Smart Reconfiguration Approach for Fault-Tolerant NoC Based MPSoCs

Jarbas Silveira, Paulo Cortez,
Alan Cadore, Rafael Mota
LESC-DETI - Federal University of Ceará (UFC)
Fortaleza, Brazil
jarbas@lesc.ufc.br

César Marcon, Lucas Brahm,
Ramon Fernandes
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brazil
cesar.marcon@pucrs.br

ABSTRACT
Newest technologies of integrated circuits fabrication allow billions of transistors arranged in a single chip enabling to implement a complex parallel system, which requires a high scalable and parallel communication architecture, such as a Network-on-Chip (NoC). These technologies are very close to physical limitations increasing faults in manufacture and at runtime. Thus, it is essential to provide a fault recovery mechanism for NoC operation in the presence of faults. The preprocessing of the most probable fault scenarios and flits retransmission capability enable to anticipate the calculation of deadlock-free routings, reducing the time necessary to interrupt the system in a fault occurrence and maintaining links operating with retransmission capability. This work proposes a smart decisions mechanism for errors on NoC links, which is composed of a hardware part implemented into the links and routers, and a software part implemented inside an operating system kernel of each processor. The mechanism defines thresholds where is better to reconfigure the NoC or to retransmit flits with errors. Experimental results, with several NoC sizes and some error models, suggest when is better to reconfigure the NoC and when is better to maintain some links operating with eventual faults.

Categories and Subject Descriptors
B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms
Design, Reliability, Verification.

Keywords
Fault-tolerance; NoC; MPSoC; routing methods; reconfiguration.

1. INTRODUCTION
The evolution of VLSI semiconductor technology enables to integrate hundreds of cores into a single circuit. This massive integration allows implementing the entire functionality of a system into a single chip producing a System-on-Chip (SoC). The International Technology Roadmap for Semiconductors (ITRS) foresees hundreds of Processing Elements (PEs) integrated into an SoC by 2020 [1]. A Network-on-Chip (NoC) [2] plays a key role in the communication of these highly integrated SoCs, with the two-dimensional (2D) mesh as the most popular NoC topology, offering a simple and regular structure for tile-based design [3].

Recent submicron technologies provide more process variability increasing the number of defective components [4], which may collapse a mesh communication's structure leading to an irregular topology [5]. Thus, static and deterministic routing algorithms tailored to a regular NoC topology will not operate properly, thus rendering the chip useless [3]. Likewise the works [6][7], we employ an approach based on turn prohibition to eliminate deadlocks in irregular NoC topologies. Additionally, each router contains a table to implement the routing algorithm, using a technique similar to [5], compressing the routing table according to NoC regions for saving area and power, increasing scalability.

The design of a fault detection/correction mechanism has to consider three types of faults: (i) the one that is detected and corrected locally, and whose effect is not propagated to a higher hardware/software layer (e.g., a fault corrected by a CRC circuit); (ii) the one that is detected, but not recovered in the detected level, requiring a higher fault correction mechanism; and (iii) the fault is not detected at a low level, requiring a higher level of detection and correction. According to the fault dynamicity, the fault-tolerant mechanism may employ a strategy that tolerates some occurrence of faults during communication, or employ a strategy that exploits new communication scenarios to find fault-free paths.

We propose an efficient approach for dealing with dynamic faults on NoC links based on Phoenix [8], which is a fault-tolerant architecture comprising a 2D mesh NoC and a software layer that controls the fault-tolerance mechanism. A fault-tolerant circuit, placed in each inter-router link, takes smart decisions about flit retransmission or NoC reconfiguration based on fault scenarios preprocessing. Moreover, there are two interrelated procedures when a fault is detected: (i) to maintain the faulty link in use; or (ii) to reconfigure the entire communication, avoiding faulty links.

This paper is organized as follows. Section 2 presents related work on routing mechanisms and NoC reconfiguration, and the main contribution here. Section 3 details the hardware and the software of Phoenix's architecture. Section 4 describes the processing flow of fault-tolerant scenarios. Section 5 presents the basics for flit retransmission with smart decisions. Section 6 describes the methodology and experimental results, while Section 7 analyzes methods and costs of the reconfiguration or retransmission approach. Finally, Section 8 presents our main conclusions.

2. RELATED WORK
A reconfigurable fault-tolerant NoC requires mechanisms of (i) fault detection; (ii) fault recovery; (iii) routing computation; and (iv) routing reconfiguration to keep the correct system operation in the presence of faults. These mechanisms, considering only network architectures without virtual channels, are discussed next.

SBCCI '15, August 31 - September 04, 2015, Salvador, Brazil
 ISBN 978-1-4503-3763-2/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2800986.2801027

Feng et al. [9] describe a fault-tolerant NoC, including an on-line fault diagnosis mechanism, a link-level error control scheme, and a fault-tolerant routing algorithm on bufferless routers for both transient and permanent faults. The fault diagnosis mechanism uses single-error-correcting and double-error-detecting to detect transient and permanent link faults. Meanwhile, Ying et al. [10] propose a fault-tolerant mechanism for transient and permanent faults based on NoC monitors and an error detection scheme at flit-level to handle transient faults on the data links, while a dynamic routing mechanism deals with a permanent faulty link. Yu et al. [11] propose an error control method for co-manage transient and permanent errors in the data link and physical layers.

According to Radetzki et al. [12], achieving time redundancy means repeating the computation, sampling, or retransmission. At the data link layer, three types of time redundancy are widely used, namely multisampling and correction, hop-to-hop retransmission, and split link transmission. Our work implements retransmission, also known as Automatic Repeat Request (ARQ), which has been proposed and used for decades to provide error control in communication networks. The ARQ technique at the data link layer in NoCs is implemented as the hop-to-hop retransmission often coupled with an error detection and correction technique.

The routing mechanism needs resources that may be changed to support routing reconfiguration at runtime; usually through routing tables, which support many topologies and are easy to implement. Several works employ techniques to reduce or minimize the size of the routing tables, aiming to reach the scalability required for current and future high-populated NoCs. It is a complex task and may imply the loss of performance and/or the impossibility of reaching all target nodes. Examples of these works are (i) Palesi et al. [13], which uses a table compression technique for application-specific routing, and (ii) Bolotin et al. [14] that uses table minimization technique applying a fixed function combined with minimal deviation tables.

Dividing the network in regions is another approach for reducing routing table sizes. For instance, Mejia et al. [5] proposed the Region Based Routing (RBR) approach, where each node contains set of regions based on paths that cover all communications. Fukushima, Fukushi and Yairi [15] propose another region based approach based on a set of rectangular faulty regions and corresponding deviation paths. Their approach improves the work of Holsmark et al. [16] providing complete and deadlock-free routing, reducing regions size and implementation complexity.

The reconfiguration process defines the computation cost of taking dynamic routing decisions. Fick et al. [17] describe the architecture of Vicis, which is a fault-tolerant NoC that preserves the functionality of the system based on the inherent redundancy found in most networks. Each router now has a Built-In Self-Test circuit to diagnose faults and to reconfigure the hardware. Triviño et al. [18] use virtual-regions to improve application performance that are simultaneously running in a Chip MultiProcessor (CMP), which results in the partitioning of the CMP in several regions through a dynamic reconfiguration algorithm.

Our work employs a dynamic fault model encompassing three phases: (i) fault detection and fault report; (ii) deadlock-free routing computation; and (iii) routing reconfiguration. The main contribution of this work is in the routing reconfiguration phase that provides fast deadlock-free routing reconfigurations for irregular NoC topologies. It is based on preprocessing the most probable fault scenarios, which are computed according to the detection of link fault tendency. Using our approach and considering a fault tendency detected condition, we can employ more complex and time-consuming algorithms to produce optimal solutions for large NoCs without compromising application runtime since the routing tables are already preprocessed. Moreover, a given set of scenarios may encompass more than one fault situation, reducing the total amount of scenarios. We provide two new and significant contributions: (i) an analytic metric to choose at runtime the substitution scenario that provides the most efficient routing; and (ii) a novel method to reduce a large set of scenarios based on cross-correlation measure that identifies dissimilarities in sets of irregular topologies, minimizing the storage area for preprocessed scenarios.

3. PHOENIX'S ARCHITECTURE

Figure 1 shows the distributed fault-tolerant architecture of Phoenix [8] on a NoC-based MPSoC that includes a hardware part (i.e. *HwPhoenix*) placed on each router of the NoC and a software part (i.e. *OsPhoenix*) placed on the operating system of each PE.

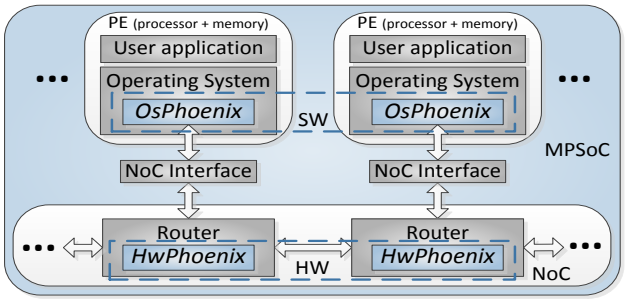

Figure 1 – Phoenix distributed architecture [8].

Each PE connects through a NoC interface the local port of each router. Each field of a Phoenix's packet is 1-flit length, and the number of flits in a packet is limited to $2^{(\text{flit size in bits})}$. Phoenix uses two types of packets: (i) data packet, for the application messages; and (ii) control packet, for the fault-tolerant mechanism. The software and hardware layers communicate via the bidirectional control packets transmitted through the local port of each PE.

3.1 OsPhoenix Architecture

The *OsPhoenix* is a software layer, which contains drivers for high-level operation, and routines that implement the distributed fault-tolerant mechanism. The PE's operating system perceives this software layer as a network driver interface, making the fault-tolerant mechanism transparent to the system operation. Figure 2 depicts the main modules of *OsPhoenix* and their interaction. The Kernel of *OsPhoenix* includes a Control Module (CM) for managing the fault-tolerant mechanism and the NoC Driver that adapt and route control and data packets.

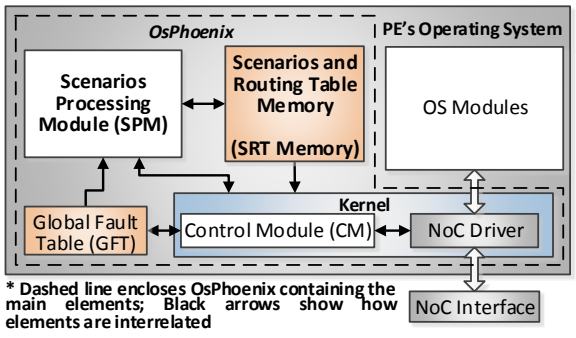

Figure 2 – Block diagram of PE's OS architecture.

The Global Fault Table (GFT) stores the status of all NoC links. It is a global copy of all routers' Fault Table (Section 3.2). The CM

writes/reads this table to synchronize information among all PEs. The Scenarios Processing Module (SPM) calculates routing tables according to the fault or fault tendency on links when commanded by the CM. It uses the information provided by the GFT together with new faults information to search for a previously computed scenario that covers this new fault situation, in the Scenarios and Routing Table Memory (SRT Memory). If a candidate scenario is found, the associated Routing Table in updated at the hardware layer. Otherwise, this module processes a new fault-tolerant scenario and its associated RBR Table. According to our system retransmission features, when a fault detection occurs, *OsPhoenix* decides if reconfiguration is necessary.

3.2 HwPhoenix Fundamentals

Phoenix NoC is a direct 2D mesh topology consisting of *m*×*n* routers using bidirectional links for routers and PEs interconnection. The NoC employs routing tables for distributed routing decisions and the *OsPhoenix* performs routing algorithms to fill the routing table according to the relative position of each PE. Further, Phoenix NoC implements wormhole switching, demanding only small buffers for data storing. Additionally, a credit-based flow control reduces transmission clock latency.

Figure 3 shows the Phoenix router architecture, which includes mechanisms for packet routing and fault tolerance. The packet routing mechanism encompasses: (i) Four bidirectional ports; (ii) a Crossbar Switch that establishes unblocking connections between input and output ports; (iii) a Routing Table that associates regions of the NoC with output ports; and (iv) a Switch Control that performs the packets routing and arbitration.

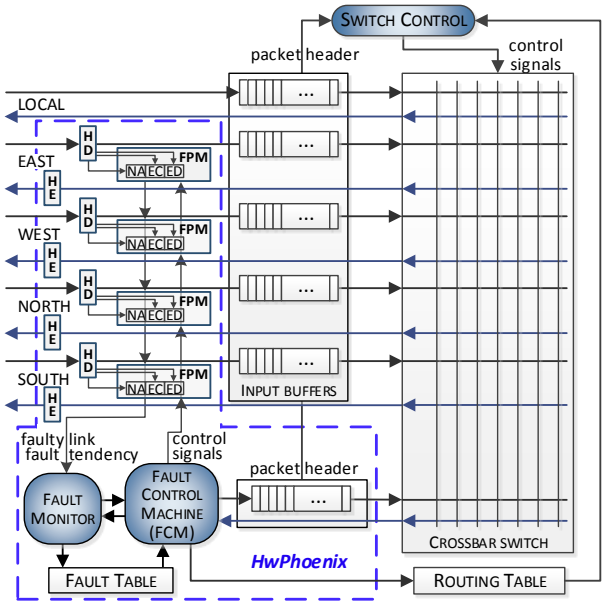

Figure 3 – Basic components of Phoenix router architecture. Dashed lines bound the main components of *HwPhoenix*.

The NoC routing algorithm is similar to RBR [5], which groups target addresses into regions to reduce the Routing Table size. The fault-tolerant circuit implemented in each router includes three types of circuits: (i) fault detection and correction module containing a Hamming Encoder (HE), a Hamming Decoder (HD) and a Fault Prediction Module (FPM - [24]), which are placed in each one of the links that interconnect routers; (ii) Fault Monitor that communicates with the FPM to set the status of the links on the Fault Table according to a two-level fault model; and (iii) Fault Control Machine, which controls the Fault Monitor and the FPM.

The adopted fault model classifies links in four situations: (i) not verified, (ii) faulty, (iii) operating correctly, or (iv) operating with fault tendency. This classification takes into account each link's monitoring history. There are static and dynamic mechanisms for testing the links quality. The static link test starts with *OsPhoenix* sending a control packet to the *HwPhoenix*. The Fault Control Machine (FCM) interprets this command broadcasting a predefined test packet to all output ports, except the local one. When a neighbor router receives the test packet, it loops back a packet with the same information. Then, the Fault Monitor detects whether the link is faulty or not, sets this information on the Fault Table and informs this procedure to the FCM, which sends a control packet containing the Fault Table to the *OsPhoenix* [8].

Each bidirectional link contains an HE and an HD to perform the dynamic link test, which is a strategy that identifies fault tendencies using circuits based on a threshold (similar strategy is used in [19]). The HD receives the data plus the redundancy bits encoded by the HE of the adjacent router. The HD module can correct one bit flip and detect at most two faults in a data flit. Thus, the module informs the communication status by the signals NE, EC, and ED.

Figure 4 illustrates the flit retransmission circuit placed in each data link between routers. This circuit uses HD information to verify if each flit was received without error, with an error that was corrected by the HD circuit, or if a double error was detected but not corrected. In the case of double error, the circuit of the target router requests a flit retransmission to the source router using the *retx* signal, implying a single clock of latency penalty (only if the *credit* signal is enabled, the source router may send flits to the target router).

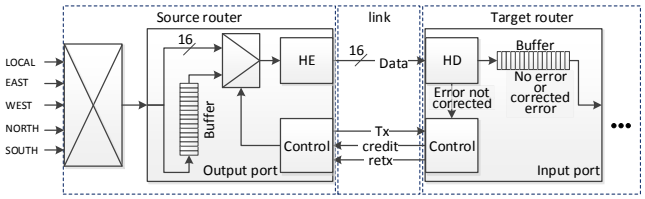

Figure 4 – Retransmission circuit.

Based on monitoring the density of errors, the FPM deduces a link fault tendency, which is propagated to the *OsPhoenix* that makes inferences to permanent errors or tendency of errors. According to these inferences, *OsPhoenix* may set on the Fault Table the bidirectional link as faulty (e.g. a permanent error) and/or may start the preprocessing of a new routing scenario. When a link is marked as faulty, the HE, HD and FPM modules are turned off and remain with this status until the *OsPhoenix* requires a new link evaluation.

4. SCENARIOS PROCESSING AND RETRANSMISSION FLOW

As soon as *OsPhoenix* is loaded, it commands the preliminary test of links. In the case of detecting faults, *OsPhoenix* decides whether to maintain a faulty link in operation, relying on the retransmission mechanism, or to perform several steps on all PEs and routers to establish routing configuration.

The FPM notifies the Fault Monitor whenever a faulty link or a fault tendency is detected. If this fault is annotated in the Fault Table, the information is not propagated. Otherwise, the Fault Monitor stores the fault information in the Fault Table and informs this event to the FCM that transmits this information to the CM of the local *OsPhoenix*. Then, the CM commands the SPM to proceed with next

fault tolerance steps (e.g. to process a new fault coverage scenario). Moreover, each *OsPhoenix* contains a timing mechanism to define a maximum time for network stabilization, which is reached when all *OsPhoenix* instances receive the same fault information. This mechanism is used when a sequence of faulty links split the NoC precluding the transmission of control packets to all routers [8].

When a message of fault tendency is received, the SPM verifies the existence of some previously computed scenario that covers this fault. If it exists, no further action is necessary. Otherwise, aiming to enable fast routing reconfiguration, this module computes and stores in the SRT Memory, together with the associated routing tables, a new set of scenarios that cover this fault. However, the amount of fault scenarios raises exponentially with the quantity of faulty links. Aiming to deal with this complexity, *OsPhoenix* preprocesses a limited set of scenarios based on a dissimilarity method using cross-correlation of fault matrices to meet the application requirements [20]. The preprocessing approach may be employed to fulfill several application requirements (e.g. to reduce power dissipation and to achieve homogeneous thermal distribution). Nevertheless, *OsPhoenix* uses latency minimization as an application requirement and employs Average Routing Distance (ARD) as a metric for fast latency estimation.

When a faulty link notification is received, *OsPhoenix* decides whether to maintain the NoC operation or trigger a routing reconfiguration process. If the reconfiguration process is chosen, and if the fault scenario is already preprocessed, the *OsPhoenix* may perform a fast routing reconfiguration, merely updating the Routing Table. Otherwise, the routing reconfiguration takes much longer, requiring the computation of a new coverage scenario, which delays the application execution.

Phoenix takes into account the premise that "all PEs have the same algorithm to generate scenarios and routing paths". This premise allows that each *OsPhoenix* has its GFT and SRT Memory, and all instances of *OsPhoenix* operate independently in a distributed way, eliminating the need for broadcast routing configurations.

5. BASICS OF RETRANSMISSION

Since the HD module may repair only single faults, but double faults requires flit retransmission, we consider the *Double Fault Error Rate* (*DFER*) as a metric that increases the transmission delay. *DFER* is a real number in the interval [0, 1]. The amount of link faults grows with the value of *DFER*; i.e.; if *DFER* = 0, the link is free of double faults, while if *DFER* = 1, means that all communications on the link are faulty. The smart decision process considers the tradeoff of dealing with retransmissions (maintaining a faulty link, i.e., DFER > 0) and issuing a global NoC routing reconfiguration.

There are four phases encompassing a reconfiguration event: (i) T_{FD} (fault detection delay) – detection of a faulty link through the FPM operation; (ii) T_{FP} (fault propagation delay) - fault propagation to all *OsPhoenix* of other PEs; (iii) T_{RP} (routing processing delay) - routing tables processing by all the *OsPhoenix*; and (iv) T_{RTL} (routing table loading delay) - loading the new routing tables into each local router. Equation 1 illustrates the summation of this reconfiguration delay (T_R):

$$T_R = T_{FD} + T_{FP} + T_{RP} + T_{TL} \quad (1)$$

Equation 2 illustrates the retransmission delay A_R as the reason for the sum of the amount of communication flows N_{comm} of a target application, considering the number of flits N_{Fi} of each i-th flow; its corresponding quantity of clock cycles employed in flit retransmission N_{CT}; and the sum of the *DFERs* of all links in the path of a j-th communication flow ($\sum_{j=1}^{L_{flow}} DFER_j$). This retransmission overhead is the percentage of added average latency for a given end-to-end communication.

$$A_R = \frac{\sum_{i=1}^{N_{comm}} \left[N_{Fi} \times N_{CT} \times \sum_{j=1}^{L_{flow}} (DFER_j) \right]}{N_{comm}} \quad (2)$$

In the event of a new fault, the *OsPhoenix* must perform the smart reconfiguration process, to reconfigure the system or maintain the failed channel in use, using the retransmission circuit. Considering these conditions, *OsPhoenix* issues a reconfiguration event when $T_R < A_R$, meaning that the average time taken for reconfiguration should be lower than correcting double faults at a set of links, considering communication flows.

6. METHOD AND EXPERIMENTS

The fabrication process variability of VLSI circuits increases every scale down of new deep submicron technologies, due to phenomena such as imprecise impurity deposition and non-uniformity in lithography exposure field. This variability may deviate the circuit from its nominal specification, or even prevent its partial or total operation [21]. In other words, it is a source of static and dynamic faults. Therefore, and without lack of generality, we choose the variability model proposed by Hargreaves et al. [22] for generating the fault scenarios presented here. This model takes into consideration the effect of variability on switch-to-switch link delay employing two variations parameters: the link delay variability σ and the spatial correlation variability λ.

The link delay variability were set to 5% (σ = 0.05) and 18% (σ = 0.18) to explore scenarios with 65 nm and 22 nm manufacturing processes, respectively, as predicted by the ITRS roadmap [23]. Additionally, the experiments were produced with λ = 0.4 and λ = 1.2, representing the high and low strength of the spatial correlation variability, respectively. These values represent the typical correlation induced by fabrication processes [22]. The experiments encompass four NoC sizes (5×5, 6×6, 7×7 and 8×8). Figure 5 shows examples of a 5×5 NoC combining link delay and spatial correlation parameters (each link is marked with its *DFER*).

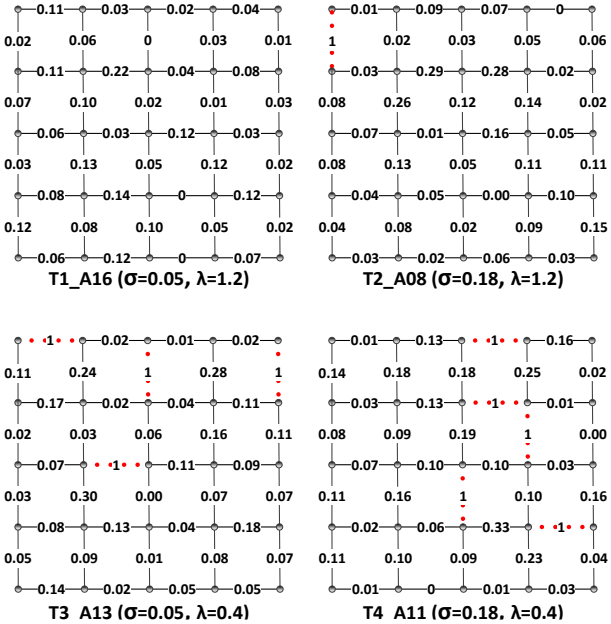

Figure 5 – 5×5 Mesh NoC example of *DFER* rates on links.

To explore the randomness of the variability model, we generated 25 times each one of these scenarios, resulting in 400 irregular NoC topologies. Figure 6 shows how simulation scenarios are composed and applied to achieve the experimental results. For each scenario, an in-house tool performs the following steps: (i) segmentation of the NoC using the segment routing algorithm [5] to generate a restriction file. This file contains all the forbidden directions that avoid deadlock situations; and (ii) computation of minimal paths using the restriction file information, which allows generating the set of virtual regions for the RBR approach.

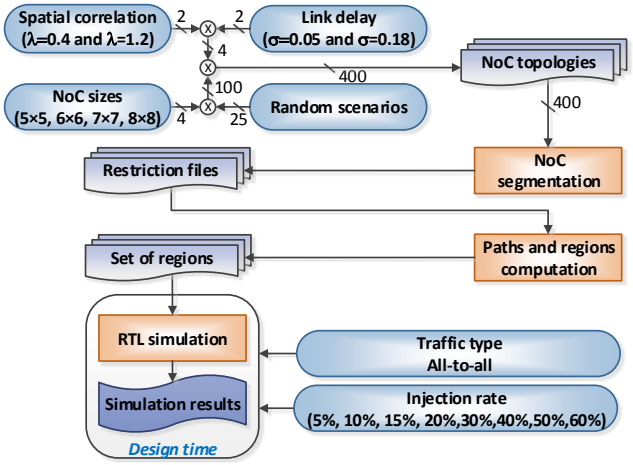

Figure 6 – Setup of experimental results.

We performed the experiments over NoCs described in VHDL RTL using eight injection rates (5%, 10%, 15%, 20%, 30%, 40%, 50% and 60%) of a synthetic traffic composed of 100 packet with 50 flits long. Additionally, all experiments are synthetic with all-to-all uniform traffic distribution.

7. EXPERIMENTAL RESULTS

The first set of experiments evaluates A_R that depends on the NoC size and on the *DFER* of each channel - that varies based on the fault distribution, as described in Section 6. Figure 7 presents the results for four sizes of NoCs (5×5, 6×6, 7×7 and 8×8) and average results acquired from 50 samples with each fault distribution configuration ([σ=0.05, λ=1.2], [σ=0.18, λ=1.2], [σ=0.05, λ=0.4] and [σ=0.18, λ=0.4]), for 10% of traffic injection rate.

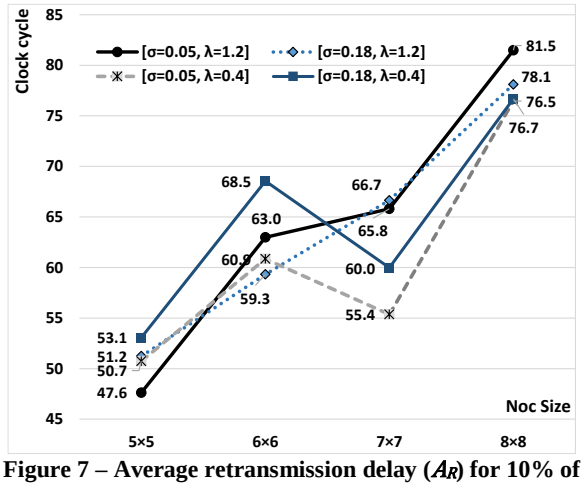

Figure 7 – Average retransmission delay (A_R) for 10% of traffic injection rate and 4 NoC sizes.

The experiments show that the A_R does not depend on the traffic injection rate, since the retransmission delay is computed according to a static quantity of packets. Besides, it is observed that the NoC size together with the fault distribution model affect A_R a lot. As the fault distributions become more aggressive, the *DFER* increases, causing the increase of A_R - i.e., more network latency. Each router determines the *DFER* values of its entire links at runtime by checking the fault tables generated by the FPM, as described in [24], since the single and double fault events are registered in each Fault Table. This dynamic information is propagated to *OsPhoenix* that verifies if it is necessary to reconfigure the NoC.

The second set of experiments calculates the same simulation results in design time by evaluating all the communication paths, while considering the fault rates generated by the fault distribution model; but now considering eight injection rates. Figure 8 illustrates the comparison between the estimated and measured results.

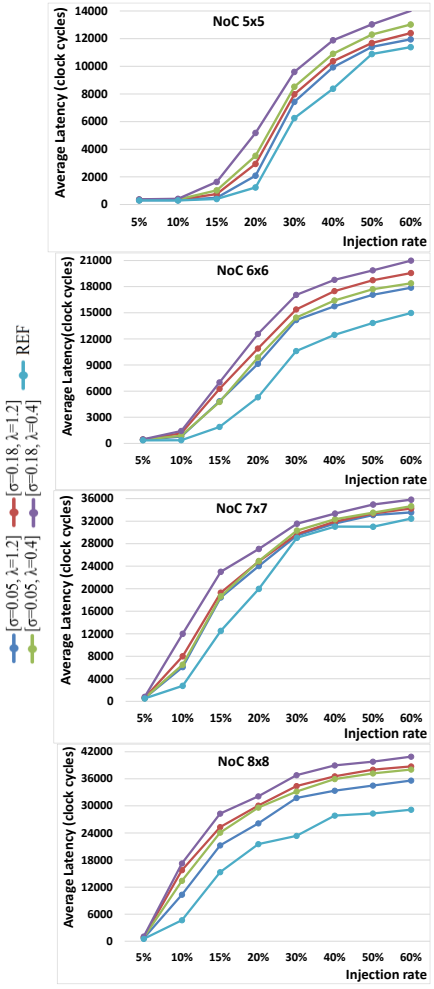

Figure 8 – Comparison of average latency measured and estimated with eight injection rates and four NoC sizes.

We notice that latency increases with more aggressive fault distribution configurations, where REF is a non-double faults scenario. This latency increase is explainable since more double-faults increases retransmission delay increasing the over latency. We measure the average error near 10%, which is due to the adaptability of the RBR routing mechanism. As there may exist more than one routing possibility, different choices yield disparate results. More aggressive fault distributions also result in greater variation, since the samples have faults packet more closely resulting in more complex segmentation configurations, creating more alternative routing options. Even though our decisions are

coherent, estimating the A_R at runtime is not viable since there is no way for the network to map every path taken for every source-destination pair. Thus, it is necessary to assume an average value that represents traffic type and distribution, for the A_R estimation.

The third set of experiments explores the over latency metric that represents the extra latency caused by retransmission delay, due to the additional cycles from retransmission circuit. Figure 9 shows the percentage of additional latency, which is defined as the value increase considering the same network topology without double errors (i.e., REF), in percentage.

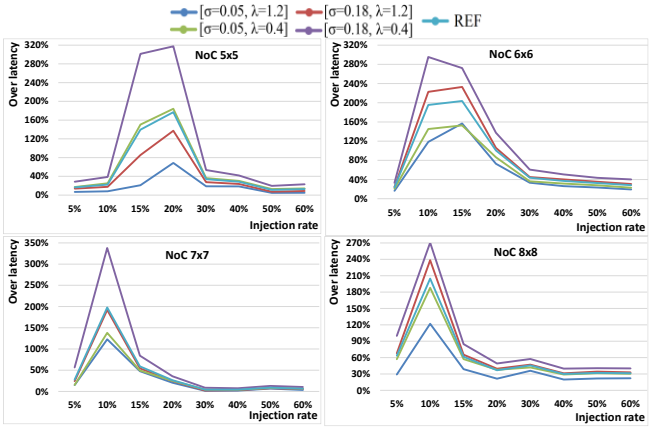

Figure 9 – Additional latency for uniform traffic in 5×5, 6×6, 7×7 and 8×8 NoCs for all 4-fault distributions.

Before and after the saturation point all curves have similar behavior and values, with less than 70% of additional latency for almost all injection rates. However, between 5% and 10% of traffic injection rate starts a saturation process, where the extra latency increases severely. This fact can be explained due to the chaotic nature that the traffic present during the network saturation process. Thus, for analysis and decisions taken during the operation of Phoenix, we consider the additional latency values before the saturation point.

8. CONCLUSIONS

This work proposes a smart reconfiguration approach for irregular NoC topologies using Phoenix's architecture. The software of Phoenix is a small part of the operating system kernel called *OsPhoenix* that take decisions as soon as a fault prediction monitor, placed on each link of each router, detects a fault tendency. The hardware of Phoenix is a fault-tolerant mesh NoC, which employs region-based routing mechanism. The NoC possesses a fault prediction module with Hamming encoding, enabling simple error correction and the detection and retransmission of double errors. Using this mechanism, the *OsPhoenix* may decide to maintain some links operating with error correction or reconfigure the NoC's routing tables to avoid these links.

Retransmitting data in faulty links inserts additional latency, reducing network performance. However, careful evaluation is necessary for issuing a global NoC reconfiguration to avoid faulty links, because it also affects the application performance, once the average distance of routing tends to increase with the reduction of operating links.

It is our understanding that retransmission's impact is too severe. The retransmission of data generates around of 70% of additional latency (considering the fault model used), whereas the latency due to the replacement of preprocessed scenarios is between 4% and 20%. Thus, in situations where fault distributions are too aggressive, the best approach is to stop the network operation to issue a reconfiguration. We also intend to evaluate scenarios with lower fault rates for determining, at runtime, whether a retransmission or reconfiguration is the most appropriate approach.

9. REFERENCES

[1] International Technology Roadmap for Semiconductors (ITRS). ITRS 2013 Edition. Available in: www.itrs.net/reports.html.

[2] R. Marculescu et al. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. TCAD, v.28, n.1, pp.3-21, Jan. 2009.

[3] S. Rodrigo et al. Cost Efficient On-Chip Routing Implementations for CMP and MPSoC Systems. TCAD, v.30, n.4, pp.534-547, Apr. 2011.

[4] E. Ioannidis et al. Evolution of Low Frequency Noise and Noise Variability through CMOS Bulk Technology Nodes from 0.5 µm down to 20 nm. Solid-State Electronics, v.95, pp.28-31, May 2014.

[5] A. Mejia et al. Region-Based Routing: A Mechanism to Support Efficient Routing Algorithms in NoCs. IEEE Transactions on VLSI Systems, v.17, n.3, pp.356-369, Mar. 2009.

[6] K. Aisopos et al. ARIADNE: Agnostic Reconfiguration in a Disconnected Network Environment. PACT, pp.298-309, Oct. 2011.

[7] A. DeOrio et al. A Reliable Routing Architecture and Algorithm for NoCs. TCAD, v.31, n.5, May 2012.

[8] C. Marcon et al. Phoenix NoC: A distributed fault tolerant architecture. ICCD, pp.7-12, 2013.

[9] C. Feng et al. Addressing Transient and Permanent Faults in NoC with Efficient Fault-Tolerant Deflection Router. IEEE Transactions on VLSI Systems, v.21, n.6, pp.1053-1066, Jun. 2013.

[10] Z. Ying et al. Fault-tolerant schemes for NoC with a network monitor. ISCIT, pp.1083-1086, 2010.

[11] Q. Yu, P. Ampadu, A Dual-Layer Method for Transient and Permanent Error Co-Management in NoC Links. IEEE Transactions on Circuits and Systems II: Express Briefs, v.58, n.1, pp.36-40, Jan. 2011

[12] M. Radetzki et al. Methods for fault tolerance in networks-on-chip. ACM Computing Surveys. v.46, n.1, pp.8:1-8:38, 2013.

[13] M. Palesi, S. Kumar, R. Holsmark. A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architectures. SAMOS, pp.373-384, 2006.

[14] E. Bolotin et al. Routing Table Minimization for Irregular Mesh NoCs. Design, DATE, pp, 16-20, 2007.

[15] Y. Fukushima, M. Fukushi, I. Yairi, A Region-Based Fault-Tolerant Routing Algorithm for 2D Irregular Mesh Network-on-Chip, Journal of Electronic Testing, v.29, n.3, pp.415-429, May 2013.

[16] R. Holsmark, M. Palesi, S. Kumar. Deadlock-Free Routing Algorithms for Irregular Mesh Topology NoC Systems with Rectangular Regions. JSA, v.54, n.3-4, pp.427-440, Mar.-Apr. 2008.

[17] D. Fick et al. Vicis: A Reliable Network for Unreliable Silicon. DAC, pp.812-817, 2009.

[18] F. Triviño et al. Network-on-Chip Virtualization in Chip-Multiprocessor Systems. JSA, v.58, n.3-4, pp.126-139, Mar. 2012.

[19] L. Dai et al. Monitoring Circuit Based on Threshold for Fault-tolerant NoC. Electronics Letters, v.46, n.14, pp.984-985, 2010.

[20] J. Silveira et al. Preprocessing of Scenarios for Fast and Efficient Routing Reconfiguration in Fault-Tolerant NoCs. PDP, pp.404-411, 2015.

[21] M. Shintani et al. A Variability-Aware Adaptive Test Flow for Test Quality Improvement. TCAD, v.33, n.7, pp.1056-1066, Jul. 2014.

[22] B. Hargreaves, H. Hult, S. Reda. Within-die Process Variations: How accurately can they be statistically modeled? ASP-DAC, pp.524-530, 2008.

[23] International Technology Roadmap for Semiconductors (ITRS). ITRS 2007 Edition. Available in: www.itrs.net/reports.html.

[24] J. Silveira et al. A fault prediction module for a fault tolerant NoC operation. ISQED, pp.284-288, 2015.