Preprocessing of Scenarios for Fast and Efficient Routing Reconfiguration in Fault-Tolerant NoCs

Jarbas Silveira [1], César Marcon [2], Paulo Cortez [1], Giovanni Barroso [3], João M. Ferreira [1], Rafael Mota [1]
[1] LESC-DETI, Federal University of Ceará, Fortaleza, Brazil - 90455-970
[2] PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil - 90619-900
[3] Department of Physics, Federal University of Ceará, Fortaleza, Brazil - 90455-970
jarbas@lesc.ufc.br, cesar.marcon@pucrs.br, cortez@deti.ufc.br

***Abstract*—Newest processes of CMOS manufacturing allow integrating billions of transistors in a single chip. This huge integration enables to perform complex circuits, which require an energy efficient communication architecture with high scalability and parallelism degree, such as a Network-on-Chip (NoC). However, these technologies are very close to physical limitations implying the susceptibility increase of faults on manufacture and at runtime. Therefore, it is essential to provide a method for efficient fault recovery, enabling the NoC operation even in the presence of faults on routers or links, and still ensure deadlock-free routing even for irregular topologies. A preprocessing approach of the most probable fault scenarios enables to anticipate the computation of deadlock-free routings, reducing the time necessary to interrupt the system operation in a fault event. This work describes a preprocessing technique of fault scenarios based on forecasting fault tendency, which employs a fault threshold circuit and a high-level software that identifies the most relevant fault scenarios. We propose methods for dissimilarity analysis of scenarios based on measurements of cross-correlation of link fault matrices. At runtime, the preprocessing technique employs analytic metrics of average distance routing and links load for fast search of sound fault scenarios. Finally, we use RTL simulation with synthetic traffic to prove the quality of our approach.**

Keywords—NoC; routing; irregular topology; fault-tolerance.

I. INTRODUCTION

The evolution of Very-Large-Scale Integration (VLSI) semiconductor technology enables to integrate hundreds or even thousands of Processing Elements (PEs) 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 Network-on-Chip (NoC) technology plays a key role in the implementation of these highly integrated SoCs. Mesh is the most popular NoC topology, which offers simple and regular structure, and small wire length being suitable to a tile-based design. Under the NoC paradigm, routers and links interconnect each PE placed into a tile to other PEs, and usually the communication is performed by packet transmission [1].

Recent submicron technologies provide more process variability increasing the quantity of defective components [2]. A defective router or link may ruin the mesh communication structure leading to an irregular topology (i.e. topologies derived from regular mesh networks with faulty links) [3]. Therefore, static and deterministic routing algorithms tailored to a regular NoC topology will not operate properly, thus rendering the chip useless. To avoid this problem, the design of routing algorithms has to provide some degree of fault-tolerance while ensuring deadlock-free operation. In order to fulfil this goal, our work employs an approach based on turn prohibition [4], which avoids deadlock by eliminating a subset of network turns. Our work employs a technique similar to [3], which compresses the routing table, placed in each router, according to NoC regions.

Aiming to provide support for dynamic faults, a reconfiguration approach, where each PE preprocesses and stores scenarios, enables to reduce the time that the network is halted waiting for the computation a new deadlock and fault free routing. The preprocessing of all possible fault-scenarios is a very complex problem, which is timing and memory consuming. Fortunately, some fault scenarios cover others, enabling to minimize the storage area. The fault-tolerant system may elect and store only some scenarios to be used only when monitors detect fault situations.

This paper proposes an efficient approach to deal with dynamic faults on NoC links employing Phoenix [5] as the fault-tolerant target system, which comprises a 2D mesh NoC and a layer of an Operating System. The approach consists of preprocessing fault scenarios to fast reconfigure the network with a new deadlock-free routing in case of fault detection. This method comprises of fault-monitors placed in each input port of the Phoenix NoC's routers. Each fault-monitor evaluate whether the link is faulty or if there is a fault tendency. This information is transmitted to the PE locally connected to the router, and a communication driver placed inside the Operating System preprocesses new scenarios based on this fault information.

This paper is organized as follows. Section II presents related works. Section III describes the Phoenix target architecture. Section IV describes the processing flow of fault-tolerant scenarios. In Section V, we present the fundamentals of preprocessing scenarios. Section VI and VII describe the setup and the experimental results, respectively. Finally, Section VIII discusses the main conclusions of this work.

II. RELATED WORKS AND MAIN CONTRIBUTIONS

A reconfigurable fault-tolerant system for irregular networks may employ solutions with or without virtual channels requiring mechanisms for (i) fault detection and diagnose; (ii) propagation of fault recognition; (iii) deadlock-free routing computation; and (iv) routing reconfiguration. Virtual channel approaches enable to explore more routing

paths. However, the virtualization engine consumes a lot of area; consequently, some researches exploit techniques to minimizes this consumption (e.g. [6][7]). Approaches without virtual channels consume less area, but reduce the exploration of the routing paths, since these approaches use prohibiting turn methods to avoid deadlocks. This section discusses works related to the routing mechanisms and the routing reconfiguration, which are the focus of this work, considering only network architectures without virtual channels.

The *routing mechanism* defines how the routing information is stored and processed. In order to support reconfiguration, the routing mechanism needs dynamic resources, which normally encompasses tables placed in each NoC's router that store routing information. However, the routing tables do not scale with the increase of NoC dimensions. Aiming to reach this scalability, several works employ techniques to compress or minimize the table size. These techniques are complex and may imply loss of performance and/or impossibility of reach all target nodes, even when these nodes are potentially reachable. Examples of these works are (i) Palesi, Kumar and Holsmark [8], which use a table compression technique applied to application-specific routing; and (ii) Bolotin et al. [9] that use a table minimization technique applying a fixed function combined with minimal deviation tables. The scalability is also achieved dividing the network in regions and producing routing tables entries that are related to these regions. For instance, Mejia et al. [3] proposed the Region Based Routing (RBR) approach, where each node contains a set of regions based on the computed paths to cover all source and target nodes. Fukushima, Fukushi and Yairi [10] propose another region based approach founded on a set of rectangular faulty regions and corresponding deviation paths, which are employed to avoid the faulty regions. Their approach improves the work of Holsmark, Palesi and Kumar [11] providing a deadlock-free routing that reduces the faulty regions size and, consequently, reducing the number of nodes to be disabled and the routing implementation complexity. This work employs an approach similar to RBR, with a technique to reduce the size of RBR tables stored on each PE.

The *reconfiguration process* defines the computation cost of dynamic routing decisions that are implemented in hardware, in software or in both. Fick et al. [12] describe a fault-tolerant NoC that employs routing reconfiguration on router and network levels. The two-level reconfiguration allows to deal with some in-router faults at the router level, simplifying the reconfiguring and allowing graceful degradation of the network performance, while increases the quantity of network faults. Feng et al. [13] propose a fault-tolerant solution for a bufferless NoC including the detection of both transient and permanent faults. They propose the reconfiguration of routing tables during packet transmission through the Reconfigurable Fault-Tolerant Deflection Routing (FTDR) algorithm. In addition, they present a hierarchical FTDR (FTDR-H) algorithm, in order to reduce the area overhead of the FTDR router. Experimental results show that FTDR and FTDR-H can protect against fault distribution pattern if the NoC is not split into two or more disconnected sub-networks. Although the dissimilarities of the works of Fick et al. [12] and Feng et al. [13], both are conceptually similar, suggesting that hierarchical implementation is a sound approach to minimize the reconfiguration process. Strano et al. [14] suggest the use of the Overlapped Static Reconfiguration, which is an old technique used in off-chip networks, but now adapted to the NoC restrictions. Their approach enables to inject new packets in the network while routing or dropping old packets. The authors use token signals to separate old packets from the new ones; and during the token signal propagation, the NoC remains with both configurations, the previous and the new one. Their approach enables fast recovery and proves shorter time to reload and to have the NoC working properly, but imply complex hardware implementation.

Our work employs a hardware/software reconfiguration approach based on preprocessing the most probably faults scenarios, which are computed according to the detection of link fault tendency. Our approach can employ more complex and time-consuming algorithms, implemented in software, aiming to produce optima solutions for large NoCs without compromising the execution time, once the routing tables would be already preprocessed. Moreover, in a given set of scenarios, some ones can cover others, reducing a large set of preprocessing scenarios, which provides two new and important contributions of this work: (i) a novel way to reduce a large set of scenarios based on cross-correlation measure to identify dissimilarities in sets of irregular matrix topologies, which enables to minimize the storage area of preprocessing scenarios; and (ii) a method to identify the most similar scenarios, and, consequently, the best scenario substitution, which enables to search fast and efficient routing solutions.

III. PHOENIX'S ARCHITECTURE

Figure 1 shows the Phoenix distributed fault-tolerant architecture [5] over an NoC-based MPSoC platform comprising 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 composed by a pair processor-memory. Additionally, each PE is connected through a NoC interface to the local port of each router.

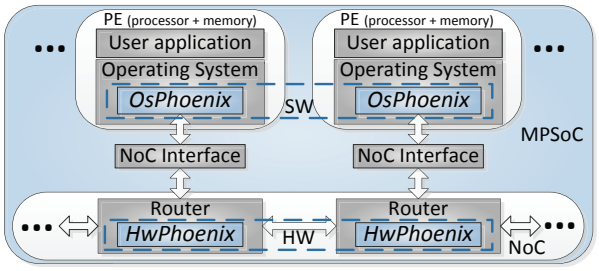


Figure 1 – Distributed and fault-tolerant Phoenix architecture [5].

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) the data packet, which carries the messages of each PE; and (ii) the control packet, which controls the fault-tolerant mechanisms. The *OsPhoenix* communicates with the *HwPhoenix* via bidirectional control packets transmitted through the local port of each PE.

Examples of control packets are (i) TEST_LINKS used to test all links of the local router; (ii) TR_ROUT_TAB, RD_ROUT_TAB and WR_ROUT_TAB used to transmit, to read and to write the *Routing Table* of each router, respectively; and (iii) TR_FAULT_TAB, RD_FAULT_TAB and WR_FAULT_TAB, which are analogous to the previous commands, but taking into account the *Fault Table* of each router.

This work aggregates to the previous Phoenix architecture [5], the follow contributions: (i) fault detection and correction modules encompassed by a *Hamming Encoder* (HE) and a *Hamming Decoder* (HD); (ii) a fault diagnosis circuit named *Fault Prediction Module* (FPM); and (iii) the scenarios preprocessing capability, providing fast reconfiguration and deadlock-free routing in case of fault detection.

A. OsPhoenix Fundamentals

The *OsPhoenix* is a software layer placed into the PE's Operating System, which contains drivers for high-level operation and routines that implement the distributed fault-tolerant mechanisms. The *OsPhoenix* is perceived by the PE's Operating System as a network driver interface. It makes the fault-tolerant mechanism of the NoC transparent to the system operation. The Figure 2 depicts the main modules of *OsPhoenix* and their interaction.

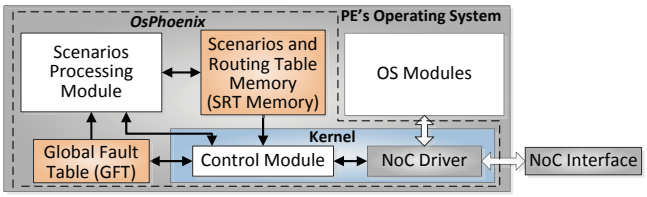


Figure 2 – *OsPhoenix's* architecture. The dashed line encompasses *OsPhoenix* containing five main elements (i.e. memories and processing modules) and black arrows show how elements are interrelated.

The *OsPhoenix's Kernel* comprises (i) the *Control Module* that concentrates the control of the fault-tolerant mechanism; and (ii) the *NoC Driver*, containing the routines to convert logical addresses of the application messages to physical addresses of the network and vice-versa. Additionally, this driver makes transparent the fault-tolerant mechanisms to the PE's Operating Systems, since it captures and transmits control packets to the *Control Module*, and exchanges data packets directly with the *NoC Interface*.

The *Global Fault Table* (GFT) stores the status of all NoC links (i.e. not tested, faulty, with fault tendency or operating properly). It is a global copy of all routers' *Fault Table* (next described). The *Control Module* writes/reads this table to synchronize the fault link status of the *OsPhoenix* of all PEs.

The *Scenarios Processing Module* computes routing tables according to the fault or tendency of fault on links, when commanded by the *Control Module*. It uses the fault links status provided by the GFT together with new faults information to search a previous computed scenario, which covers this new fault situation, in the *Scenarios and Routing Table Memory* (*SRT Memory*). If a candidate scenario is found, the related *Routing Table* covering a new routing is transmitted to the *HwPhoenix*. Otherwise, this module preprocesses a new fault-tolerant scenario and the RBR Table.

Phoenix platform takes into account the premise that "all PEs have the same algorithm to generate scenarios and routing paths". This premise allows that all *OsPhoenix* (i.e. one per PE) operate independently and distributedly, each *OsPhoenix* has its own GFT and its own *SRT Memory*. This also prevents that the local *OsPhoenix* should propagate tables containing routing algorithm through the network, since each router has its own *Routing Table*.

B. HwPhoenix Fundamentals

Phoenix's NoC employs a direct 2D mesh topology with M lines and N columns, consisting of M×N routers using bidirectional links to connect with other routers and PEs. The NoC employs tables for source routing decisions and the *OsPhoenix* performs routing algorithms to fill the *Routing Table* according to the position of the PEs and the faulty links. Further, Phoenix NoC implements wormhole switching, which divides packets into flits that demands only small buffers for the necessary data storing.

Figure 3 shows the Phoenix's router architecture, which includes mechanisms for packet routing and fault-tolerance.

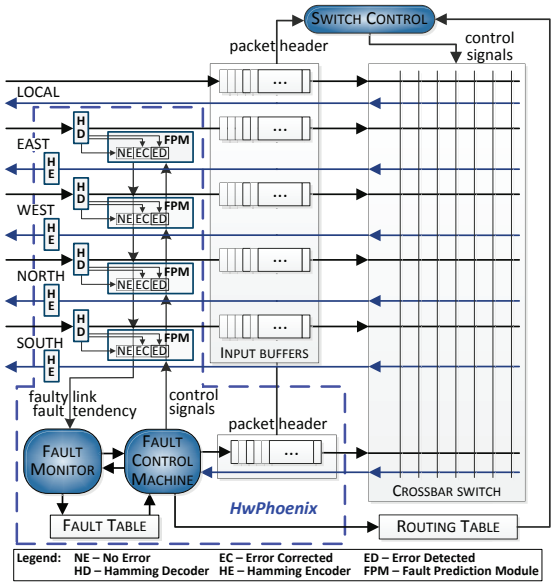


Figure 3 - The basic components of Phoenix's router architecture. The main components of *HwPhoenix* are surrounded by dashed lines.

The routing mechanism of Phoenix's router includes: (i) four bidirectional ports (NORTH, SOUTH, EAST and WEST) to interconnect routers and a bidirectional port (LOCAL) that connects the router with its local PE. All input links contain configurable buffers used when packets congest the routing path; (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* circuit that performs packets routing and arbitration according to the packet header and to the *Routing Table* content. The arbitration follows a dynamic rotating policy to ensure that all incoming requests are processed, avoiding the starvation phenomenon.

Phoenix's NoC employs source routing, where paths are computed according to the *Routing Table*, which is initialized

with XY routing. Nonetheless, depending on the occurrence of faults, *OsPhoenix* provides a new deadlock-free routing modifying the *Routing Table*. The NoC routing algorithm is similar to RBR [3], which groups target addresses into regions, in order to reduce the *Routing Table* size. In addition, the *Routing Table* provides several paths, even in the presence of faults, with a minimum of four regions.

The fault-tolerant mechanism implemented in each router of the *HwPhoenix* includes: (i) a fault detection and correction module containing *Hamming Encoder* (HE), *Hamming Decoder* (HD) and *Fault Prediction Module* (FPM) that are placed in each one of the bidirectional links that interconnect routers; (ii) the *Fault Monitor* that communicates with FPM to set the status of the links on the *Fault Table* according to a two-level fault model; and (iii) the *Fault Control Machine*, which controls the *Fault Monitor* and the FPM, and communicates via control packets with the *OsPhoenix* and with the *Fault Control Machine* of other routers.

Figure 4 depicts the two-level fault model implemented in each Phoenix's router. The first level is a 4-field vector, where each field stores the operation status of the NORTH, SOUTH, EAST and WEST links, containing two bits to inform whether the link is (i) not verified, (ii) faulty, (iii) operating properly, or (iv) operating with fault tendency. The second level improves the previous Phoenix implementation [5] providing information about the quality of link once it is operating properly. The second level complements the first one, providing extra information about the quality of link. This second level, which is physically placed inside each FPM, contains counters with the operation status of the output link: (i) No Error (NE), (ii) Corrected Error (CE) and (iii) Detected Error (DE). In the example of Figure 4, the counters have different lengths in order to implement a window of events to capture probabilities of NE, CE and DE.

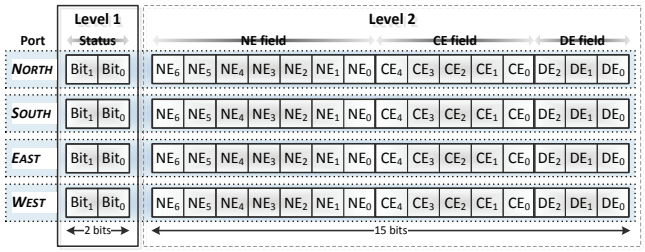


Figure 4 – The two-level fault model (i.e. Level 1 - Fault Table; Level 2 – Link status counter).

The Phoenix tests all links through static and dynamic mechanisms, which are independent, but complementary.

The static link test starts with *OsPhoenix* sending the TEST_LINKS control packet to *HwPhoenix*, through the local port of the router. The *Fault Control Machine* 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 *Faulty Table* and informs this status to the *Fault Control Machine* that sends a TR_FAULT_TAB control packet to the *OsPhoenix* containing the *Faulty Table* [5]. The static mechanism sets only the Level 1 of the fault model with "faulty" or "operating properly" status. Typically, the *OsPhoenix* sends the TEST_LINKS control packet when the system is started, or asynchronously by a high-level command generated on the application layer.

In order to accomplish the dynamic link test, each bidirectional link contains a HE and a HD to perform a strategy that identifies fault tendencies based on thresholds of fault occurrences, which similar to the one used in [15]. The HD, placed on the input of the buffer, receives the data and the redundancy bits, that was encode by the HE in 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 communications quality by the signals NE, EC and ED. In the experimental setup, the HE module generates the redundancy bits according to (16, 5, 1) Hamming code: 16 data bits, 5 bits of data redundancy and 1 bit of capability correction. Based on monitoring the density of acknowledges and negative-acknowledges (ACKs/NACKs), an error detection and correction circuit distinguishes between transient and non-transient faults, and the FPM measures the density of errors corrected and deduces a possible link fault tendency. This fault tendency information is propagated to the *OsPhoenix* that makes inferences to permanent errors or tendency of errors. According to these inferences, *OsPhoenix* may set on Level 1 of *Fault Table* the bidirectional link as faulty and/or start preprocessing a new routing scenario that avoid the use of the faulty link. When a link is marked as faulty, the HE and HD modules are turned off and remains with this status until the *OsPhoenix* requires a new link evaluation.

IV. SCENARIOS PROCESSING FLOW

The Phoenix NoC starts with all local routing tables configured to operate with the XY routing algorithm. As soon *OsPhoenix* is loaded, the preliminary link tests are commanded. If all links are working properly, the network starts operating. In the case of detecting faults, the *OsPhoenix* and the *HwPhoenix* perform several steps on all PEs and routers to establish a new routing algorithm that allows a deadlock-free data communication [5]. Aiming to capture dynamic faults during the network operation, the FPMs search for faults and fault tendencies in all NoC's links. The information of these faults is transmitted to *OsPhoenix* for an appropriated handling. The Figure 5 illustrates a Message Sequence Chart (MSC) containing the steps taken after a FPM detects a fault link.

The FPM notifies the *Fault Monitor* whenever a faulty link or fault tendency is detected. If this fault is already 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 *Fault Control Machine* that propagates this event to the *OsPhoenix* through a control packet containing the *Fault Table*.

Inside the *OsPhoenix* kernel, the *NoC Driver* module captures the control packet directing to the *Control Module* that checks if the fault is already annotated in the GFT. In case of positive checking, the *Control Module* considers that the problem is already known and treated, ending the fault processing, i.e. no other message is generated. Otherwise, the *Control Module* updates the GFT, and through a control

packet, requests that the *Fault Control Machine* propagates, in broadcast to all neighbor routers, control packets containing the fault information. Simultaneously, the *Control Module* requests the *Scenarios Processing Module* to proceed with next fault-tolerance step (e.g. to process a new fault scenario).

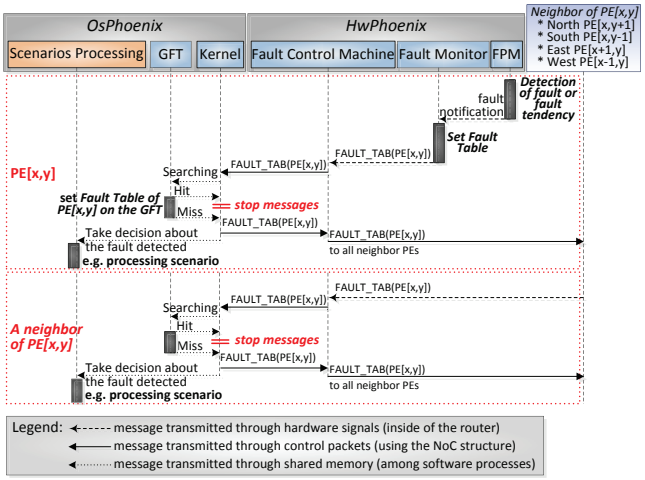


Figure 5 – MSC diagram of the processing fault mechanism when a faulty link or a fault tendency is detected.

The control packets are propagated among neighbor routers until all *OsPhoenix* are notified. Any neighbor router that receives the control packet with fault information propagates the same information to its *OsPhoenix* following the same steps described above. Additionally, each *OsPhoenix* has a timing mechanism to define a maximum time for network stabilization, which is reached when all *OsPhoenixs* receive the same fault data. Phoenix NoC uses this mechanism when a sequence of faulty links splits the network precluding the transmission of control packets to all routers [5].

Figure 6 illustrates the fault-tolerance steps when the *Scenarios Processing Module* receives a fault message, which can be of two types: faulty link or fault tendency. When a message of fault tendency is received, the *Scenarios Processing Module* verifies if some scenario previously computed already covers the fault. In case of positive answer, no action is taken. Otherwise, in order 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, this paper proposes to preprocess a limited set of scenarios based on a dissimilarity method using cross correlation of fault matrices in order to meet the application requirements.

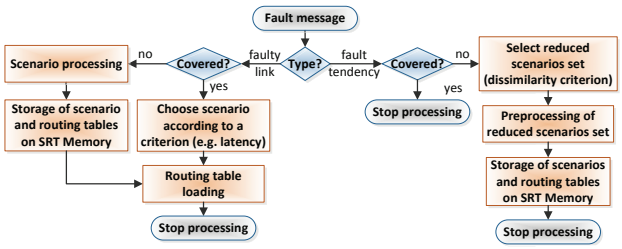


Figure 6 – *Scenarios Processing Module* operation.

When a message of faulty link is received, the decision of network recovery is preeminent because the FPM can no longer recover defective packets, preventing network communication. The *Scenarios Processing Module* checks if the scenarios stored in *SRT Memory* cover the fault. If the fault is covered, the *Scenarios Processing Module* chose the one that better fulfils the application requirement. Thus, the *OsPhoenix* performs fast routing algorithm reconfiguration with the *Scenarios Processing Module* sending a control packet to the *Control Module* containing the routing table that will be configured on the local router. Otherwise, the routing reconfiguration takes much more time. The *Scenarios Processing Module* starts computing the best set of coverage scenarios along with the minimum cost routing algorithm, and at that time configures the *Routing Table*, as described above.

V. Fundamentals of Preprocessing Scenarios

In a given set of fault scenarios, some ones cover others enabling to reduce the amount of scenarios to be stored. Let s_a and s_b be two fault scenarios, then s_a is a coverage scenario of s_b (the covered scenario) when all communications allowed in s_a are allowed in s_b, but the opposite is not necessarily true. Thus, a coverage scenario is equally or more restrictive than the covered one, and then a coverage scenario can be used in substitution of the covered one. The Figure 7(a) shows this situation, where a 5×5 mesh NoC has 4 links (l_1, l_2, l_3 and l_4) tending to fail. In order to cover all fault situation would be necessary to preprocess 16 scenarios. Figure 7(b) shows the best coverage scenario for faults on links l_1 and l_3. However, the scenarios presented in Figure 7(c) and (d) cover this same fault combination. Note that these two coverage scenarios have an extra faulty link (i.e. l_4 or l_2) and that the preprocessing mechanism need to choose which of these scenarios provides better performance for the system operation.

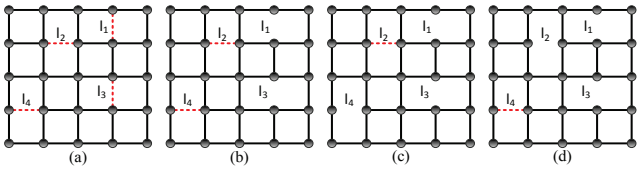


Figure 7 – A 5x5 mesh NoCs with four links with fault tendency (figure a) and three coverage scenarios (figures b, c and d).

When a given scenario covers others, the set of routing tables applied to the coverage scenario may be used on the routers of the covered one. Thus, in a fault occurrence, the *OsPhoenix* will find out, in a reduced set of scenarios, which are the stored scenarios that cover the fault links together with the associated *RBR Table* that was already computed.

A. Level of Similarity with Cross-Correlation Method

Aiming to select efficient preprocessed scenarios, this work uses two approaches based on the 2D cross-correlation method [16] that consider similarity and dissimilarity of scenarios.

The 2-D cross-correlation of an M-by-N matrix A with a P-by-Q matrix B is a matrix X of size M+P-1 by N+Q-1, such

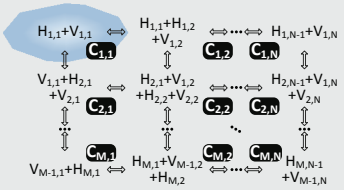


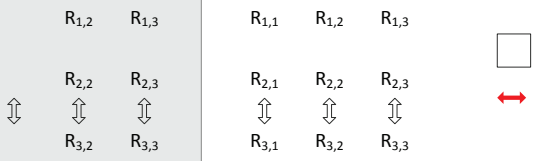
Figure 8 – Matrices employed on cross-correlation method: (a) illustrates matrices R, H and V, which are represented in a NoC fashion (routers are rectangles, and links are double arrows; (b) shows the matrix C, where each rectangle contains the sum of all links directelly connected to each router.

$R_{m,n}$ is a router and $C_{m,n}$ is a composition of links, both with coordinates m and n, $\forall\ 1 \le m \le M$ and $1 \le n \le N$. In addition, $H_{m,n}$ is an horizontal connection between $R_{m,n}$ and $R_{m,n-1}$ $\forall\ 1 \le m \le M$, $1 \le n \le N\text{-}1$; $V_{m,n}$ is a vertical connection between $R_{m,n}$ and $R_{m-1,n}$ $\forall\ 1 \le m \le M\text{-}1$, $1 \le n \le N$. If a link in the basic scenario has the same status (i.e. with or without fault) the element in H or V matrix is 1 otherwise is 0; and following the same rule, each element of C matrix is the sum of each link directly connected to the router.

This work uses 2D cross-correlation to compare scenarios in order to find similarities and dissimilarities. The method *LsHV* employs cross-correlation on vertical and horizontal links (i.e. matrices H and V) and the method *LsC* takes into account the joint effect of links in each router (i.e. matrix C). *LsHV* and *LsC* indicate the level of similarity between the scenarios, given by the Euclidian norm (represented by operator ||) of cross-correlation. Both levels of similarity are normalized by an auto cross-correlation (e.g. $||V_a * V_a||$). The highest level of similarity occurs when *LsHV* or *LsC* are equal to 1. Inasmuch as the value departs from 1, the level of similarity is reduced, increasing the level of dissimilarity. Equations 2 and 3 illustrate *LsHV* and *LsC*, respectively, taking into account two hypothetic scenarios *a* and *b*.

$$LsC_{ab} = \frac{||C_a * C_b||}{||C_a * C_a||}$$

Next, follows a synthetic example of 3×3 mesh NoC with 3 and 4 faulty links, preforming the basic scenario a and the evaluated scenario b, respectively, with the correspondent the *LsC* formulation.



$$C_a = \begin{bmatrix} 2 & 3 & 2 \\ 3 & 4 & 3 \\ 2 & 3 & 2 \end{bmatrix}, C_b = \begin{bmatrix} 2 & 3 & 2 \\ 3 & 3 & 2 \\ 2 & 3 & 2 \end{bmatrix}, C_a * C_b = \begin{bmatrix} 4 & 12 & 17 & 12 & 4 \\ 10 & 29 & 43 & 32 & 12 \\ 14 & 41 & 61 & 45 & 17 \\ 10 & 29 & 43 & 32 & 12 \\ 4 & 12 & 17 & 12 & 4 \end{bmatrix}, C_a * C_a = \begin{bmatrix} 4 & 12 & 17 & 12 & 4 \\ 12 & 34 & 48 & 34 & 12 \\ 17 & 48 & 68 & 48 & 17 \\ 12 & 34 & 48 & 34 & 12 \\ 4 & 12 & 17 & 12 & 4 \end{bmatrix}$$

$$LsC_{ab} = \frac{||C_a * C_b||}{||C_a * C_a||} = \frac{130.5517}{144.3460} = 0.9044$$

B. Reduction of Coverage Scenarios

Equation 4 computes the amount of links (Q_L) connecting all M×N NoC's routers (e.g. Q_L=112 links in an 8×8 NoC).

$$Q_L$$

Considering that all and every link may fail, Equation 5 computes the maximum quantity of fault scenarios (Q_S), which is the combination of all possibilities of faulty link.

$$Q_S \quad Q_L$$

As the number of scenarios grows exponentially with the quantity of faults, computing all possible scenarios is time and memory consuming, even for a small percentage of links, making this approach unfeasible. For instance, if 10% of the faulty links were considered in a 9×9 mesh NoC, i.e. 15 of faulty links, would be $Q_S = 2^{15}$ scenarios to be preprocessed. However, we employ three strategies to decrease the number of preprocessed scenarios: (i) differential treatment for static and dynamic faults, (ii) incremental fault scenarios, and (iii) dissimilarity approach.

The monitoring system classifies faults as static or dynamic. Faults classified as static determine an irregular NoC topology that is perceived by the *Scenarios Processing Module* as a basic topology. Consequently, independent of the quantity of static faults there will have only a single scenario representing a basic NoC topology. Over this basic topology, only the dynamic faults may perform temporary path changes, implying to compute new fault scenarios. Furthermore, the *OsPhoenix* can handle new routing tables, so that the *Fault Monitor* transmits a message of fault tendency.

While the amount of fault scenarios grows exponentially with the total number of faults, the amount of new fault scenarios that must be preprocessed is incremental, i.e. it is not necessary to recalculate the previously stored scenarios. Let Q_F be the quantity of dynamic faults previously know and Q_N be the quantity of new faults tendency just now detected, than Equation 6 calculates the quantity of new scenarios to be computed (Q_{SC}). For instance, if the system has 4 links with fault tendency already know (Q_F = 4) and the *Fault Monitors* detect two new links with fault tendency (Q_N = 2), the *OsPhoenix* needs to preprocess more 48 scenarios (i.e. $2^{(4+2)} - 2^4$), assuring the system will provide all fault scenarios.

$$Q_{SC} \quad Q_L \quad Q_F \quad Q_F$$

Identifying the most dissimilar scenarios that cover the same fault scenario allows saving a reduced set of scenarios

with a wider coverage. This wider and reduced coverage is achieved sorting the new scenarios according to the dissimilarity level and storing only a percentage of the most dissimilar ones. When the percentage of storage is low, only dissimilar scenarios are stored, otherwise when the percentage increases, more similar scenarios are also stored. The choice of a suitable storing percentage depends on the architecture size and on the faults quantity. We consider this percentage as a selection criterion that is analyzed on Section VII.

VI. Setup of Experimental Results

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 [17]. In other words, it is a source of static and dynamic faults. Therefore, this work uses the variability model proposed by Hargreaves et al. [18] for generating the fault scenarios for the experimental results. 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. Aiming to explore scenarios with 65 nm and 22 nm manufacturing process, the link-delay variability where set to 5% (σ = 0.05) and 18% (σ = 0.18), respectively, as predicted by the ITRS roadmap [19]. Additionally, the experiments where produced with λ = 0.4 and λ = 1.2, representing high and low strength of the spatial correlation variability, respectively. These values represent typical correlation of the fabrication processes [18]. The experimental setup covers four NoC sizes (5×5, 6×6, 7×7 and 8×8). Each NoC size combined with link-delay and spatial correlation parameters produces 16 fault scenarios. Aiming to explore the randomness of the variability model, we generated three times each one of these scenarios, resulting 48 irregular mesh NoCs. For these experiments, we selected topologies with less than 7 link faults, resulting in 30 topologies. Figure 9 describes the composition of simulation scenarios, and the steps applied on the experimental evaluation.

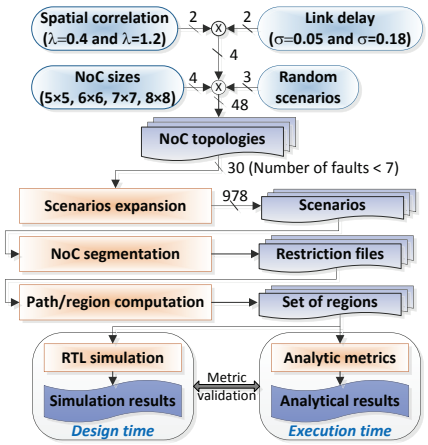


Figure 9 – Setup of experimental results.

An in-house tool expands these 30 topologies in 978 scenarios that combine all possibilities of faulty links. Thus, for each one of the 978 scenarios, the in-house tool performs the following two steps: (i) Segmentation of the network using a segment routing approach to generate a restriction file. This file contains all the forbidden directions, in order to 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.

In the event of a new fault, *OsPhoenix* selects, from the available set of scenarios, the one that minimizes the overall system latency. Aiming to choose an appropriate runtime metric, we employed RTL simulation with synthetic traffic to evaluate all scenarios. We compare simulation results with two analytical metrics: (i) Average Routing Distance (ARD), which is the sum of all path lengths (measured as the number of hops) divided by the number of paths (in a NoC with N nodes, there are N^2 paths [20]); and (ii) Link Weight (LW), which is the number of paths that cross each link, considering links direction. ARD produces the shortest path for all (or most) source-destination pairs, while LW reduces the overall latency through a sound traffic balancing. Although both metrics are suitable for scenarios selection, the ARD obtained better results. Thus, *OsPhoenix* implemented the ARD metric for the execution of all experiments.

VII. Experimental Results

This section displays experimental results obtained in the evaluation of the selection method for reduced set of scenarios, which takes into account a percentage of the most peer-to-peer dissimilarity, among all fault scenarios, using *LsHV* and *LsC* methods. The experiments evaluate the costs of employing new fault scenarios, once the preprocessed scenarios have already computed their routing tables. Figure 10 shows the flow of the experimental results.

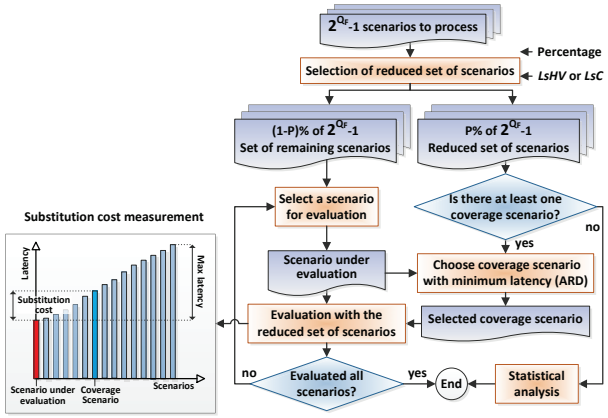


Figure 10 – Flow of substitution cost analysis.

For each scenario, which was not included into the reduced set of scenarios, we compute the substitution cost according to the average latency of all communications. When the reduced set of scenarios does not contain a scenario that covers the scenario under evaluation, the flow computes as an uncovered situation, otherwise *OsPhoenix* searches the one with minimum latency, considering this one as the selected coverage scenario. The difference of latency between the scenario under evaluation and the scenario selected by

OsPhoenix is the substitution cost evaluated here. Note that, at runtime, the *OsPhoenix* estimates the latency of a scenario applying the ARD metric.

As mentioned before, in a new situation where there is not a preprocessed scenario to cover a faulty link, the network needs to stop waiting for the processing of the new routing table that considers this fault, which is a very timing consuming task. The following experiment analyses the efficiency of *LsC* and *LsHV* dissimilarity methods in searching a suitable reduced set of coverage scenarios. The experiment takes into account the average substitution costs of all 30 basic mesh topologies. The *LsC* method did not presented uncovered scenarios, showing its efficiency in finding the best scenarios according to the coverage criterion. It happens because *LsC* selects most dissimilar scenarios, which provides large coverage possibilities. On the other hand, Figure 11 shows that the *LsHV* method produced a large quantity of uncovered scenarios (13 of 30), even selecting 50% of scenarios to compose the reduced set of scenarios.

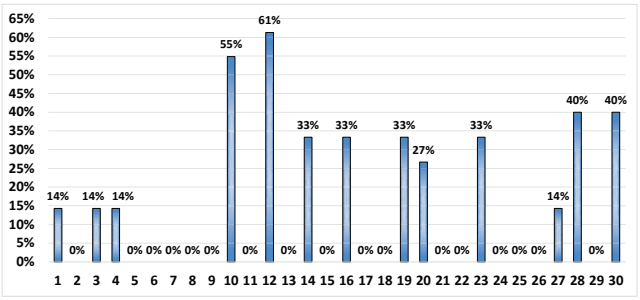


Figure 11 – Percentage of uncovered scenarios when applying the *LsHV* dissimilarity method, with 50% of selection percentage.

As shown in Figure 10, once a coverage scenario if found, its latency cost (substitution cost) may be compared against the best possible scenario. Figure 12 presents this cost taking into account two traffic injection rates (IR 5% and IR 15%) and the *LsHV* and *LsC* methods. It highlights that (i) both dissimilarity methods have low substitution cost for low injection rate, which is rewarded by the fast system recovery; (ii) although *LsHV* produces uncovered scenarios, these ones have the lowest average substitution cost; and (iii) near of the network saturation point, which was achieved with IR 15%, the substitution costs increases significantly. It happens because ARD metric does not give dynamic information.

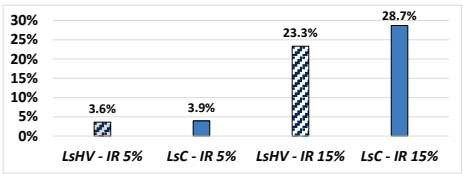


Figure 12 – Average substitution costs for 50% of reduction on fault scenarios set for *LsHV* and *LsC* methods and 2 injection rates (5%, 15%).

VIII. CONCLUSIONS

This paper proposes a hardware/software reconfiguration approach based on preprocessing fault scenarios. The software is a small part of the Operating System kernel called *OsPhoenix* that preprocess fault scenarios as soon as a fault prediction monitor place in each link of each router detects a fault tendency. The hardware part is a fault-tolerant mesh NoC, which employs region-based routing mechanism.

The preprocessing scenarios approach reduces the time that the network is halted waiting for the computation of a new routing algorithm to support a new set of faults. The quantity of scenarios grows exponentially with the quantity of faults, implying large area of memory and processing time to compute all set of scenarios. Aiming to minimize this problem, this work employs three strategies: (i) differential treatment for static and dynamic faults, (ii) incremental fault scenarios, and (iii) dissimilarity approach, which enable to find the most dissimilar scenarios based on the 2D cross-correlation method. The preprocessing of scenarios implies substitution costs with the coverage scenario. However, according to our simulations, the average substitution cost is less than 4% for low injection rates.

REFERENCES

[1] R. Marculescu et al. “Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives”. IEEE trans. comput.-aided des. integr. circuits syst., v. 28, n. 1, pp. 3-21, Jan. 2009.

[2] 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.

[3] A. Mejia et al. “Region-Based Routing: A Mechanism to Support Efficient Routing Algorithms in NoCs”. IEEE Trans. Very Large Scale Integr. (VLSI) Syst., v. 17, n. 3, pp. 356-369, Mar. 2009.

[4] A. DeOrio et al. “A Reliable Routing Architecture and Algorithm for NoCs”. IEEE trans. comput.-aided des. integr. circuits syst., v. 31, n. 5, May 2012.

[5] C. Marcon et al. “Phoenix NoC: distributed fault tolerant architecture”. International Conference on Computer Design, pp. 7-12, 2013.

[6] J. Domke, T. Hoefler, W. Nagel. “Deadlock-Free Oblivious Routing for Arbitrary Topologies” IPDPS, pp. 616-627, 2011.

[7] M. Ebrahimi et al. “Minimal-path fault-tolerant approach using connection-retaining structure in Networks-on-Chip”. IEEE/ACM International Symposium on Networks on Chip (NOCS), pp. 1-8, 2013.

[8] M. Palesi et al. “A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architectures”. SAMOS, pp. 373-384, 2006.

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

[10] 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.

[11] R. Holsmark et al. “Deadlock-Free Routing Algorithms for Irregular Mesh Topology NoC Systems with Rectangular Regions”. Journal of Systems Architecture, v. 54, n. 3-4, pp. 427-440, Mar. 2008.

[12] D. Fick et al. “Vicis: A Reliable Network for Unreliable Silicon”. Design Automation Conference (DAC), pp. 812-817, 2009.

[13] C. Feng et al. “Addressing Transient and Permanent Faults in NoC with Efficient Fault-Tolerant Deflection Router”. IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, n. 6, p. 1053-1066, Jun. 2013.

[14] A. Strano et al. “OSR-Lite: Fast and Deadlock-free NoC Reconfiguration Framework”. Conference on Embedded Computer Systems, pp. 86-95, 2012.

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

[16] B. Pan et al. “Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review”. Measurement Science and Technology, vol. 20, n. 6, pp. 1-17, Apr. 2009.

[17] M. Shintani et al. “A Variability-Aware Adaptive Test Flow for Test Quality Improvement”. IEEE trans. comput.-aided des. integr. circuits syst., v. 33, n. 7, pp. 1056-1066, Jul. 2014.

[18] B. Hargreaves et al. “Within-die Process Variations: How accurately can they be statistically modeled?” ASP-DAC, pp. 524-530, 2008.

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

[20] J. Flich et al. “A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms”. IEEE Transactions on Parallel and Distributed Systems, v. 23, n. 3, pp. 405-425, Mar. 2012.