interacting with integrated circuit (IC), and in turn, can reverse cell state in sequential logic [30, 31]. The reversed cell state or bit flip is a fault which is termed as Single Event Upset (SEU). Beside neutron particles from cosmic radiation, bit flips can also be caused by alpha particles due to impurities in packaging materials [18, 30]. Bit flip faults caused by those particles can occur in storage cells of an electronic circuit, such as latches, flip

flops, or memory cells [31]. The likelihood of faults that are caused by these particles increases as the feature sizes in a chip keep decreasing and, therefore, electric charge stored in sequential elements on the chip can be changed more easily because of the effects of collision with the radiation particles.

Crosstalk is a prevalent source of transient malfunction in interconnecting links [19]. One factor that causes crosstalk is the reduction of noise margin in the link wires since the power of supply voltage is reduced in modern computer chips to minimize power dissipation. As the link wires are closely packed to each other, the signal on each wire has greater effects on neighboring wires, which can cause signal delays, glitches, or damped voltage oscillations [32]. To model faults on link wires due to crosstalk, some of previous researches that proposed fault tolerance techniques for link wires followed MAF model which considers error in only a single wire [32]. There are also those that used a model where higher number of bit flips is possible, such as [33] which implemented a model where a bit flip on a wire can affect up to three adjacent bit flips in order to simulate a burst error.

The framework that is proposed in this work can be used to protect critical communications from errors which might occur either in router buffers or network links by marking some of the links in the topology as *protected links*. These *protected links* can be equipped with needed fault tolerance techniques which can detect or correct errors that might happen in link wires or input and output buffers of routers that are connected to the wires. The runtime system of the NoC makes use of this hardware level protection to protect critical communications by directing these communications to include protected links in their paths.

2.2 Fault Tolerance for NoCs

Several strategies to provide reliability on NOC have been developed throughout years. Similar to fault tolerance strategies applied to other kinds of architecture, fault tolerance in NOC relies on redundancy to detect, mask, or correct errors. By considering the type of redundancy that they use, fault tolerance strategies can be classified into spatial redundancy, temporal redundancy, and information redundancy.

Fault tolerance approaches developed for NOC in literature can be classified

according to the layer to which they are applied. A survey in [29] which reviews existing fault tolerance methods for Networks-on-Chip focuses their review on three layers, data link layer, network layer, and transport layer.

Redundancy techniques for fault tolerance that have been proposed for data link layer in literature rely on all of the three redundancy strategies; spatial redundancy, temporal redundancy, and information redundancy [29]. Since the hardware-level fault tolerance techniques used in the framework introduced in this thesis are mainly based on temporal and information redundancies, the fault tolerance techniques on data link layer which are reviewed in this subsection will be those which are based on temporal and information redundancies.

Temporal redundancy in data link layer is implemented by retransmitting packets or flits when error is detected in previous transmission, by taking multiple samples of packet (multi-sampling), flit or signal transmissions, or by repeating computation when the previous computation result is erroneous [29]. Some works which use multi-sampling as means to detect and correct error are [34] and [20].

Another study proposes a fault tolerance scheme which tolerates timing errors in link wires of NOCs [34]. This scheme divides each link into several pipeline stages and each pipeline buffer contains two flip-flops, which are called main flip-flop and delayed flip-flop. The incoming signal that reaches a pipeline buffer is sampled twice. The main flip-flop samples the signal at main clock edge (clk), and the delayed flip-flop samples the signal at delay clock edge (clk-d). In case there is difference between the samples in main flip-flop and delayed flip-flop, the sample in delay flip-flop will be used.

In another study [20], a variation of sampling method used in [34] is implemented. Instead of deploying only two sample storages, three storages or registers are used to detect and correct errors which might occur in link wires through triple modular redundancy scheme. Each incoming signal is sampled three times, and majority voter is utilized to decide the correct signal in case a difference among the sampled signals is detected.

Information redundancy in data link layer is applied by embedding an additional code to the original protected information which is encoded and decoded within a single hop-by-hop transmission. When error happens to the information or to the protecting

code, the error will be detected and, probably, corrected after the code is decoded. There have been a number of coding schemes which have been implemented in previous works. Several works implemented coding scheme simply for error detection purpose [16-17, 35]. These works combine information redundancy for error detection with temporal redundancy. Once an error is detected, the correct information is retransmitted. Some other works use coding schemes for Forward Error Correction (FEC). There is a variety of coding techniques that have been used for FEC. [16] uses EC/ED (Error Correction / Error Detection) scheme which can correct single bit error (FEC) and detect multiple bit errors. In case a multiple bit error is detected, retransmission of flit is requested. ECC is used in [20] to correct errors due to SEUs that might happen in input buffers of switches. More complex coding schemes are needed to correct multiple bit errors. For instance, [36] employs Bose-Chaudhuri-Hocquenghem (BCH) codes to correct multiple bit errors that can occur due to multiple single-bit or burst errors in network-on-chip (NoC) links. A work in [27] introduces unequal error protection code which can correct or detect single bit and adjacent double bit errors that inflict a whole packet instead of only a single flit. This code can correct any single bit error and detect any double adjacent bit errors in an entire packet/code word. More emphasis is given for protection on the packet header as this code can correct all double adjacent bit errors in the header, while it only has the capability to detect double adjacent bit errors in other parts. Hamming product code is used to correct adjacent bit errors due to burst errors which can be induced by crosstalk [33].

Fault tolerance methods which protect network layer are mainly concerned with routing and switching strategies. Some methods for network layer, such as flooding [37], gossip flooding [38], and random walk [39] provide fault tolerance by duplicating packets or flits (transmission units in data link layer) which are passed through different paths in reaching the destination node. Some other methods for network layer involve diagnosis of faults in interconnects or routers and reconfigure the switching for any packet or flit transmission to avoid paths which pass the faulty interconnects or routers [40, 41].

Transport layer deals with end-to-end transmission of packets from sender to destination. Fault tolerance strategies which are applied on this layer are based on time redundancy, information redundancy, hybrid time/information redundancy, and spatial redundancy [29]. The previous works that will be discussed in the following paragraphs

will be those which introduce techniques that deploy time redundancy, information redundancy, and hybrid time/information redundancy, since spatial redundancy is not implemented in the scheme proposed in this thesis.

Time/temporal redundancy in transport layer involves retransmission of a packet or a flit from sender to destination once an error is detected on the transmitted flit or packet by the destination node. Normally this strategy requires destination node to transmit a feedback signal after it acknowledges a flit or packet that it receives (ACK) or after it detects an error in the packet or flit that it receives (NACK). An example of fault tolerance technique which uses temporal redundancy in transport layer is [18]. In this work, an acknowledgement signal (ACK) is transmitted from destination of a packet to the sender of a packet once the destination node receives certain number of packets from the sender without detecting any error. On the other hand, a warning signal (NACK) is transmitted when a packet is identified to be erroneous. Another work in [16] compare several error correction schemes in transport layer, which are Automatic Repeat Request (ARQ) with parity bit, ARQ with CRC, and hybrid ARQ with Hamming ECC-SECDED. A comparison was also performed between end-to-end protection schemes with hop-by-hop protection schemes. The article concluded that end-to-end schemes are more power efficient than link layer schemes. However, as stated in [42], both end-to-end ARQ or Hybrid ARQ (HARQ) schemes have higher overheads in terms of retransmission buffer size in packet sender node which has to be bigger than the retransmission buffer size in either hop-by-hop ARQ or HARQ schemes. This is due to the necessity to account for the transmission of NACK from the receiver node within the distance of worst case scenario. Until NACK or ACK is received, the transmitted flits which will be necessary to be retransmitted must be stored in the retransmission buffer of the packet sender node. Furthermore, the work in [42] also mentions that the latency in end-to-end schemes can be higher than the latency in hop-by-hop schemes when the error rates are high.

Information redundancy techniques in transport layer are similar with information redundancy techniques in data link layer in the sense that coding scheme is utilized to detect and correct errors in the unit of data transmitted in that layer. A work in [43] that compares forward error correction (FEC) in data link layer (hop-by-hop) and FEC in transport layer (E2E) concludes that protection at transport layer is more power-efficient than protection at data link layer. The coding scheme that was used in that work is

Hamming ECC-SECDED.

There is also a fault tolerance technique for transport layer which involves transmission of duplicated packets through different non-intersecting routes [44]. Another work deploys route discovery and route maintenance to find an available route for transmission from source node to destination node to ensure the availability of that route [45]. However, those fault-tolerance schemes which typically rely on redundancies for error detection, correction or tolerance might be too costly if there is limited budget of performance or energy available.

2.3 Fault Tolerance through Partial Protection

In [46], a hardware level fault tolerance technique was proposed. This technique applies spatial redundancy by running two similar threads on two different cores when one of the cores is idle. Redundancy is only introduced when there is opportunity for it, i.e. there is usable core that is idle.

In another work, fault tolerance through redundancy is implemented to instructions that are considered “vulnerable” [15]. Metric to measure the vulnerability level of each instruction is used to support the scheme. The instructions with high vulnerability are those which have the highest probability to propagate errors that they encounter to visible output. These instructions are then duplicated and the results of their executions are compared in order to detect possible errors.

Researchers propose fault tolerance scheme which offers partial reliability by using two types of processing cores [11,12]. The first type of core is called unreliable core, and it doesn't have error correcting code protection on its cache. The second type of core is called reliable core and it has ECC embedded on its local cache. Threads that are working on critical parts of a program and are currently mapped to unreliable core will be mapped reliable cores. On the other hand, threads that have finished working on critical parts will be mapped to unreliable cores.

A heterogeneous-reliability scheme which incorporates both software and hardware supports to provide necessary protections for memory regions against memory errors is proposed in a related study [13]. The technique proposed in this work at first uses the software support to characterize the reliability requirements of memory regions in

applications. Based on the classification by the software support, the memory regions of the input applications are mapped to the locations in hardware which are equipped with different hardware-level fault tolerance schemes, such as ECC-SECDED, parity, or no protection. The memory regions which have higher reliability requirements will be mapped to locations which have higher error detection or correction capability.

Another work proposes Dynamic Reliability Management System (DyReMS) which is a scheme that is “resilience-driven resource allocation and mapping” [14]. The scheme is supported with an underlying multi-core architecture which has cores that have different fault protection techniques with varying levels of error detection and correction capabilities. The cores are classified into fully-protected cores, partially-protected cores, and unprotected cores. Fully protected cores are equipped with hardware level fault tolerance features like triple modular redundancy (TMR), partially protected cores have ECC-protected memories, and unprotected cores are not equipped with any fault tolerance feature. The scheme, at first, identifies the resilience properties of tasks in parallel program. Then, based on the different levels of resilience of the tasks, the tasks are mapped to the cores which have varying error recovery capability. Low-resilient tasks will be mapped to either fully-protected or partially-protected cores, while highly-resilient tasks will be mapped to unprotected cores.

2.4 Compiler-based Fault Tolerance Techniques

Compiler technique is used in this work as a guide to direct the fault tolerance schemes in hardware level in protecting critical components of the targeted Networks-on-Chip architecture. A benefit that can be gained from using compiler is that the analysis which differentiate critical and non-critical components becomes doable automatically and possible even on large-sized input program.

The idea of using compiler to assist in fault protection of input program is not new. Compiler techniques for fault detection and correction have been implemented in various works. In [47], some compiler techniques are used to assist a multiple-instruction-retry scheme which facilitates recovery from transient errors in processors. The compiler techniques used in this work, which involve pseudo-register renaming and several loop modifications, are able to resolve several on-path and branch hazards. To reduce

performance penalty, the compiler techniques are made to work in conjunction with hardware-level fault tolerance scheme in the form of read buffer which enables instruction rollback. The read buffer can be used to resolve on-path hazards and, through assembly language modification by compiler, also several branch hazards.

Another study implemented algorithm-based checking with the help of compiler techniques [48]. The compiler in the proposed framework is used to identify linear transformations in Fortran DO loops and insert system level checks by making use of the linear transformations in the code. The compiler technique is also capable of restructuring program statements to change non-linear transformations into linear ones, and therefore, add more opportunities for system-level check insertions.

Another work proposes the insertion of Software Implemented Hardware Fault Tolerance (SIHFT) algorithms during compilation process in order to protect array variables and pointer-referenced array variables [49]. Coding scheme is used to detect or correct errors that may inflict the memory locations of the variables. Each protected variable is to be stored as sequences of bytes which is stored along with a checksum that enables error recovery. The proposed strategy is implemented in GCC compiler [49, 50]. The strategy is implemented in [51] through modification of gimple representation of input source code by adding a new stage of compilation process in cc1, which is the C compiler of GCC.

A work in [52] uses compiler to analyze the Control Flow Graph (CFG) of an input program in order to determine the placement of assertions which are used to check and detect Control Flow Errors (CFEs). The assertions are based on Signature Monitoring (SM) strategy, in which a unique signature is matched to each basic block. The signature is used to check the correctness of control flow by comparing runtime signatures with the signatures associated with basic blocks during program execution. The compiler technique in this work is used to place the assertions for SM in the optimal locations for reducing overheads.

Another study implements compiler technique to detect errors in distributed memory-based parallel programs [53]. Single-Program Multiple-Data (SPMD) execution model is used as a means to detect faults that may inflict processors that work in parallel in a distributed memory system. In this scheme, a selected data item is duplicated, and

every time a value of the data is computed, the processors that own the duplicates are also checked for errors.

3. COMPILER ENHANCED RELIABILITY FRAMEWORK FOR NoC ARCHITECTURES

In this study, we propose a reliability framework for NoCs which comprises a compiler support and a partially-protected NoC-based multicore architecture. The compiler support is used to identify critical data in an input application. During the execution of the application, any packet transmission between the cores (or nodes) which carries critical data, is routed to use as many protected links as possible given a constraint of length (length of shortest path by default).

Through the compiler support, the programmer will be able to mark continuous array regions or scalar variables on an input application which are considered critical. The compiler will propagate the criticality to other array elements or scalar variables through backward program slicing [54].

The final set of critical data resulting from the backward slicing is given as an input to a runtime system which manages the routing strategy of the NoC architecture. Any packet which carries any fraction of the critical data will be transmitted by following a route which maximizes the fault tolerance of the transmission, while the rest of other packet transmissions will follow the default XY-routing.

The workflow of this framework can be seen in Figure 3.1.



Figure 3.1. The workflow of the compiler-enhanced reliability framework

3.1 Compiler Part of Our Framework

Source-to-source compiler needed in this work was implemented by using the concept of polyhedral compilation [55, 56]. The main idea is to convert parts of program that are to be optimized or parallelized by converting them into polyhedral model. Parts of program that can be represented in polyhedral model are called Static Control Parts

(SCoPs) [55]. The conversion of SCoPs in input program into polyhedral form and the manipulation of the polyhedral representations for optimization, parallelization, and code generation are facilitated by several polyhedral compilation framework libraries and a parallelization program named Pluto [21-23].

In order to enable the compiler support to meet the capabilities required in this work, it is necessary to modify the source code of Pluto parallelizer [21-23] and the source code of some polyhedral framework libraries which are utilized by Pluto. Through the modification of the source code, Pluto becomes able to perform several functions which are needed for the whole compiler-hardware fault tolerance method to work. First, Pluto will be able to recognize pragma annotations added by programmer to mark critical scalar variables or array regions. Second, Pluto will propagate the criticality from the originally annotated variables and array elements marked by programmer to other variables and array elements which assign values to the originally annotated variables and array elements. Finally, by using the information given by the compiler support about the final set of critical variables and array regions resulting from criticality propagation, a runtime system will be able to identify critical communications on the hardware layer.

3.1.1 Polyhedral Model

To perform a polyhedral model-based compilation, there are four types of polyhedral representation that need to be extracted from each program statement in a SCoP. The first representation is iteration domain, which contains information on iteration boundaries and the loop strides of the loops that encapsulate the statement. The second representation is access function of each variable reference in the statement. Access function is the affine algebraic function which defines how an array variable is accessed in the statement. Access function combined with iteration domain give information about the array regions that are accessed in the statement. The third representation is the scattering function of the statement. A scattering function can be interpreted in different ways. It can define the schedule of statement instances in a SCoP or define the allocation of the statement instances among the processing nodes in the hardware architecture. The fourth representation that can be extracted from statements in a SCoP is the dependencies among the statements. Polyhedral representation of schedule combined with polyhedral representation of dependencies, together, can be used for code

generation from polyhedral forms without violating dependencies between statements.

The polyhedral representations necessary for polyhedral model-based compilation can be further explained and illustrated as follows.

3.1.1.1 Iteration Domain

Iteration domain of a statement describes the boundaries of iterations that are reached by the statement. Iteration domain of a statement is defined by the conditional statements and the control statements of loops which enclose the statement. Iteration domain can be illustrated by the example in figure 3.2.

$$\begin{array}{l}
 \text{for (} i = 1; i \leq n; i++) \\
 \quad \text{for (} k = 1; k \leq n; k++) \\
 \quad \quad \text{if (} i \leq n+k-3) \\
 \quad \quad \quad s[i] = \dots
 \end{array}
 \quad
 D_{S1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & 1 & 1 & -3 \end{bmatrix} \cdot \begin{pmatrix} i \\ k \\ n \\ 1 \end{pmatrix} \geq \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

Figure 3.2. Iteration domain

The matrix-vector multiplication on the right part displays the iteration domain of the statement enclosed by the loop nest and conditional statement on the left. The multiplication between the first line of the matrix and the vector represents the inequation $i - 1 \geq 0$, which is the lower boundary of the outer loop. The multiplication between the second line in matrix and the vector is for the inequation $-i + n \geq 0$, which is the upper boundary of the outer loop. The result of multiplication between the third line of the matrix and the vector gives the lower boundary of the inner loop, which is inequation $k - 1 \geq 0$. The multiplication between the fourth line and the vector results in inequation $-k + n \geq 0$, which is the upper boundary of the inner loop. Finally, the multiplication between the fifth line and the vector represents the inequation that represents the condition of the if statement on the left, which is $-i + k + n - 3 \geq 0$.

3.1.1.2 Access Function

An access function in a statement is the affine function of loop invariants and other scalar variables which defines the indexes of array cells of the array variable that are accessed in that statement. An example of access function is in figure 3.3.

<pre> for (i = 1; i <= n; i++){ s[i] = 0; for(k = 0; k < n; k++){ s[i] = s[i] + a[i][k]*x[i+k]; } </pre>	$f_s(\vec{x}_{s2}) = [1 \ 0 \ 0 \ 0] \cdot \begin{pmatrix} i \\ k \\ n \\ 1 \end{pmatrix}$
	$f_a(\vec{x}_{s2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} i \\ k \\ n \\ 1 \end{pmatrix}$
	$f_x(\vec{x}_{s2}) = [1 \ 1 \ 0 \ 0] \cdot \begin{pmatrix} i \\ k \\ n \\ 1 \end{pmatrix}$

Figure 3.3. Access Function

The affine functions on the right are the access functions of the three variable references on the statement “s[i] = s[i] + a[i][k] * x[i+k]”. The first affine function represents i, which is the index of array reference s[i]. The second affine function represents two separate indexes i and k, which are the indexes of array reference a[i][k]. The third affine function is for the index of the third array reference, which is x[i+k].

3.1.1.3 Scattering Function

Scattering function is used to represent ordering information in polyhedral model [57]. The ordering information can be of any kind, for instance space mapping and time mapping. Space mapping or allocation refers to the decision about on which processing nodes instances of a statement have to be executed. An example of the use of scattering function for space mapping is illustrated as follow.

Given that there are the following statements in the source code.

$$f = b * c; \ //S1$$

$$a = b + d; \ //S2$$

$$x = a - d; \ //S3$$

The allocation or space mapping function of the above statements is as follows.

$$P_{S1} = 2$$

$$P_{S2} = 1$$

$$P_{S3} = 1$$

According to the allocation functions above, statement S1 must be executed in processing node 2, while statements S2 and S3 must be executed in processing node 1.

Scattering function can also be used to define time mapping. Time mapping or scheduling refers to the decision regarding the logical date at which instances of statements have to be executed. An example of a simple scheduling function is shown below.

$$T_{S1} = 1$$

$$T_{S2} = 2$$

$$T_{S3} = 1$$

From the simple scheduling functions above, it can be concluded that statements S1 and S3 can be executed in parallel, while statement S2 must be executed after both statement S1 and S3.

3.1.1.4 Polyhedral Representation for Data Dependence

Data dependence existing between two statements can be represented in polyhedral form. The polyhedral representation of a dependence must contain several elements. First, it must contain the condition under which the dependence applies. This condition is represented by an algebraic equation which involves the loop iterators which are used in the access function of the variable on which the dependence exists between the two statements. Second, the polyhedral representation also contains the iteration domain of the source statement. Third, it must also contain the iteration domain of the target statement. An example of polyhedral representation of a dependence between two statements is in Figure 3.4.

```

for (i=1; i<=3; ++i) {
. s[i] = 0;
. for (j=1; j<=3; ++j)
. . s[i] = s[i] + 1;
}

```

$$\mathcal{D}_{S1\delta S2} : \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \cdot \begin{pmatrix} i_{S1} \\ i_{S2} \\ j_{S2} \\ 1 \end{pmatrix} \begin{matrix} = 0 \\ \geq 0 \end{matrix}$$

Figure 3.4. Data dependence between statement 1 and 2 on variable s

The matrix-vector multiplication on the right part is the polyhedral form of the dependence which exists between statement $s[i] = 0$ and statement $s[i] = s[i] + i$. The multiplication between the first line of the matrix and the vector represents the condition under which the dependence exists. This multiplication results in an equation, which is $i_{s1} - i_{s2} = 0$ in this case. It means that dependence between both statements exist only when the indexes of variable s in both statements are similar. Different from the multiplication between the first line and the vector which resembles an equation, the multiplications between the other lines in the matrix and the vector represent in-equations as they constitute the iteration domains of the source statement and the target statement. The second and third lines of the matrix are for the iteration domain of the source statement and the rest of the matrix is for representing the iteration domain of the target statement.

3.1.2 Design of Compiler Support

The compiler support for the fault tolerance framework proposed in this work was developed by modifying several polyhedral compilation libraries and a parallelizer. The libraries which were modified are OpenScop [57], which defines the polyhedral representations that are used in the compilation, and Clan [24], which provides lexical and syntax analyzers which convert a SCoP into polyhedral representations that are defined by OpenScop. Propagation of criticality and decision on the placement of fault tolerance schemes on the hardware architecture is enabled in the compiler support by modifying the source code of Pluto parallelizer [21-23].

3.1.2.1 Representation for Critical Region

OpenScop library [57] specifies the main data structures that are used in most of the phases of polyhedral analysis and code optimizations in the source-to-source compiler. Therefore, in order to enable critical array regions to be represented in the data structures, there are several modifications which need to be added in the source code of OpenScop. New macros for data structures that contain information on critical regions were added to OpenScop library. *osl_relation_struct*, which is the data structure in OpenScop that is used to represent a matrix, is used to represent critical array region. The information on critical array regions is added as an attribute to each statement that operates on and propagates the critical regions.

3.1.2.2 Extraction of Polyhedral Representation from Source Code

Clan library [24] is used for extracting information for polyhedral analysis from source code. The information extracted is converted into data structures that have been specified in OpenScop. Since in this work a criticality propagation analysis needs to be carried out, an initial annotation for assigning criticality to certain variables or array regions is needed for each SCoP. This annotation will be enabled by Clan, since it is this library that provides front-end functions which extract polyhedral forms of SCoPs in input source code. To enable Clan to accept this criticality annotation, the parts of Clan source code which contain its lexical and syntax analyzers were modified. Therefore, slight changes were made to the syntax and lexical analyzers, which are based on yacc [58] and lex [59], so that Clan can accept criticality annotation pragma with the following grammar.

Table 3.1. Syntax of criticality pragma

```
<annotation> ::= "#pragma critical" <critical_set>
<critical_set> ::= '{' <critical_data> '}'
<critical_data> ::= <critical_vars> |
<critical_vars> ::= <critical_var> | <critical_vars> ',' <critical_var>
<critical_var> ::= ID '(' <critical_region> ')'
<critical_region> ::= <regions> |
<regions> ::= <region> | <regions> ',' <region>
<region> ::= '[' <range_set> ']'
<range_set> ::= <ranges> |
<ranges> ::= <range> | <ranges> ',' <range>
<range> ::= INTEGER ':' INTEGER
```

In the grammar above, *ID* represents a variable's name and *INTEGER* represents an integer regardless of the number of its digits.

The *<annotation>* nonterminal in the grammar is to be placed right after *#pragma scop*, which marks the beginning of a SCoP in the input source code which is scanned by Clan library, and before the SCoP. Therefore, the annotation pragma is to be put at the beginning of each SCoP that is enclosed between *#pragma scop* and *#pragma endscop*. An example is illustrated in Figure 3.5 In the example, there are two continuous regions in array C which are marked as critical by programmer. The first region consists of any array element whose first index is within the range [2..6] and whose second index is within the range [5..55]. The included elements in the second region, are those whose first index in the range [19..23] and whose second index in the range [45..75].

```

1. #pragma scop
2. #pragma critical {C([2:6,5:55],[19:23,45:75])}
3. for (i = 2; i < 80; i++)
4.   for (j=0; j < 80; j++) {
5.     B[i-2][j+3] = A[i][j];
6.   }
7. for (i=0; i<80; i++)
8.   for (j=0; j<80; j++) {
9.     C[i+1][j+2] = B[i][j];
10.  }
11. #pragma endscop

```

Figure 3.5. Example of criticality pragma

During the lexical and syntax analysis phases of the compilation, the information in the annotation is converted into a linked list of *osl_generic* data structure which can contain a list of *osl_relation* data structure. Each *osl_relation* data structure in the *osl_generic* contains the polyhedral form of a critical region of a variable. The created *osl_generic* linked list is then attached to *osl_scop* data structure which is the end product of lexical and syntax analysis by Clan.

3.1.2.3 Propagation of Criticality

The goal of criticality propagation is to spread the criticality annotations to the variables or array elements in the input application which affect or are affected by the initial variables or array regions that are marked by programmer. This propagation follows the concept of program slicing [54]. It is enabled by modifying the source code of Pluto parallelizer [21-23]. Functions defined in Polylib library [60] are put into the source code of Pluto parallelizer in order to facilitate the propagation by providing necessary algebraic operations.

The type of program slicing applied to the compiler which support criticality propagation is backward slicing. Backward slicing phase propagates criticality from the scalar variables and array regions which are already marked manually. Backward slicing works as follow. At first, the criticality information contained in *osl_generic* extensions of *osl_scop* is again converted into a set of Polylib polyhedral representations. After that,

for each statement, criticality information is propagated from write variables to read variables. After the propagation is performed, the regions of the read array and scalar variables which affect the values of the write scalar or array variable in the statement are also marked as critical, and will be included in the set of critical regions that is used for propagation in the statements which are located above the current statement.

In the Figure 3.5, backward slicing tracks the propagation of criticality from array C to array B in line number 9 and from array B to array A in line number 5. This is due to the assignment operations which causes the value of array B to affect the value of array C due to the statement in line number 9 and causes the value of array A to affect the value of array B due to assignment statement in line number 5. Due to backward slicing, some elements of array B and array A are also marked as critical. Elements of array B which are categorized as critical are those with first index in the range [1..5] and second index in the range [3..53], and those with first index in the range [18..22] and second index in the range [43..73]. The elements of array A that are considered as critical are the elements with first index in the range [3..7] and second index in the range [0..50], and the elements with first index in the range [20..24] and second index in the range [40..70].

3.1.2.4 Generation of Parallel Code

Parallelized code that is produced by the source-to-source compiler designed in this work is generated with Pluto parallelizer. The command line options for compilation with Pluto are "--parallel". These options instruct Pluto to parallelize the input source code into OpenMP pragma annotated code.

3.2 Tracking and Protecting Critical Communications on Hardware Level

The output of compiler layer is used in identifying critical communications in the transport layer of NoC. If a packet transmission is found to carry critical data, that packet is routed to cross as many protected links as possible without violating west-first routing algorithm and without exceeding the length constraint of any routes between the two communicating nodes. The default length constraint is the length of shortest routes between the two communicating nodes. For other packets which don't contain critical

data, XY-routing is used in leading the packets from their sender nodes to their destination nodes.

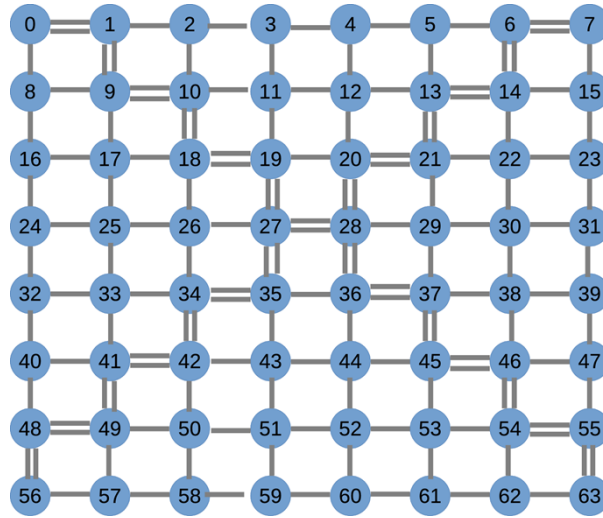


Figure 3.6. A partially protected 8x8 NoC which has 27 protected links

In this study, we consider an 8x8 NoC architecture which has a 2D-mesh-based topology with 64 nodes (see Figure 3.6). A router which is connected to the network interface of a processing unit is resembled by a node in the topology. Since a processing unit is matched exactly with one router, there are 64 processing units/cores and 64 routers in the entire architecture of the system. There are two types of links used in the NoC architecture; *protected links* and *unprotected links*. *Protected links* are illustrated by double lines and *unprotected links* are illustrated by single lines in Figure 3.6.

To give a clear idea of how the routing strategy on critical data transmission works, assume that a packet is sent from *node 1* to *node 21*. As shown in Figure 3.7, under the XY-routing strategy, the following path is determined: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 13 \Rightarrow 21$. The paths produced by XY routing are always shortest paths, and in this case, the length of the path is 6 edges. However, if there is a critical data within the packet, a different strategy is adopted to produce another shortest path between sender and destination nodes, which has the maximum number of protected links in the path. From this point on, this routing strategy is called *fault-tolerant routing* and the routes that it generates are called *fault-tolerant routes*. For packet transmission from *node 1* to *node 21*, the generated path becomes $1 \Rightarrow 9 \Rightarrow 10 \Rightarrow 18 \Rightarrow 19 \Rightarrow 20 \Rightarrow 21$. The length of this path is similar with the length of the path generated by XY-routing. However, the

number of protected links contained in this path is higher. In this path, there are 5 protected links and 1 unprotected link (see Figure 3.8), while the path generated by XY-routing only contains 1 protected link and 5 unprotected links.

The fault tolerance of a path can be improved by adding more protected links in the path through increasing the upper-bound of path length. Let's assume that n is the length of shortest paths between a pair of nodes. Then, the fault tolerance of the paths that are generated for communication between the two nodes can be increased by allowing the length of the generated path to be $n + x$, with x being the difference between the upper bound of path length and the length of shortest paths between the two nodes. Since the tolerable length is increased, the *fault-tolerant routing* strategy is able to add more protected links to the paths that it generates. For instance, if $x = 2$, the number of links included in the transmission from *node 1* to *node 21* becomes 8 (see Figure 3.9). The generated path follows the route $1 \Rightarrow 9 \Rightarrow 10 \Rightarrow 18 \Rightarrow 19 \Rightarrow 27 \Rightarrow 28 \Rightarrow 20 \Rightarrow 21$, which is a route that has a higher proportion of protected links. The proportion of protected links is increased from $5/6$ in Figure 3.8 to 1 in Figure 3.9.

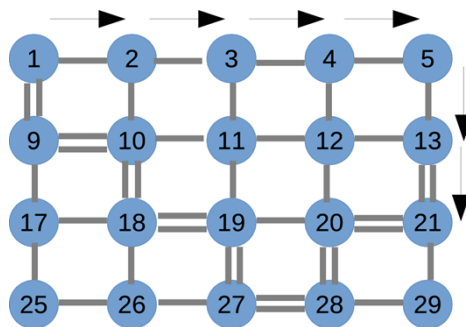


Figure 3.7. XY route on partially protected NoC

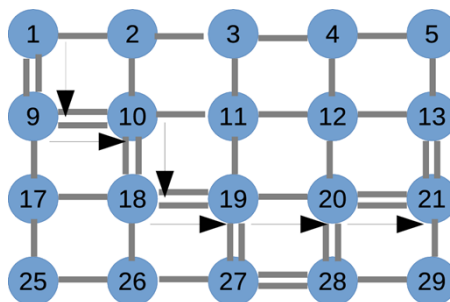


Figure 3.8. Fault-tolerant route with $x=0$

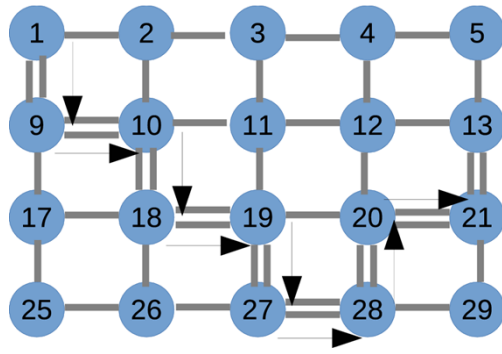


Figure 3.9. Fault-tolerant route with $x=2$

4. EXPERIMENTAL STUDY

The experiments to evaluate the framework that is developed in this thesis are carried out by using Pluto parallelizer [21-23] on which the compiler part of the framework is developed and Sniper multicore simulator [61]. The experimental study consists of three phases as explained in the introduction section.

Several benchmarks are prepared to test program slicing performed by the compiler framework and to evaluate the whole framework by running the output of the compiler part on the simulated partially protected NoC hardware in Sniper.

The following subsection will explain the Sniper simulator, the parameters of the simulated hardware, the compared protection schemes, and finally the benchmarks.

4.1 Setup for Experiments

4.1.1 Sniper Simulator

Sniper simulator [61] is a simulator which has the ability to simulate parallel computer architectures along with the bus or network-on-chip connectors which are used for communication among the processing nodes.

The simulator has interval core model, which enables higher simulation speed as it allows jumps between miss events. The core model empowers Sniper to simulate the execution of multi-programmed, multi-processed, or multi-threaded workloads on a parallel architecture which has tens or 100+ processing nodes at high speed. This core model trades off speed with accuracy as it causes Sniper to have less accuracy but higher speed than cycle accurate simulator, yet Sniper is still more accurate than one IPC simulators. The accuracy of the Sniper's core model has been evaluated by having simulations in Sniper compared with real executions in Intel Core2 and Nehalem systems. It is shown that Sniper yields “average performance prediction errors within 25% at a simulation speed of up to several MIPS” [62].

4.1.2 Configuration of Simulated Architecture

The architecture considered in the experiments is a multicore with 8-by-8 2D mesh-based network-on-chip as the underlying interconnecting system which regulates the communication among the processing cores (see Figure 3.6). The network-on-chip implements virtual-cut-through switching [63]. The simulated processing units in the system are Xeon X5550 Gainestown, and have out-of-order execution mode with reorder buffer. Each processing unit has local L1, L2 caches, and a shared static NUCA cache in the last level. The bandwidth of links in NoC is 64 bits. The latency of one hop through a link is 2 clock cycles.

The placement of *protected links* and *unprotected links* is following the topology in Figure 3.6. In *protected links*, every flit is protected with hop-by-hop Single Error Correction (SEC) coding scheme, while in *unprotected links* the coding scheme is only applied to flits which carry packet headers (header flits). The decoding of SEC code incurs a single clock cycle overhead for every hop. This one clock cycle overhead is in accordance with [26], which considers SEC as one of the coding schemes that can process 100 to 200 bit wide buses in one clock cycle.

The packets transmitted across the simulated NoC can have two different sizes. The first size is 2 flits (rounded from 14 bytes), as the packet contains only the necessary information for communication between two cores. The second size is 10 flits (rounded from 14 + 64 bytes), as the packet does not only contain information for communication, but also contains the payload/memory line that is requested by destination core.

4.1.3 Compared Fault Tolerance Schemes

In the first experiment, there are three schemes which will be compared. These schemes are the *XY-routing scheme*, the *full fault-tolerant routing scheme*, the *XY-routing scheme with hop-by-hop single error correction (XY-routing with HBH-SEC)*, and the scheme proposed in this work which is referred to as *critical transmission only scheme*.

XY-routing scheme refers to the ordinary dimensional order routing strategy which works on the partially protected NoC architecture as shown in Figure 3.6. The

routing strategy doesn't consider the differentiation between *protected links* and *unprotected links* at all.

In *fault-tolerant routing scheme*, fault-tolerant routing as illustrated in Figure 3.7 is applied to all communications among different nodes. This scheme doesn't differentiate between critical and non-critical communications since all communications are treated as if they were critical and, therefore, require protection.

In *XY-routing with HBH-SEC* scheme, XY-routing is used as the routing algorithm but protection against soft errors is implemented on each link in the NoC and for each flit, i.e. every link is a *protected link*. The protection is implemented by using Error Correcting Code (ECC) which can detect and correct single bit flip. The ECC is encoded and decoded for every hop of flit which follows the examples in [16] and [26].

Critical-transmission-only scheme is the proposed fault-tolerant scheme that differentiates between critical and non-critical communications. *Critical communications* are defined as transmissions which carry data that have been marked as critical by compiler support. In this scheme, fault-tolerant routing is applied only to critical communications.

4.1.4 Benchmarks

In the experimental study, there are eight applications which are selected from Polybench benchmark collection [64]. The reason for selecting this benchmark collection suite is that the benchmarks contained in this suite have Static Control Parts which can be processed by polyhedral compiler such as Pluto. The applications that are selected for the experimental study are listed in Table 4.1. The experimental study is divided into three phases. All eight of the applications are used in the first and third phases, while only two of them, which are *covariance* and *gesummv*, are used in the second phase. Each of these two applications are assigned with different forms of criticality annotations to mark parts of their computation results that are considered as critical. The purpose of the second phase of the experimental study is to evaluate how the annotations impact on the performance and fault tolerance of the applications.

Table 4.1. Applications considered in experimental study

Benchmark	Description	Parameter Sizes
Gemver	vector multiplication and matrix addition	$N = 500$
Atax	matrix transpose and vector multiplication	$NX = 500, NY = 500$
Bicg	BiCG sub kernel of BiCGStab linear solver	$NX = 500, NY = 500$
Gesummv	scalar, vector and matrix multiplication	$NX = 500, NY = 500$
Mvt	matrix vector product and transpose	$N = 500$
Trisolv	triangular solver	$N = 500$
Gramschmidt	Gram-Schmidt decomposition	$NI = 128, NJ = 128$
Covariance	covariance computation	$N = 500, M = 500$

4.1.5 Criticality Annotations with Compiler Support

To define data which are to be considered as critical by the compiler support, some regions in the arrays which contain computation results that need to be protected must be marked manually by the programmers. In the experimental study, the arrays that are marked in each application are the arrays which display the final computation result in the SCoP area of the source code. From here on, these arrays are referred to as the **result arrays**.

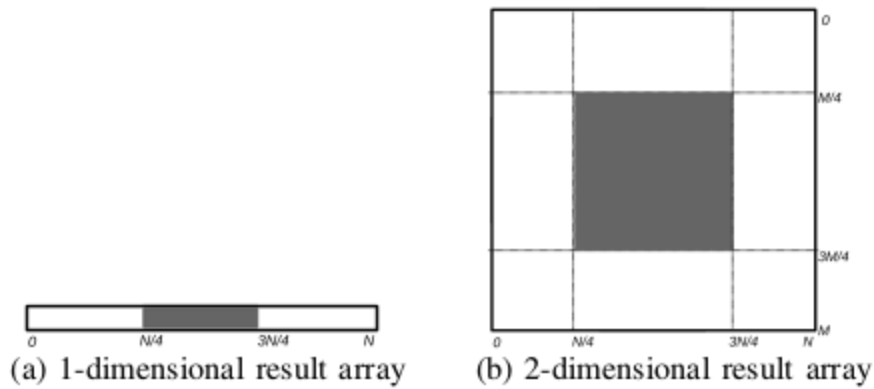


Figure 4.1. Manual criticality annotations on 1-dimensional and 2-dimensional result arrays

In the first and third phases of experimental study, the part of the result array of each application that is marked as critical is the middle part of the array. In case the result array is a one-dimensional array with length N , the marked region consists of the elements within the range $[(N/4)..(3N/4-1)]$; and a similar marking is also used for the 2 dimensions of 2-dimensional result arrays (see Figure 4.1).

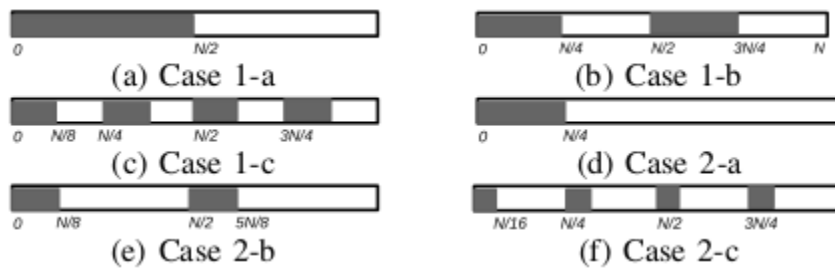


Figure 4.2. Manual criticality annotations on 1-dimensional result array of gesummv application for the second experiment

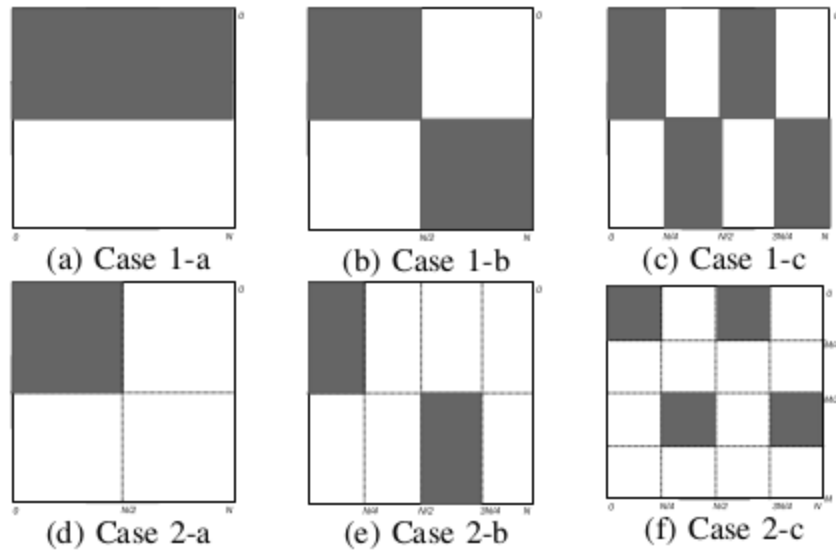


Figure 4.3. Manual criticality annotations on 2-dimensional result array of covariance application for the second experiment

For the second phase of the experimental study, the result arrays of the tested applications are subjected to different forms of annotations. Six different forms of criticality annotations are applied to each application. These forms are illustrated in Figure 4.2. In the first three cases, only half of the result arrays are annotated, yet the distribution and shapes of criticality marking in each case is different. In the last three cases, the size of criticality markings is a quarter of the result array. These three cases also have different distribution and shapes of criticality marking. The similar strategy of annotations is also applied to 2-dimensional result arrays as shown in Figure 4.3.

From the result arrays, criticality is propagated within the SCoP of the source code to all of the other scalar variables and array elements which assign values to the variables and array elements that are already marked as critical. The final output of the compiler support, which is the final set of scalar variables and array elements that are marked as critical, is provided as an input for the runtime system of Network-on-Chip. By using this information, it becomes possible to differentiate packet transmissions which carry critical data from those who don't.

As the goal of the second experiment is to evaluate the effects of different sizes, distributions, and shapes of criticality markings on some applications, it is necessary to identify and measure the sizes of the sets of critical array elements and scalar variables produced by the compiler support. The sizes of these sets are expected to have impact on

the performance and reliability of the applications. Figure 4.4 and Figure 4.5 show the SCoPs of *covariance* and *gesummv* kernels which are used in the second experiment. The sets of critical array elements which are annotated manually and the sizes of the final sets of critical data after backward slicing are listed in Table 4.2.

```
1. #pragma scop
2. for (j = 0; j < 500; j++)
3.   {
4.     mean[j] = 0.0;
5.     for (i = 0; i < 500; i++)
6.       mean[j] += data[i][j];
7.     mean[j] /= float_n;
8.   }
9. for (i = 0; i < 500; i++)
10.  for (j = 0; j < 500; j++)
11.    data[i][j] -= mean[j];
12. for (j1 = 0; j1 < 500; j1++)
13.  for (j2 = j1; j2 < 500; j2++)
14.    {
15.      symmat[j1][j2] = 0.0;
16.      for (i = 0; i < 500; i++)
17.        symmat[j1][j2] += data[i][j1] * data[i][j2];
18.      symmat[j2][j1] = symmat[j1][j2];
19.    }
20. #pragma endscop
```

Figure 4.4. SCoP of covariance benchmark

```
1. #pragma scop
2.   for (i = 0; i < 500; i++)
3.   {
4.     tmp[i] = 0;
5.     y[i] = 0;
6.     for (j = 0; j < 500; j++)
7.     {
8.       tmp[i] = A[i][j] * x[j] + tmp[i];
9.       y[i] = B[i][j] * x[j] + y[i];
10.    }
11.    y[i] = alpha * tmp[i] + beta * y[i];
12.  }
13. #pragma endscoP
```

Figure 4.5. SCoP of gesummv benchmark

Table 4.2. Results of program slicing of criticality for the second experiment

Cases	Manually Annotated Variables/Array Elements	Number of Final Critical Variables / Array Elements
Case 1-a (covariance)	symmat[0-249][0-499]	375501
Case 1-b (covariance)	symmat[0-249][0-249], symmat[250-499][250-499]	375501
Case 1-c (covariance)	symmat[0-249][0-124], symmat[0-249][250-374], symmat[250-499][125-249], symmat[250-499][375-499]	438001
Case 2-a (covariance)	symmat[0-249][0-249]	187751
Case 2-b (covariance)	symmat[0-249][0-124], symmat[250-499][250-374]	344251
Case 2-c (covariance)	symmat[0-124][0-124], symmat[0-124][250-374], symmat[250-374][0-124], symmat[250-374][250-374]	187751
Case 1-a (gesummv)	y[0-249]	251002
Case 1-b (gesummv)	y[0-124], y[250-374]	251002
Case 1-c (gesummv)	y[0-61], y[125-186], y[250-312], y[375-437]	251002
Case 2-a (gesummv)	y[0-124]	125752
Case 2-b (gesummv)	y[0-61], y[250-312]	125752
Case 2-c (gesummv)	y[0-30], y[125-155], y[250-280], y[375-406]	125752

4.1.6 Fault Injection Framework

To evaluate the fault tolerance of the proposed framework in comparison to other schemes, under different criticality markings, and under different length constraints, fault injection is performed throughout the experimental study. The injected faults are single bit flips, each of which is injected in every interval of 10 packet transmissions. The single bit flip simulates with either the MAF crosstalk model [32] or Single Event Upset (SEU) in input buffers.

4.1.7 Measurement Metrics

The goal of the experimental study is to evaluate the introduced scheme with respect to three aspects; performance, reliability, and energy consumption.

To measure the performance of the scheme, *total packet latency* and *application execution time* are used. *Total packet latency* is the sum of the time taken for each individual packet transmission, while *application execution time* refers to the duration of application execution.

In order to measure reliability, four metrics are used. The first metric is the *proportion of protected links on critical communications*. This metric is measured by dividing the number of times protected links are crossed in critical communications with the number of times any links are crossed in critical communications. The second metric to measure reliability is the *proportion of erroneous bytes on critical communications*. This metric is calculated by dividing the number of bytes in the entire critical packet payloads which suffer from errors with the number of bytes in the entire critical packet payloads. These first and second metrics are intended to measure the fault tolerance of critical communications. The third metric to measure reliability is the *proportion of protected links on all communications*. This metric is measured by dividing the number of times protected links are used in all communications with the number of times any links are used in all communications. The fourth metric for measuring reliability is the *proportion of erroneous bytes on all communications*. This metric is calculated by dividing the number of bytes in the entire packet payloads which suffer from errors with the number of bytes in the entire packet payloads. The last two metrics are intended to measure the fault tolerance of the entire packet transmissions.

McPAT [65] is used as the tool to measure simulated energy consumption for each application execution in the experimental study. The unit of measured energy consumption is Joule.

4.2 Experimental Results and Discussion

As mentioned in the previous subsection, the experimental study is divided into three phases. The first phase compares our proposed scheme, which is referred to as *critical transmission only fault-tolerant routing* scheme, with two other schemes which apply oblivious routing strategy, i.e. XY routing, and apply fault-tolerant routing for all packet transmissions. The second phase evaluates the impact of different annotation forms on performance and reliability of application execution. The third phase evaluates the effects of increasing the upper bound of path length for the proposed scheme.

4.2.1 Evaluating Performance and Fault Tolerance of The Schemes

In the first phase of experimental study, the performance and fault tolerance of three different schemes are compared. These schemes are the *XY-routing*, the *full-fault-tolerant routing*, *XY-routing with HBH-SEC* and the *critical transmission only fault-tolerant routing* schemes. These routing schemes are run on partially-protected NoC architecture as depicted in Figure 4.6.

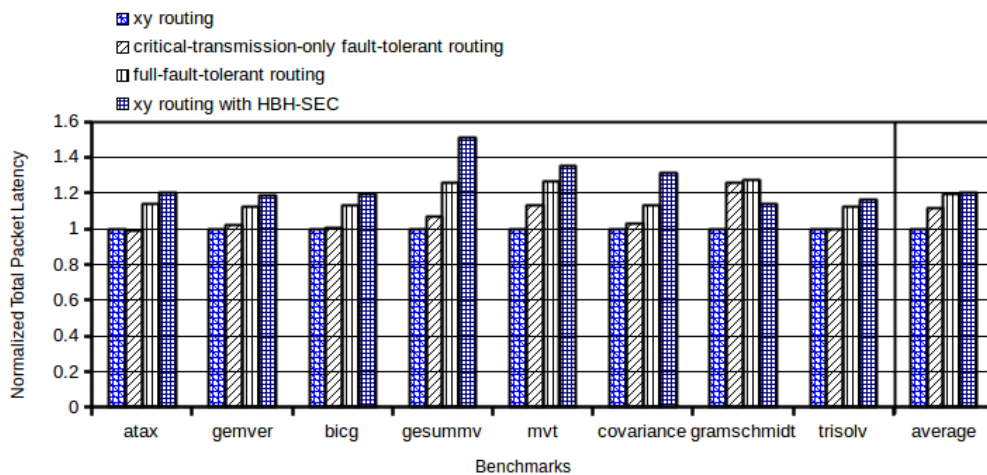


Figure 4.6. Total Packet Latency of Compared Schemes

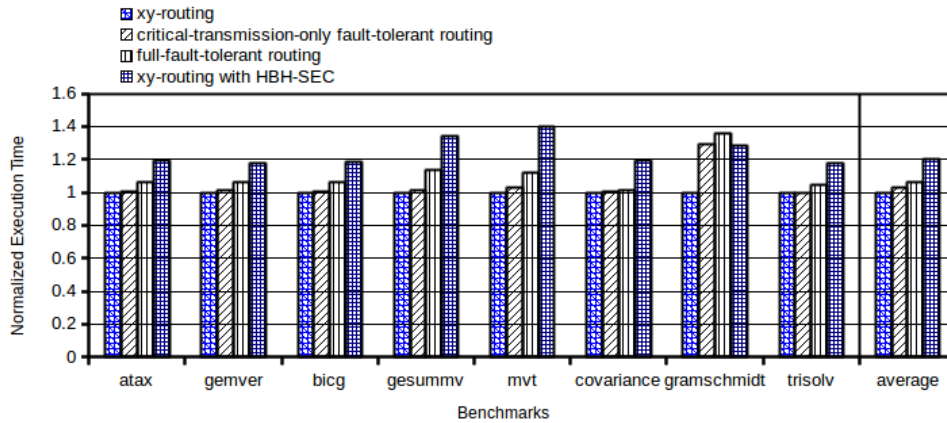


Figure 4.7. Application Execution Time of Compared Schemes

Figure 4.6 displays the comparison of performance of the four schemes in terms of total packet latency. The displayed metric is a normalized metric which is computed by dividing the total packet latency of each compared scheme with the total packet latency of the XY routing scheme. This normalization method is chosen because the XY routing scheme is expected to have the lowest latency among the compared schemes and, hence, the results of normalization are expected to never be lower than 1. XY routing scheme displays the best performance among the compared schemes for all of the tested benchmarks. This is expected, since *critical-transmission-only* scheme implements *fault-tolerant routing* for critical communications and, therefore, increases contention on protected links, while *full-fault-tolerant routing* scheme increases contention on protected links even further as it generates *fault-tolerant routes* for all communications. This contention on *protected links* is what degrades the performance of *critical-transmission-only* and *full-fault-tolerant routing* schemes with respect to XY-routing scheme. Furthermore, for nearly all of the applications, *XY-routing with HBH-SEC* incurs the highest total packet latency among the compared schemes. On average, *critical-transmission-only* scheme yields 12% more total packet latency than *XY routing* scheme, while *full-fault-tolerant routing* scheme incurs 19% more total packet latency and *XY-routing with HBH-SEC* incurs 20% more total packet latency than *XY routing* scheme.

Figure 4.7 shows the performance results in terms of application execution time. Application execution time is also normalized in the graph by dividing the application execution time of each compared scheme with the application execution time of the XY

routing scheme. The comparison results in terms of application execution time are consistent with the comparison results in terms of total packet latency. *XY-routing* scheme has the shortest execution time, *critical-transmission-only* scheme has the second lowest execution time for nearly all applications with average execution time 3% higher than the *XY-routing* scheme, *full fault-tolerant-routing* scheme has the third lowest execution time for nearly all applications which is 6% higher on average than the *XY-routing* scheme, and *XY-routing with HBH-SEC* has the highest application execution time in nearly all applications which is 20% higher than the *XY-routing* scheme.

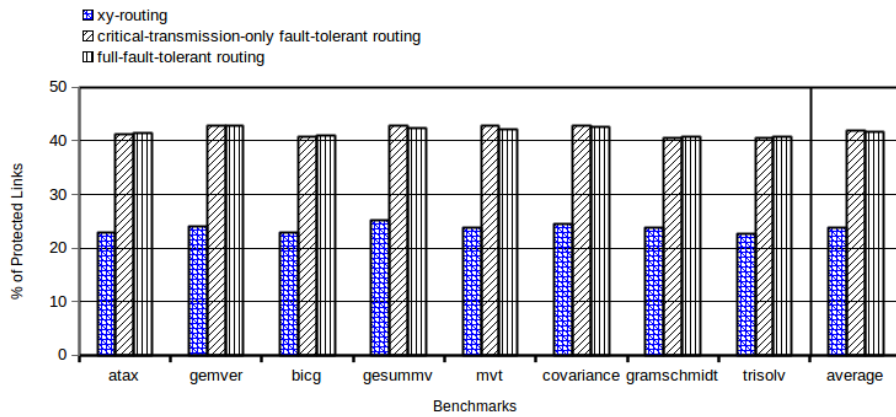


Figure 4.8. Percentage of Protected Links on Critical Communications of Compared Schemes

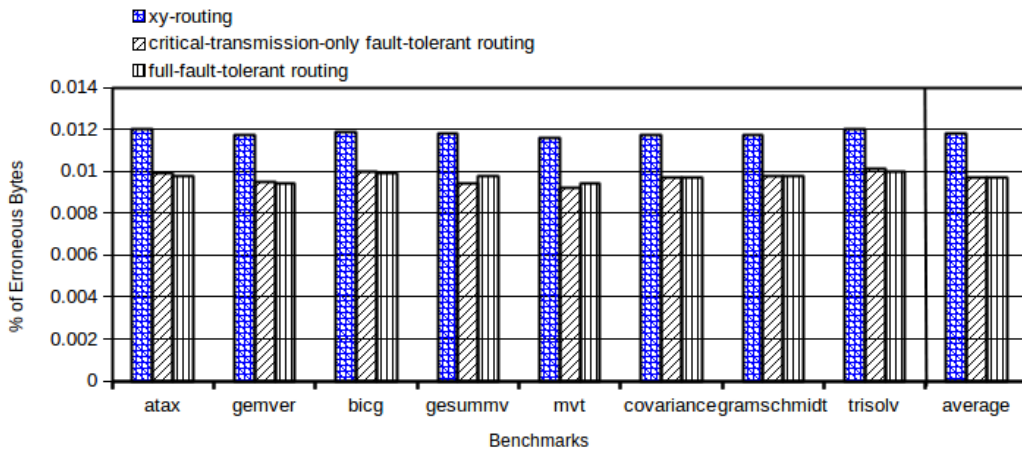


Figure 4.9. Proportion of Erroneous Bytes on Critical Communications

Evaluation of the compared schemes in terms of the fault tolerance of critical communications is shown in Figure 4.8 and Figure 4.9. In this evaluation and in the evaluation which considers the fault tolerance of all communications, *XY-routing with*

HBH-SEC scheme is not included, since this scheme considers all links to be *protected links* and therefore the proportion of protected links is always 1 for all communications.

The first metric that is used for this evaluation is the proportion of protected links on critical communications. According to this metric, *critical-transmission-only* scheme gives a nearly similar level of fault tolerance as the *full fault-tolerant routing* scheme for critical communications. Both schemes also outperform *XY-routing scheme* with respect to this metric, which is expected as *XY-routing scheme* doesn't consider the presence of *protected links* in generating routes. On average, the routes that are generated by critical-transmission-only scheme include 1.76 times more protected links in critical communications than *XY-routing scheme*, and the routes that are generated by *full-fault-tolerant routing* scheme also have a nearly similar value.

The second metric that is used in this evaluation is the *proportion of erroneous bytes on critical communications*. The results of comparison with this metric are consistent with the results of comparison with the *proportion of protected links on critical communications*. *Critical-transmission-only* scheme gives a nearly similar level of fault tolerance on critical communications as the *full fault-tolerant routing* scheme. This conclusion can be drawn since the *proportion of erroneous bytes on critical communications* of both *critical-transmission-only* and *full fault-tolerant routing* schemes is shown to be 0.82 times the *proportion of erroneous bytes on critical communications* of the *XY-routing scheme* on average.

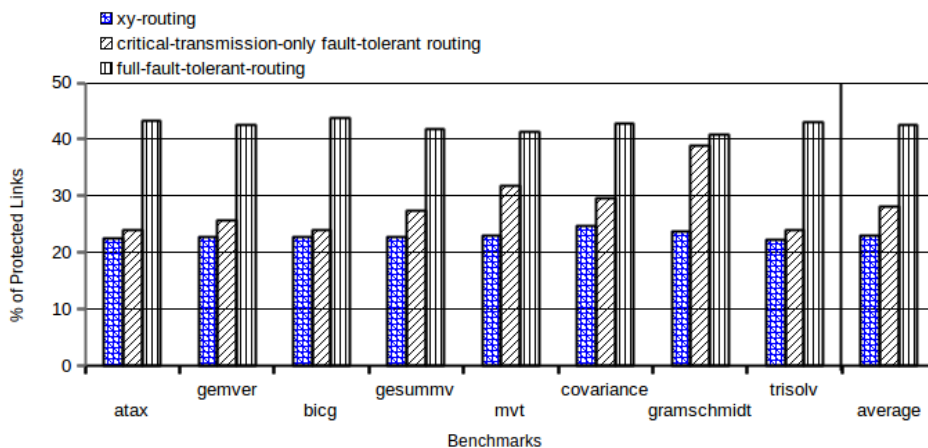


Figure 4.10. Proportion of Protected Links on All Communications of Compared Schemes

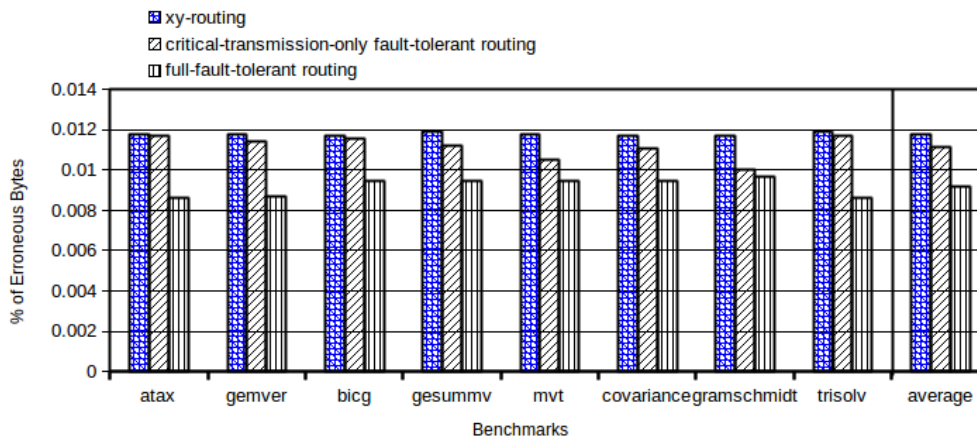


Figure 4.11. Proportion of Erroneous Bytes on All Communications

The difference in terms of *proportion of protected links on all communications* and *proportion of erroneous bytes on all communications* is apparent between *critical-transmission-only* scheme and *full-fault-tolerant routing* scheme as displayed in Figure 4.10 and Figure 4.11. This outcome is expected, since *critical-transmission-only* scheme generates fault-tolerant routes only for packet transmissions which carry critical data, while *full fault-tolerant* scheme generates fault-tolerant routes for all communications. However, the fault tolerance of *critical-transmission-only* scheme is still higher than the fault tolerance of *XY-routing* scheme in terms of *proportion of protected links on all communications* and *proportion of erroneous bytes on all communications*. The routes generated by *critical-transmission-only* scheme, on average, include approximately 1.22 times more protected links in all communications than the routes generated by *XY routing* scheme. Furthermore, the identified *proportion of erroneous bytes* in *critical-transmission-only* scheme is 0.95 times the proportion identified in *XY routing* scheme. The *full fault-tolerant* scheme displays the highest fault tolerance on all communications among the three compared schemes. The routes generated by *full-fault-tolerant* routing scheme include approximately 1.84 times more protected links in all communications than the routes generated by *XY routing* scheme. Additionally, *the proportion of erroneous bytes on all communications* identified in the *full fault-tolerant* scheme is only 0.78 times the proportion identified in the *XY routing* scheme.

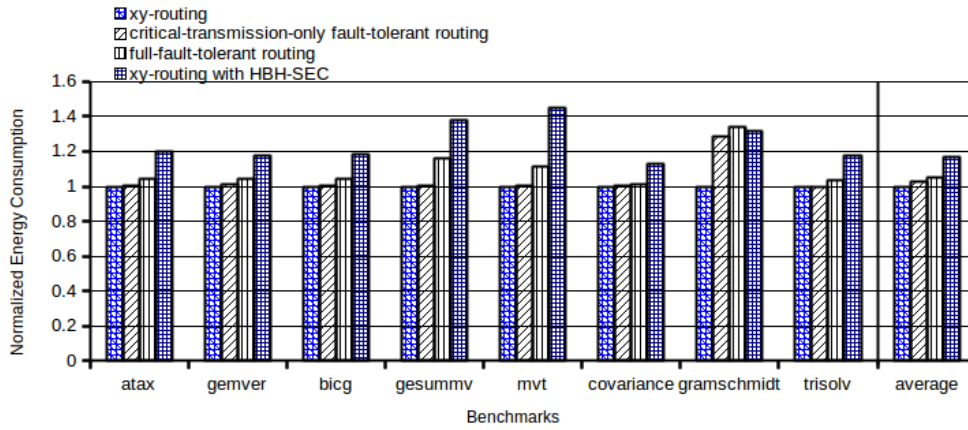


Figure 4.12. Energy Consumption of Compared Schemes

Energy consumption results of XY-routing, critical-transmission-only, full fault-tolerant, and XY-routing with HBH-SEC schemes are displayed in Figure 4.12. The displayed comparison is consistent with the comparison shown in Figure 4.6 and Figure 4.7. *Critical-transmission-only* scheme is shown to consume 3% more energy on average than the XY-routing scheme, full fault-tolerant-routing scheme on average consumes 5% more energy than the XY-routing scheme, and XY-routing scheme with HBH-SEC consumes 17% more energy than the XY-routing scheme. The close similarity between the application execution time graphs in Figure 4.7 and the energy consumption graphs in Figure 4.12 is intuitive since energy consumption is computed by multiplying power with application execution time.

4.2.2 Evaluating Impact of Annotation Sizes and Shapes

The goal of this experiment is to evaluate the effects of different forms of criticality markings on the performance, fault tolerance, and energy consumption of the applications that are running on the *critical-transmission-only* scheme. Figures 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, and 4.19 show the experiment results in terms of performance, fault tolerance, and energy consumption for *covariance* and *gesummv* applications under different cases of criticality markings.

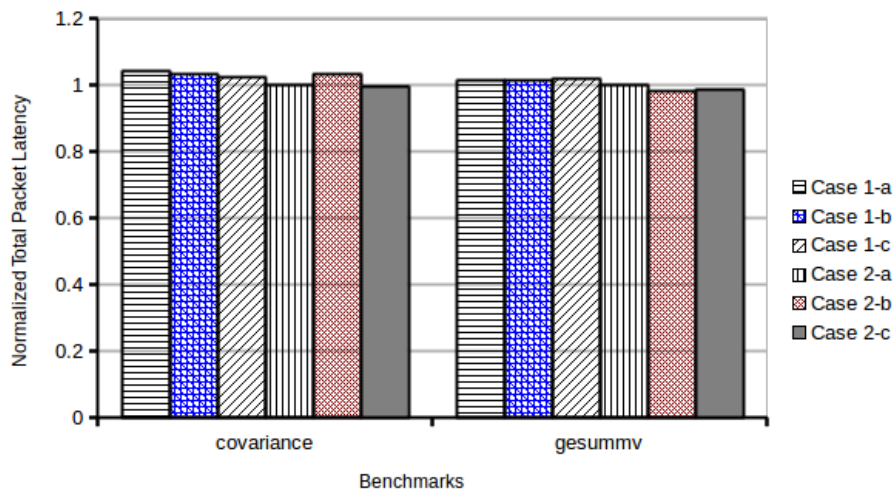


Figure 4.13. Total Packet Latency of Compared Cases

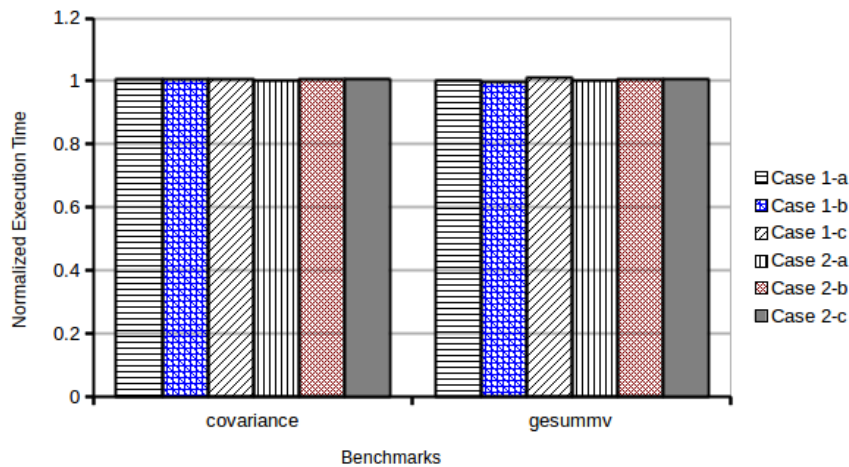


Figure 4.14. Application Execution Time of Compared Cases

Figure 4.13 and 4.14 present the performance results of the cases for each application. The *normalized total packet latency*, which is used as the metric in Figure 4.13, is calculated by dividing the total packet latency of each case with the total packet latency of case 2-a. Likewise, the *normalized execution time*, which is used as the metric in Figure 4.14, is calculated by dividing the application execution time of each case with the application execution time of case 2-a. Case 2-a is used in the calculation of the normalized metric because the criticality marking in case 2-a has a contiguous shape and has the smallest size of manually marked array elements for each application. For *covariance*, cases with either larger sizes of final critical data (array elements and scalar

variables) or more spread locations of marked critical data have slightly worse performance in terms of total packet latency than cases with smaller sizes of final critical data and more contiguous locations. Cases 1-a, 1-b, 1-c, and 2-b in the experiment for the *covariance* application incur slightly higher total packet latency than cases 2-a and 2-c which are displayed to have less number of critical data and have contiguous manually marked data. Similar relationship between size of critical data and total packet latency can also be observed for *gesummv*. Cases 1-a, 1-b, and 1-c of the *gesummv* application, which have higher sizes of critical data, incur slightly higher total packet latency than cases 2-a, 2-b, and 2-c. However, in terms of application execution time, there is no apparent difference among the compared cases. It can be due to the small portion that communication latency at NoC level contributes to the entire application execution time.

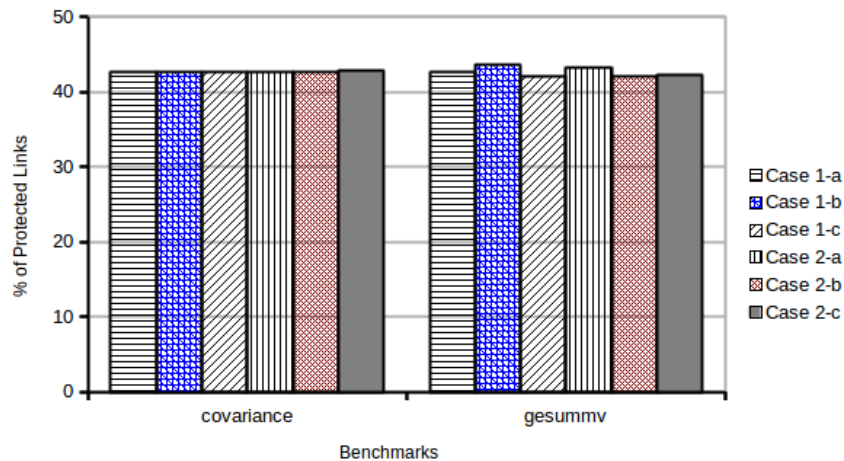


Figure 4.15. Percentage of Protected Links on Critical Communications of Compared Cases

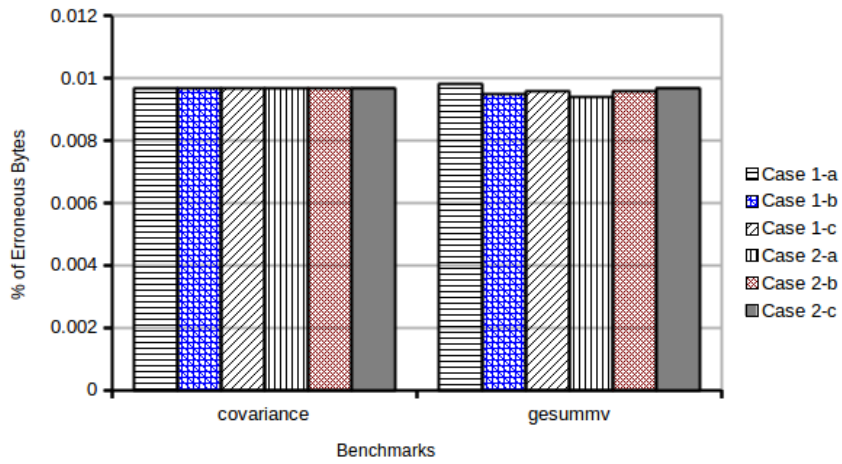


Figure 4.16. Proportion of Erroneous Bytes on Critical Communications

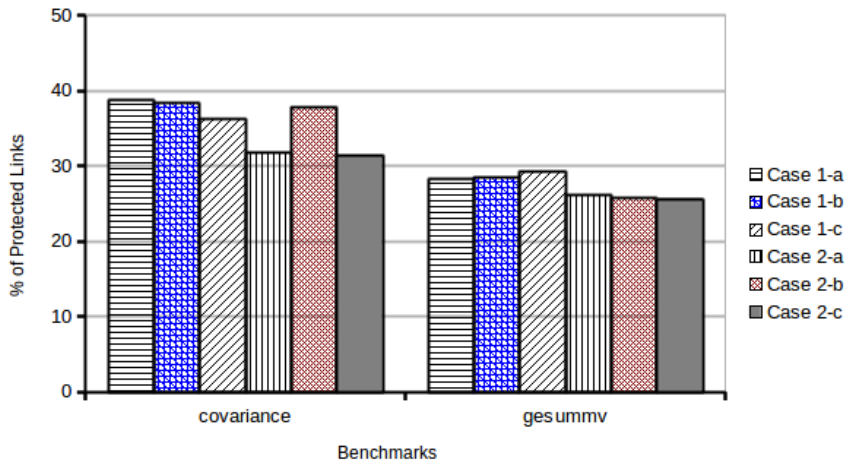


Figure 4.17. Percentage of Protected Links on All Communications of Compared Cases

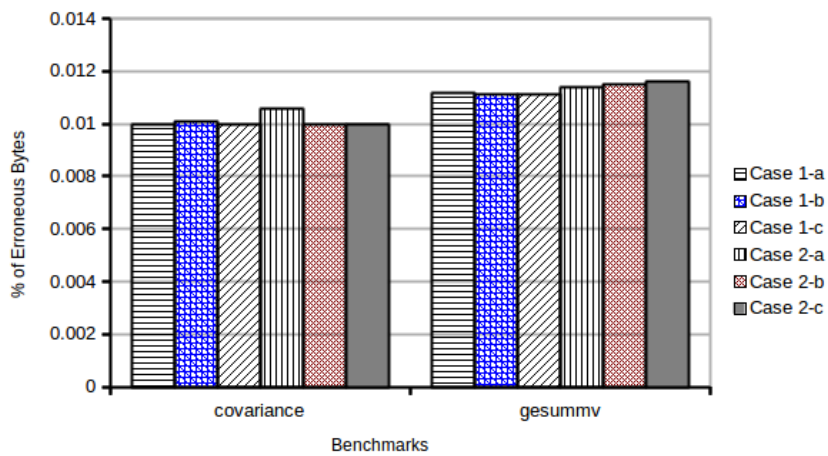


Figure 4.18. Proportion of Erroneous Bytes on All Communications

Figures 4.15, 4.16, 4.17, and Figure 4.18 show the fault tolerance results of the applications under different cases of criticality annotations. As displayed in Figure 4.15 and Figure 4.16, there is no significant difference in the proportion of protected links in critical communications among the cases within each application and across the two applications. This is expected, since packets containing critical data are always transmitted through fault-tolerant routes. Therefore, the proportions of protected links in critical communications for the compared cases and applications are expected to be close to certain value, which in this case is 0.43. Consequentially, since the error injection rate is similar for all cases, the proportion of erroneous bytes on critical communications is also expected to be close to certain value. In this case, the value is 0.01%. However, striking difference in fault tolerance among the cases and applications becomes apparent when the proportions of protected links in all communications are considered. This is because there are differences in the sizes of final critical data and the forms of criticality markings among the cases and applications. The sizes and locations of critical data should affect the number of critical communications involved in application execution. For the proposed scheme, the higher the number of critical communications, the higher the proportion of protected links on all communications becomes. As can be seen in the cases with *gesummv* application, cases 1-a, 1-b, and 1-c, which have bigger sizes of critical data, also have higher proportions of protected links on all communications than cases 2-a, 2-b, and 2-c as displayed in Figure 4.17. This result is consistent with the fault tolerance results measured using proportion of erroneous bytes in Figure 4.18 where cases 1-a, 1-b, and 1-c have lower proportion of erroneous bytes on all communications than cases 2-a, 2-b, and 2-c. On the other hand, for *covariance* application, slightly different comparison among the cases can be observed. Case 2-b has a nearly equal *proportion of protected links on all communications* as cases 1-a, 1-b, and 1-c, while cases 2-a and 2-c still has lower values of this metric. However, this difference can be explained by the fact that the size of critical data resulting from program slicing in the case 2-b is higher than the number in the cases 2-a and 2-c as shown in Table 4.2, and slightly lower than the numbers in cases 1-a and 1-b. This comparison is also nearly consistent with the comparison using proportion of erroneous bytes shown in Figure 4.18. The only exception is the case 2-c of *covariance* which is displayed to have a nearly equal value of proportion of erroneous bytes with case 2-b.

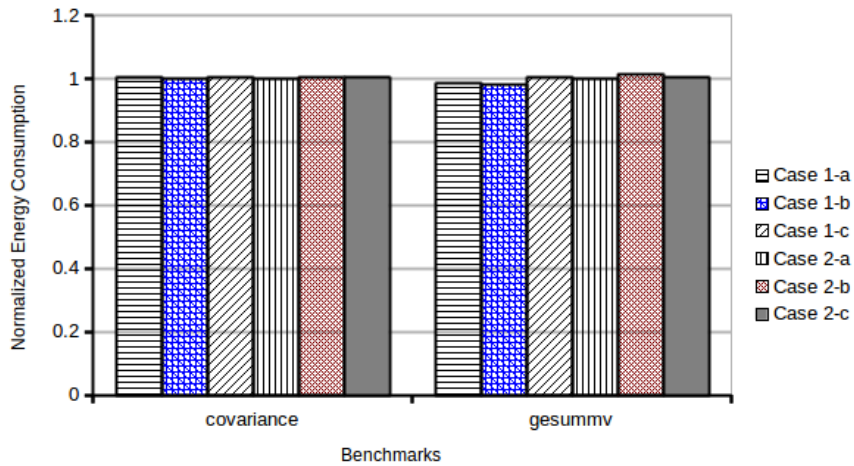


Figure 4.19. Energy Consumption of Compared Cases

Energy consumption results of this experiment are displayed in Figure 4.19. As can be seen in the figure, there is no significant difference among the cases. Each case consumes approximately 18.35 Joules for *covariance* and 0.23 Joules for *gesummv* during the simulated application executions.

4.2.3 Evaluating Impact of Route Length Upper Bound

In this phase of experimental study, the default *critical-transmission-only scheme* is modified to improve its reliability by enabling it to generate route between any two nodes with length that is longer than the length of shortest paths between the two given nodes. A parameter x called *number of additional edges* is introduced, which defines the maximum number of additional edges that can be added to the length of shortest paths between any two given nodes. It is expected that the higher the value of x is, the higher the number of protected links which can be included in the produced routes and therefore the more fault tolerant the critical communications becomes.

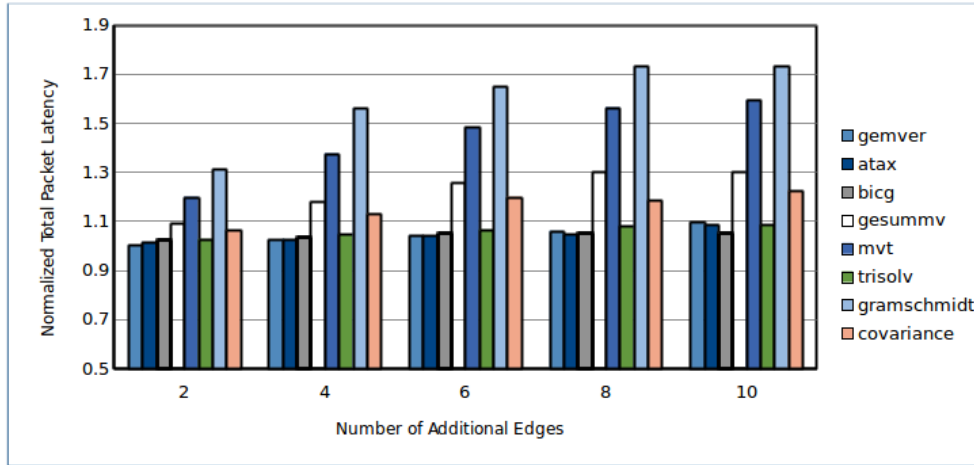


Figure 4.20. Total Packet Latency over Different Length Constraints

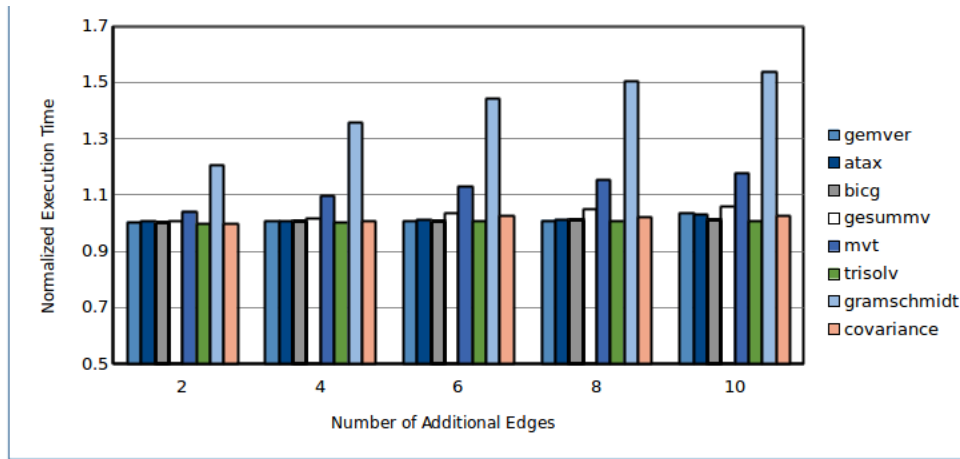


Figure 4.21. Application Execution Time over Different Length Constraints

Figure 4.20 and Figure 4.21 present the performance results of all applications under *critical-transmission-only* scheme across different numbers of additional edges. The *normalized total packet latency* in Figure 4.20 is calculated by dividing the total packet latency of each case in the experiment by the total packet latency of the case when the *number of additional edges* is 0. The *normalized execution time* in Figure 4.21 is calculated by dividing the application execution time of each case in the experiment by the application execution time of the case when the *number of additional edges* is 0. The smallest increments of packet latency are shown by applications *bicg* and *trisolv*. For *bicg*, the *normalized total packet latency* becomes 1.026, 1.034, 1.050, 1.054, and 1.055 when the value of x is 2, 4, 6, 8, and 10 respectively. For *trisolv*, the *normalized total packet latency* becomes 1.026, 1.046, 1.065, 1.078, and 1.085 when the value of x is 2, 4,

6, 8, and 10 respectively. When application execution time is considered, nearly all applications don't display significant increments when length constraint gets higher. The only exceptions are *mvt* and *gramschmidt*. For *mvt*, the *normalized execution time* becomes 1.041, 1.095, 1.129, 1.153, and 1.176 when the value of x is 2, 4, 6, 8, and 10. For *gramschmidt*, the *normalized execution time* becomes 1.205, 1.356, 1.441, 1.503, and 1.539 when the value of x is 2, 4, 6, 8, and 10. As can be observed, the increments in application execution time are much less than the increments in total packet latency. This is because communication latency at NoC level only constitutes a small portion of the entire application execution time.

For all applications displayed in the figures, performance degradation is apparent as the upper bound of route length gets higher. This decrease in performance is caused by the increase in length of generated routes for critical communications. This, in turn, makes the total packet latency of critical communications higher and reduces the performance of the application.

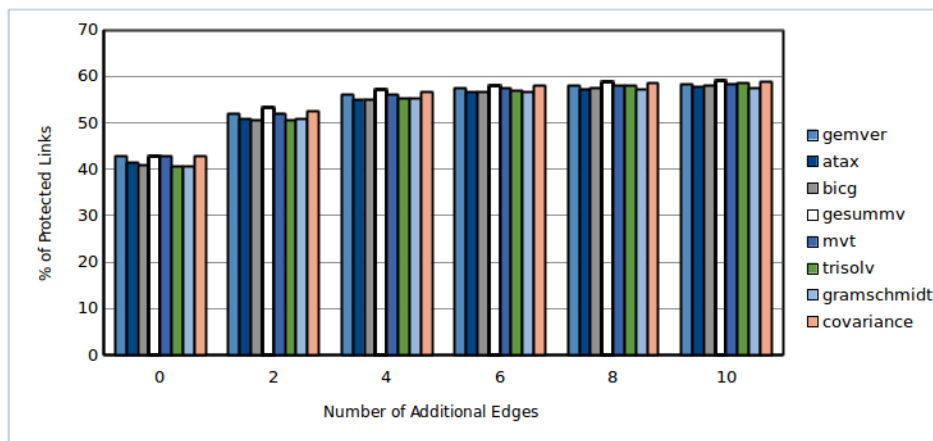


Figure 4.22. Percentage of Protected Links on Critical Communications over Different Length Constraints

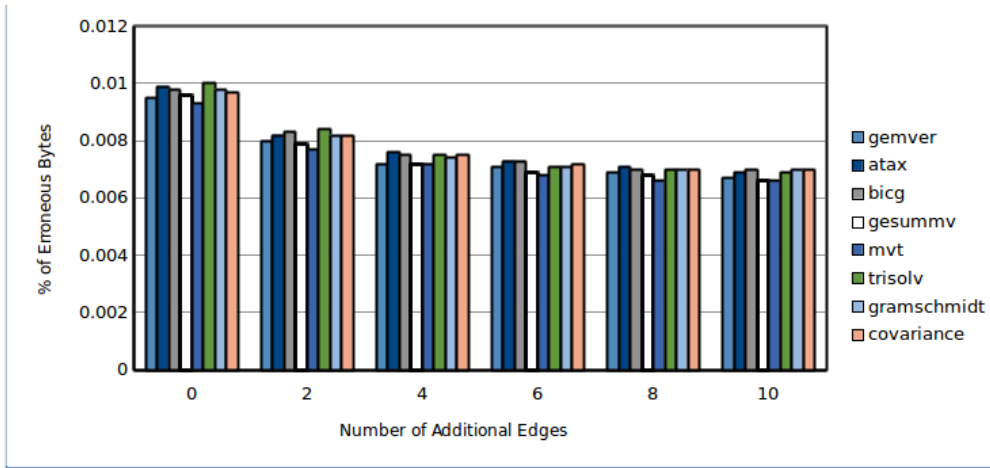


Figure 4.23. Proportion of Erroneous Bytes on Critical Communications over Different Length Constraints

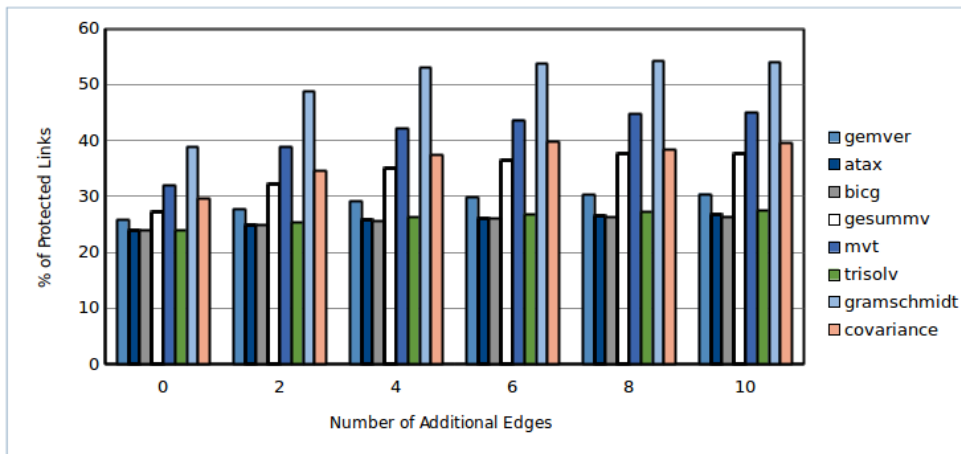


Figure 4.24. Percentage of Protected Links on All Communications over Different Length Constraints

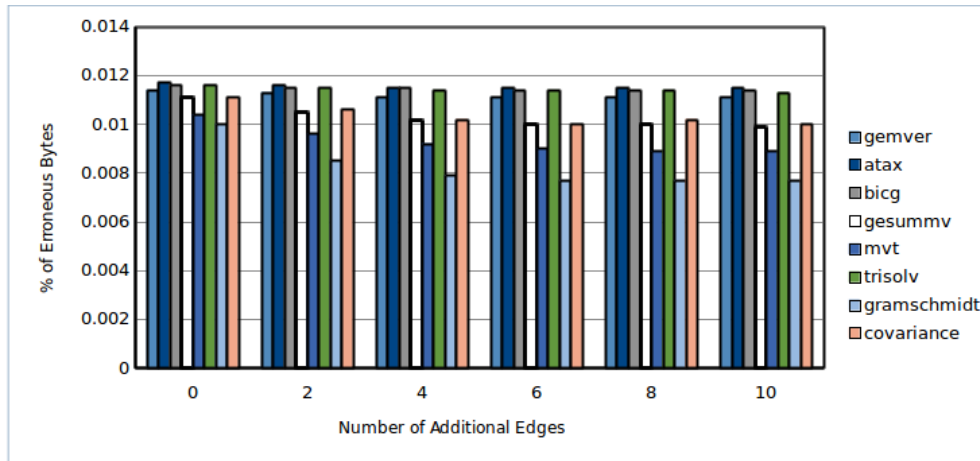


Figure 4.25. Proportion of Erroneous Bytes on All Communications over Different Length Constraints

Figures 4.22, 4.23, 4.24 and 4.25 display the fault tolerance results of this experiment. In all cases of different applications, there is an improvement of fault tolerance in terms of proportion of protected links either in critical transmissions only or in all communications as the upper bound of route length gets higher. A similar pattern can be observed in Figure 4.23 and Figure 4.25 where proportion of erroneous bytes is used as the metric to measure fault tolerance in critical transmissions and in all communications. The smallest increment of *proportion of protected links on all communications* is shown by *bicg*, while the smallest decrements of proportion of erroneous bytes on all communications are displayed by *gemver*, *atax*, and *bicg*. For application *bicg*, the *percentage of protected links on all communications* is 24%, 25%, 25.6%, 26%, 26.2% and 26.4% when x is 0, 2, 4, 6, 8, and 10 respectively. This increase in fault tolerance is due to the higher number of protected links included in the fault-tolerant routes for critical communications as the upper bound of route length gets higher. Therefore, in selecting the number of additional edges to increase the upper bound of route length, we might need to consider the trade-off between fault tolerance improvement and performance degradation. Setting upper bound to high values might reduce performance while increasing fault tolerance.

Another pattern that can be observed from the results is that as the number of additional edges increases, the degree of increment in the proportion of protected links and the degree of decrement in the proportion of erroneous bytes gets smaller. As displayed by nearly all applications, the degrees of increment and decrement are the

highest as the number of additional edges get higher from 0 to 2, and then, the values of increments and decrements become smaller for the next following increases of number of additional edges. This pattern shows that there might not be significant improvement that can be acquired in terms of fault tolerance beyond certain value of upper bound.

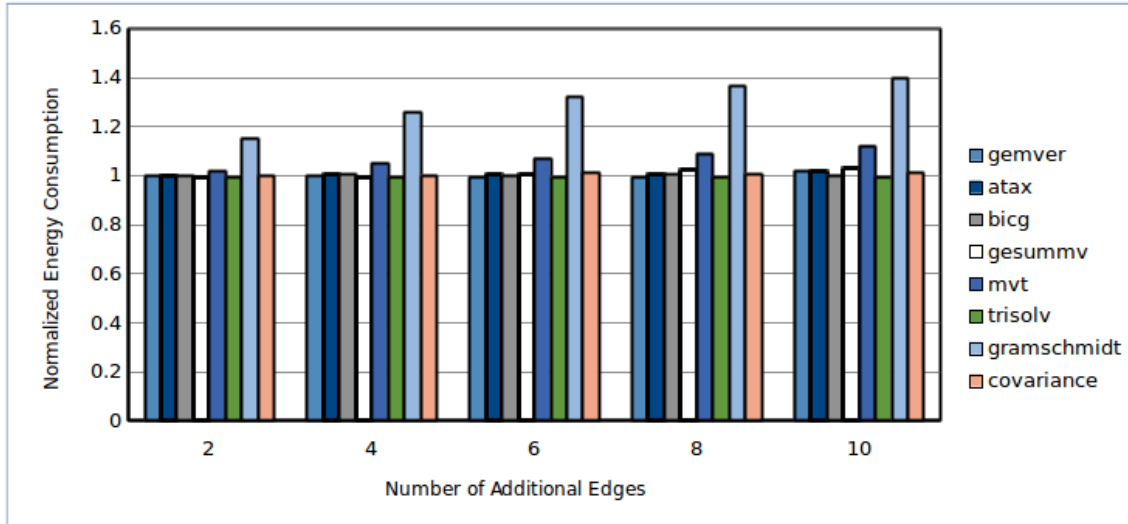


Figure 4.26. Energy Consumption over Different Length Constraints

Energy consumption results are presented in Figure 4.26. The presented results indicate that there is no apparent increase in energy consumption as length constraint increases, except for *mvt* and *gramschmidt*. For *mvt*, the *normalized energy consumption* becomes 1.017, 1.051, 1.070, 1.090, and 1.121 when the value of x is 2, 4, 6, 8, and 10. For *gramschmidt*, the *normalized energy consumption* becomes 1.154, 1.258, 1.320, 1.365, and 1.4 when the value of x is 2, 4, 6, 8, and 10. These results are consistent with the results presented in Figure 4.21 since energy consumption is directly proportional with application execution time.

5. CONCLUSIONS AND FUTURE WORK

In this thesis, a new fault tolerance scheme is introduced for NoC which increases the reliability of packet transmissions that contain critical data. This proposed scheme comprises a compiler layer and a hardware layer. The compiler layer enables the programmer to mark parts of computation results which are considered as critical and propagates the criticality to other array elements or variables which directly/indirectly assign values to the manually marked array elements and variables. The hardware layer uses the information about the list of critical data that is generated by the compiler layer to identify packet transmissions in NoC which carry any of the critical data. These packet transmissions are called as *critical communications*. The runtime system of the NoC produces routes which improve the fault tolerance of critical communications by including as many *protected links* as possible without surpassing the length constraint of the routes. The default length constraint is the length of shortest paths between sender and destination nodes. These routes are referred to as *fault-tolerant routes*.

The results of the experimental study in this work demonstrate that the proposed scheme has higher fault tolerance, in terms of the *proportion of protected links in critical communications*, *proportion of erroneous bytes on critical communications*, *proportion of protected links on all communications*, and *proportion of erroneous bytes on all communications*, than XY-routing scheme, even though the performance of XY-routing scheme is still better and it still consumes less amount of energy. The results also validate that the fault tolerance of critical communications in our scheme is approximately equal to the fault tolerance of critical communications in the *full-fault-tolerant* routing scheme. However, the performance and energy consumption of our scheme are still better and less costly than the performance and energy consumption of *full-fault-tolerant* and *XY-routing with HBH-SEC* schemes.

Another result of experimental study shows that as the size of critical data resulting from program slicing becomes higher, the communications during the overall application execution will become more fault tolerant. This is because the proportion of critical communications, which is subjected to *fault-tolerant-routing* within the entire communications in the application execution gets higher. On the other hand, the latency of the communications will also become longer as the proportion of critical

communications within the entire communications also increases.

Another conclusion that can be taken from the experimental study is that as the number of additional edges that can be included to fault-tolerant routes gets higher, critical communications will also become more fault-tolerant, but the latency of the entire communications will increase. However, the increase of communication latency due to the increase of length constraints might have little impact on the performance and energy consumption of the entire application. Furthermore, there is also an apparent slight reduction in the degree of increment of fault tolerance as the number of additional edges increases. This pattern implies that there might not be significant improvement in fault tolerance if the number of additional edges reaches beyond certain value.

As a future work, we plan to evaluate the trade-off between the number of *protected links* and the performance, fault tolerance, and energy consumption of the compiler-enhanced reliability framework.

6. REFERENCES

- [1] M. Ali, M. Welzl, and M. Zwicknagl. Networks on chips: scalable interconnects for future systems on chips. In *4th European Conference on Circuits and Systems for Communications, 2008. ECCSC 2008*, pages 240-245, 2008.
- [2] L. Benini, and G. De Michelli. Networks on chips: a new SoC paradigm. *IEEE Computer*, 35 (1): 70–78, 2002.
- [3] S. Kumar, et al. A network on chip architecture and design methodology. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, pages 117–124, April 2002.
- [4] Semiconductor Association. "The International Technology Roadmap for Semiconductors (ITRS)", 2001.
- [5] E. Dupont, M. Nicolaidis, and P. Rohr. Embedded robustness IPs for transient-error-free ICs. *IEEE Des Test Comput*, 19 (3):54–68, May–June 2002.
- [6] A. Piotrowski, D. Makowski, G. Jabłonski, S. Tarnowski, and A. Napieralski. Hardware fault tolerance implemented in software at the compiler level with special emphasis on array-variable protection. In *MIXDES 2008 - Mixed Design of Integrated Circuits and Systems*, pages 115-119, June 2008.
- [7] M. Rebaudengo and M. Sonza Reorda. Evaluating cost and effectiveness of software redundancy techniques for hardware errors detection. In *The 28th Annual International Symposium on Fault-Tolerant Computing (FTCS-28)*, pages 88–89, June 1998.
- [8] M. Rebaudengo, M. Sonza Reorda, and M. Violante. A new approach to software-implemented fault tolerance. *Journal of Electronic Testing: Theory and Applications*, 20:433–437, 2004.
- [9] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante. Soft-error detection using control flow assertions. In *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, pages 581-588, 2003.
- [10] Semiconductor Association. "The International Technology Roadmap for

- Semiconductors (ITRS)", 2001.
- [11] S. Arslan, H.R. Topcuoglu, M.T. Kandemir, and O. Tosun. Performance and energy efficient asymmetrically reliable caches for multicore architectures. In: *Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, pages 1025–1032, 2015.
 - [12] S. Arslan, H.R. Topcuoglu, M.T. Kandemir, O. Tosun. Protecting code regions on asymmetrically reliable caches. In: *Proceedings of the 2016 29th Architecture of Computing Systems*, pages 375-387, 2016.
 - [13] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu. Characterizing application memory error vulnerability to optimize data center cost via heterogeneous-reliability memory. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 467–478, 2014.
 - [14] S. Rehman, F. Kriebel, M. Shafique, and J. Henkel. Compiler-driven dynamic reliability management for on-chip systems under variabilities. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–4, 2014.
 - [15] D. Borodin et al. Protected redundancy overhead reduction using instruction vulnerability factor, *IEEE CF*, pages 319-326, 2010.
 - [16] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. Demicheli. Analysis of error recovery schemes for networks on chips. *IEEE Des. Test Comput.* 22 (5): 434–442, 2005.
 - [17] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das. Exploring fault-tolerant network-on-chip architectures. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. *IEEE Computer Society*, pages 93–104. 2006.
 - [18] M. Ali, M. Welzl, S. Hessler, and S. Hellebrand. An efficient fault tolerant mechanism to deal with permanent and transient failures in a network on chip. *Int. J. High Performance Syst. Architecture*, 1 (2): 113–123, 2007.

- [19] A. Ganguly, P. P. Pande, B. Belzer, and C. Grecu, Design of low power & reliable networks on chip through joint crosstalk avoidance and multiple error correction coding. *J. Electron. Testing: Theory Appl. (JETTA), Special Issue on Defect Fault Tolerance*, pages 67–81, Jun. 2008.
- [20] A. Frantz, F. Kastensmidt, L. Carro, and E. Cota. Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk. In *Proceedings of the IEEE International Test Conference (ITC'06)*, pages 1–9, 2006.
- [21] PLUTO: A polyhedral automatic parallelizer and locality optimizer for multicores. <http://pluto-compiler.sourceforge.net>.
- [22] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In *Proc. International Conference on Compiler Construction (ETAPS CC)*, 2008.
- [23] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proc. ACM SIGPLAN Programming Languages Design and Implementation (PLDI)*, 2008.
- [24] C. Bastoul. Clan-a polyhedral representation extractor for high level programs, May 2008.
- [25] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA '98)*, 1998.
- [26] H. Zimmer and A. Jantsch. A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 188-193, 2003.
- [27] A. Dutta and N. A. Touba. Reliable network-on-chip using a low cost unequal error protection code. In *Proc. DFT'07*, pages 3–11, 2007.
- [28] A. B. Ahmed, M. Meyer, Y. Okuyama, and A. B. Abdallah. Adaptive Error- and Traffic-Aware Router Architecture for 3D Network-on-Chip Systems. In *Proceedings of the IEEE 8th International Symposium on Embedded*

Multicore/Manycore SoCs (MCSoc), 2014.

- [29] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Comput. Surv.* 44: 1–35, 2012.
- [30] M. Nicolaidis. Design for soft error mitigation. *IEEE Trans. on Device and Materials Reliability*, 5 (3): 405-418, Sept. 2005.
- [31] A. P. Frantz, F. L. Kastensmidt, L. Carro, and E. Cota, Evaluation of seu and crosstalk effects in network-on-chip switches. In *Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI'06)*, pages 202-207, 2006.
- [32] M. CuvIELLO, S. Dey, X. Bai, and Y. Zhao. Fault modeling and simulation for crosstalk in system-on-chip interconnects. In *IEEE/ACM International Digest of Technical Papers on Computer-Aided Design*, pages 297–303, 1999.
- [33] B. Fu, and P. Ampadu. On hamming product codes with type-ii hybrid arq for on-chip interconnects. *IEEE Trans. Circ. Syst. I: Regular Papers*, 56 (9): 2042–2054, 2009.
- [34] R. R. Tamhankar, S. Murali and G. De Micheli. Performance driven reliable link design for networks on chips. In *Proceedings Asia and South Pacific Design Automation Conference*, 2: 749–754, 2005.
- [35] David Bertozzi and Luca Benini, Xpipes: A network-on-chip architecture for gigascale system-on-chip. Magazine, *IEEE Circuits and Systems*, Second Quarter, pages 18-31, 2004.
- [36] T. Lehtonen, P. Liljeberg, and J. Plosila. Analysis of forward error correction methods for nanoscale networks-on-chip. In *Proc. 2nd Int. Conf. Nano-Networks (Nano-Net 2007)*, pages 1–5, 2007.
- [37] P. Bogdan, T. Dumitras, and R. Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. *VLSI Design*, 2007: 1–17, 2007.
- [38] T. Dumitras and R. Marculescu. On-chip stochastic communication. In *Proc. the Design, Automation and Test in Europe Conference and Exhibition*, 2003.
- [39] M. Piretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. in *Proc.*

International Symposium on VLSI, 2004.

- [40] J. Flich, A. Mejia, P. Lopez, and J. Duato. Region-based routing: an efficient routing mechanism to tackle unreliable hardware in network on chips. in *Proc. Symposium on Networks-on-Chip (NoCS'07)*, 2007.
- [41] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. Lopez, R. Holsmark, and J. Duato. Region-based routing: a mechanism to support efficient routing algorithms in nocs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3): 356–369, 2009.
- [42] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. R. Das. Design and analysis of an NoC architecture from performance, reliability and energy perspective. In *Proc. of the Symposium on Architecture for networking and communications systems (ANCS)*, 2005.
- [43] A. Jantsch, R. Lauter, and A. Vitkowski. Power analysis of link level and end-to-end data protection in networks on chip. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'05)*, 2: 1770–1773, 2005.
- [44] S. Murali, D. Atienza, L. Benini, and G. D. Micheli. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proc. 43rd ACM/IEEE Design Automation Conference (DAC06)*, 2006.
- [45] Y. B. Kim and Y. B. Kim, Fault tolerant source routing for network-on-chip. In *Proc. 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'07)*, 2007.
- [46] P. M. Wells, K. Chakraborty, and G. S. Sohi. Mixed-mode multicore reliability. In *17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 169–180, 2009.
- [47] N. J. Alewine, S. K. Chen, C. C. Li, W. K. Fuchs, and W.M. Hwu. Branch recovery with compiler-assisted multiple instruction retry. In *The 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 66-73, 1992.
- [48] V. Balasubramanian and P. Banerjee. Compiler-assisted synthesis of algorithm-based checking in multiprocessors. *IEEE Zbm. on Computers*, Vol. 39, pages 436-

446, 1990.

- [49] A. Piotrowski, D. Makowski, G. Jablonski, and A. Napieralski, The automatic implementation of software implemented hardware fault tolerance algorithms as a radiation-induced soft errors mitigation technique. In *Nuclear Science Symposium NSS/MIC/RTSD*, pages 841-846, 2008.
- [50] A. Piotrowski, D. Makowski, S. Tarnowski, and A. Napieralski. Automatic implementation of radiation protection algorithms in programs generated by GCC compiler. In *European Particle Accelerator Conference*, 2008.
- [51] A. Piotrowski. Automatic installation of software-based fault tolerance algorithms in programs generated by GCC compiler. In *MIXDES 2010 – Mixed Design of Integrated Circuits and Systems*, pages 101-105, 2010.
- [52] G. Nazarian, R. M. Seepers, C. Strydis, and G. N. Gaydadjiev. Compiler-aided methodology for low overhead on-line testing. In *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, pages 219-226, 2013.
- [53] C. Gong, R. Melhem, and R. Gupta. Compiler assisted fault detection for distributed-memory systems. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 373-380 , IEEE, 1994.
- [54] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4): 352-357, 1984.
- [55] C. Bastoul, A. Cohen, S. Girbal, S. Sharma, and O. Temam. Putting polyhedral loop transformations to work, In *Workshop on Languages and Compilers for Parallel Computing (LCPC'03)*, LNCS, pp. 23–30. Springer-Verlag, College Station, Texas, 2003.
- [56] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *IEEE PACT*, pages 7–16, Sept. 2004.
- [57] C. Bastoul. Openscop: a specification and a library for data exchange in polyhedral compilation tools. Technical Report, Paris-Sud University, France, September 2011.

- [58] S. C. Johnson. Yacc: Yet Another Compiler-Compiler, Bell Laboratories; Murray Hill, NJ, 1978.
- [59] M. E. Lesk. Lex – a lexical analyzer generator, Bell Laboratories; Murray Hill, NJ, 1975.
- [60] PolyLib - A library of polyhedral functions. <https://icps.u-strasbg.fr/polylib>.
- [61] T. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1-12, 2011.
- [62] T. E. Carlson and W. Heirman. The sniper user manual. November 2013.
- [63] P. Kermani and L. Heinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3: 267-286, 1979.
- [64] Polybench/c. The polyhedral benchmark suite. [Online]. Available: <http://web.cse.ohio-state.edu/~pouchet/software/polybench/>
- [65] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, N. P. Jouppi. McPAT: an integrated power area and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd annual IEEE/ACM Intl. Symp. on Microarchitecture*, 2009.

RESUME

Muhammad Aditya Sasongko

Marmara University,

Computer Engineering Department, Faculty of Engineering,

Goztepe Campus, Kadikoy, Istanbul, Turkey

Phone (Cell)+90-5543286203

e-mail: aditya.sasongko4@gmail.com

EDUCATION

M.S., Computer Engineering Marmara University, Faculty of Engineering, Istanbul, Turkey.

Thesis Topic: Compiler Support for Enhancing Reliability of Network-on-Chip Architectures, *Advisor:* Prof. Haluk TOPCUOGLU.

B.S., Middle East Technical University, Ankara, Turkey, 2012

WORK EXPERIENCE

2013-January 2017, *Research Assistant*, Multicore Computing Research Laboratory, Marmara University, Computer Engineering Dept, Istanbul, Turkey

PUBLICATIONS

M.A. Sasongko, H. Topcuoglu, S. Arslan, M. Kandemir, "Compiler Enhanced Reliability for Network-on-Chip Architectures", Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2017), St Petersburg, Russia, March 2017

RESEARCH INTERESTS

Multicore Computing, Fault Tolerance, Compiler Design, Networks on Chip

LANGUAGES

English (fluent), Turkish (Intermediate), Indonesian (Native Language)