Microprocessors and Microsystems 40 (2016) 137–153

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Scenario preprocessing approach for the reconfiguration of fault-tolerant NoC-based MPSoCs

Jarbas Silveira [a,*], César Marcon [b], Paulo Cortez [a], Giovanni Barroso [c], João M. Ferreira [a], Rafael Mota [a]

[a] LESC-DETI, Federal University of Ceará, Fortaleza 90455-970, Brazil
[b] PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre 90619-900, Brazil
[c] Department of Physics, Federal University of Ceará, Fortaleza 90455-970, Brazil

ARTICLE INFO

Keywords:
NoC
Irregular topology
Fault-tolerance
Preprocessing
Routing methods

ABSTRACT

The latest technologies of integrated circuit manufacturing allow billions of transistors to be arranged on a single chip, enabling the chip to implement a complex parallel system, which requires a communications architecture that has high scalability and a high degree of parallelism, such as a Network-on-Chip (NoC). These technologies are very close to the physical limitations, which increases the faults in manufacturing and at runtime. Therefore, it is essential to provide a method for fault recovery that would enable the NoC to operate in the presence of faults and still ensure deadlock-free routing. The preprocessing of the most probable fault scenarios enables us to anticipate the calculation of deadlock-free routings, reducing the time that is necessary to interrupt the system during a fault occurrence. This work proposes a technique that employs the preprocessing of fault scenarios based on forecasting fault tendencies, which is performed with a fault threshold circuit operating in accordance with high-level software. We propose methods for dissimilarity analysis of scenarios based on cross-correlation measurements of link fault matrices. Experimental results employing RTL simulation with synthetic traffic prove the quality of the analytic metrics that are used to select the preprocessed scenarios. Furthermore, the experiments show the efficacy and efficiency of the proposed dissimilarity methods, quantifying the latency penalization when using the coverage scenarios approach.

1. Introduction

The evolution of Very-Large-Scale Integration (VLSI) semiconductor technology enables us to integrate hundreds or even thousands of cores into a single circuit. This massive integration allows us to implement the entire functionality of a system into a single chip, generating a System-on-Chip (SoC). The International Technology Roadmap for Semiconductors (ITRS) foresees thousands of Processing Elements (PEs) integrated into an SoC by 2020 [1]. The Network-on-Chip (NoC) technology will play a key role in the implementation of these highly integrated SoCs. The two-dimensional (2D) mesh is the most popular NoC topology; it offers a simple and regular structure, and the small wire length is suitable for the tile-based design [2]. Under the NoC communication paradigm, each PE that is placed in a tile is interconnected throughout with routers and links to other PEs, and the communication is normally performed by packet transmission [3].

Recent submicron technologies provide more process variability, increasing the quantity of defective components [4,5]. A defective router or link could ruin the 2D mesh communication structure, which would lead to an irregular topology (i.e., topologies derived from regular networks with induced faults or faulty links, also known as agnostic topologies) [6]. As a consequence, static and deterministic routing algorithms that are tailored to a regular NoC topology will not operate properly, thus rendering the chip useless [2]. Aiming to avoid this problem, the design of the routing algorithms must provide some degree of fault-tolerance while ensuring deadlock-freeness. There are two approaches to achieve deadlock-free routing algorithms for irregular topologies: (i) one approach is based on virtual channels (e.g., [7–12]), which requires a large extra area for multiplexing schemes and implies significant energy consumption by the buffers of the routers; and (ii) another approach is based on turn prohibition (e.g., [13–17]), which avoids deadlock by eliminating a subset of network turns. Our work employs this last approach using tables in each router that implement the routing algorithm. However, the area consumption and power dissipation make table-based techniques prohibitive for large networks. Aiming to overcome

* Corresponding author.
E-mail addresses: jarbas@lesc.ufc.br (J. Silveira), cesar.marcon@pucrs.br (C. Marcon), cortez@deti.ufc.br (P. Cortez).

http://dx.doi.org/10.1016/j.micpro.2015.08.005
0141-9331/

these problems, several studies (e.g., [6,18,19]) have proposed to compress the routing table. Our work employs a technique that is similar to [6], which compresses the routing table according to the NoC regions.

Efficient implementations of routing strategies address the dynamicity of the faults with fast computation and reconfiguration of the fault-tolerant topology without requiring packet retransmission. This fast computation and reconfiguration could combine some static and/or dynamic tasks. For example, all of the possible fault scenarios could be computed before the system operation, which characterizes a static computation in opposition to the approach that computes new scenarios only during the system operation. Independent of the computation dynamicity, the reconfiguration could occur statically or dynamically. In the dynamic network reconfiguration, the entire system remains operating and the network traffic is not stopped, which implies that the router must know all of the communication possibilities to avoid deadlock conditions. On the other hand, in the static approach, the network enters into a reconfiguration phase, where it is halted and all packets are drained.

If each PE (connected to a router) preprocesses and stores scenarios, the system reconfiguration could be improved, which would reduce the time that the network is halted waiting for the routing algorithm computation that handles the topology changes. Unfortunately, preprocessing all of the possible fault scenarios is a very complex problem, which consumes time and memory. Aiming to minimize this problem, a system could statically elect and store only some scenarios to be used in an appropriate situation, such as a virtualization perspective or low power scenario. Additionally, in a given set of fault scenarios, there are some scenarios that can be discarded because they are covered by others, which enables us to minimize the memory area that is required to store the preprocessed scenarios.

This work employs Phoenix [20] as the fault-tolerant target system, which consists of a 2D mesh NoC and a layer for an operating system. Using Phoenix, this paper proposes an efficient approach to addressing dynamic faults on NoC links. This approach consists of preprocessing fault scenarios to quickly reconfigure the network with new deadlock-free routing, in the case of fault detection. Additionally, this work proposes a method for reducing a large set of scenarios based on cross-correlation to identify dissimilarities in sets of irregular topologies. This method consists of fault-monitors that are placed in each input port of the Phoenix NoC routers. Each fault-monitor evaluates whether the link is faulty or if there is a fault tendency. This information is transmitted to a PE that is locally connected to the router, and a communications driver placed inside the operating system preprocesses new scenarios based on this fault information.

This paper is organized as follows. Section 2 presents related work on routing mechanisms and reconfiguration processing. Section 3 describes the Phoenix architecture, which is the target architecture employed here. Section 4 describes the processing flow of fault-tolerant scenarios. In Section 5, we present the fundamentals of the fault scenario preprocessing. Section 6 presents some synthesis results of Phoenix for a specific MPSoC architecture. Section 7 describes the methodology and experimental results of the exploration of analytic metrics for runtime latency estimation, while Section 8 analyzes the methods and costs for the preprocessing approach. Finally, Section 9 presents the main conclusions of this work.

2. Related work and main contributions

Strategies applied to fault-tolerant routing for irregular NoCs can employ static or dynamic fault models. In the first strategy, all of the tolerable faults must be known at the design time. Then, during the system operation, when a fault is discovered, the system is halted, the packets are dropped, and a new deadlock-free routing is employed. The second strategy does not need to halt the network, and the routing computation is performed at runtime. Moreover, the second strategy requires complex routing distributed mechanisms to avoid inconsistences (e.g., in routing tables) during the network reconfiguration. The strategies that use static fault models (e.g., [21,22]) are less complex and less area consuming, but they can tolerate only a limited quantity of faults [23]. On the other hand, the routing mechanisms of strategies that use dynamic fault models (e.g., [23–25]) are more complex, but they allow many more changes to the network topology. This work focuses on this last approach.

A reconfigurable fault-tolerant system for irregular networks requires mechanisms for (i) fault detection and diagnosis; (ii) fault recognition reporting; (iii) deadlock-free routing computation; and (iv) ***routing reconfiguration*** (e.g., routing tables and auxiliary circuits). This section discusses studies that are related to the last mechanism, which is the main focus of this work, accounting for only network architectures *without virtual channels*. Additionally, the routing reconfiguration is composed of the ***routing mechanism*** and ***reconfiguration processing*** ; both are discussed next.

The ***routing mechanism*** defines how the routing information is computed and stored. To support reconfiguration, the routing mechanism needs resources that could be changed during the runtime, which normally encompasses tables placed in each NoC router, storing the routing information. Routing tables support a large quantity of topologies and are easy to implement. On the other hand, they do not scale with an increase in the NoC size. Aiming to reach the scalability that is required for current and future highly populated NoCs, several studies employ techniques to compress or minimize the sizes of the routing tables, which is a complex task and could imply a loss of performance and/or an impossibility of reaching all of the target nodes. Examples of these studies are (i) Palesi et al. [18], which uses a table compression technique for application-specific routing; and (ii) Bolotin et al. [19], which uses a table minimization technique that applies a fixed function combined with minimal deviation tables.

Scalability is also achieved by dividing the network into regions and mapping those regions onto routing table entries. For example, Mejia et al. [6] proposed the Region Based Routing (RBR) approach, where each node contains a set of regions that are based on paths that cover all of the communications. RBR could be employed with a specific routing application, enabling the use of a small number of regions and assuring full network coverage. Fukushima, Fukushi and Yairi [26] propose another region-based approach that is based on a set of rectangular faulty regions and the corresponding deviation paths, which are employed to avoid the faulty regions. Their approach improves the work of Holsmark et al. [27], providing a complete and deadlock-free routing that reduces the faulty regions' sizes. Consequently, it reduces the number of nodes to be deactivated and the routing implementation complexity. Although the region-based approaches scale with the NoC size, it is necessary to have a large number of regions to maintain full NoC coverage with efficient paths, which prohibits a large routing table reduction. Aiming to overcome this problem, Rodrigo et al. [2] improved the previous LBDR work [28], proposing the universal Logic-Based Distributed Routing (uLBDR), which provides an efficient routing mechanism using a small set of configuration bits together with logic cells instead of large routing tables implemented in memories. The main limitation of their work is that to cover all of the deadlock situations, the approach must use virtual cut-through switching, which limits the packet size to the length of the buffers.

The ***reconfiguration processing*** defines the computation cost of making dynamic routing decisions, which could be implemented in hardware, in software or in both, depending on the application requirements.

Fick et al. [24] 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. Vicis employs routing reconfiguration at the router and network levels. The two-level reconfiguration allows it to confine some in-router faults and the corresponding action at the router level, thus simplifying the process of reconfiguring and allowing the network performance to degrade gracefully while increasing the quantity of network faults. Feng et al. [29] propose a fault-tolerant solution for a bufferless NoC that includes 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 to tolerate permanent faults without deadlock and livelock situations and without loss of packets during the reconfiguration process. In addition, their work presents a hierarchical FTDR (FTDR-H) algorithm, to reduce the area overhead of the FTDR router. The experimental results show that FTDR and FTDR-H are implemented with reliable bufferless routers, which can protect against any fault distribution pattern. Although there is variation in the studies of Fick et al. [24] and Feng et al. [29], both are conceptually similar, suggesting that a hierarchical implementation is a sound approach to optimizing the reconfiguration process.

Strano et al. [23] propose the Overlapped Static Reconfiguration (OSR), which is an old technique that was used in off-chip networks; however, it is now adapted to the NoC restrictions. Their approach allows new packets to be injected into the network while old packets are routed or dropped. The authors use token signals to separate the old packets from new ones; and during the token signal propagation, the NoC retains both configurations, the previous and the new one. This approach enables fast recovery, providing little time to reload and have the NoC working properly, but a complex hardware implementation is implied.

Triviño et al. [25] use a virtual-regions strategy to improve the performances of the applications that are simultaneously running on a Chip MultiProcessor (CMP), which splits the CMP into several regions. Their approach dynamically reconfigures the network partitions, generating irregular NoC topologies that isolate the traffic of the applications and consequently reduce or even eliminate the interferences among inter-application messages.

Our work employs a dynamic fault model that encompasses three phases: (i) fault detection and fault propagation; (ii) deadlock-free routing computation; and (iii) routing reconfiguration. The main contribution of our work is in the routing reconfiguration phase that provides fast deadlock-free routing reconfigurations for irregular NoC topologies. Our approach is based on preprocessing the most likely fault scenarios, which are computed according to the detection of the link fault tendency. Our approach, with the information on the fault tendencies, enables us to employ more complex and time-consuming algorithms that are aimed at producing optimal solutions for large NoCs without compromising the execution time, once the routing tables were already preprocessed. Moreover, in a given set of scenarios, some scenarios can cover others, which allows us to diminish a large set of preprocessed scenarios. This approach provides two new and important 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 measures to identify dissimilarities in sets of irregular topologies, which minimizes the storage area of the preprocessed scenarios.

3. Phoenix architecture

Fig. 1 shows the Phoenix distributed fault-tolerant architecture [20] over an NoC-based MPSoC platform that consists of 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 of a paired processor-memory. Additionally, each PE is connected through an NoC interface to the local port of each router. Each field of a Phoenix packet has a 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 PE messages; and (ii) the control packet, which controls the fault-tolerant mechanisms. The *OsPhoenix* communicates with the *HwPhoenix* via bidirectional control packets that are transmitted through the local port of each PE. Examples of control packets are (i) test_links, which is employed when the *OsPhoenix* needs to test all of the links of the local router; (ii) tr_rout_tab, rd_rout_tab and wr_rout_tab, which are used to transmit, read and write the *Routing Table* of each router (refer to Section 5.1), respectively; and (iii) tr_fault_tab, rd_fault_tab and wr_fault_tab, which are analogous to the previous commands but account for the *Fault Table* of each router.

3.1. 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 that makes the fault-tolerant mechanism of the NoC transparent to the system operation. Fig. 2 depicts the main modules of *OsPhoenix* and their interactions.

The *OsPhoenix*'s *Kernel* consists of (i) the *Control Module*, which manages the fault-tolerant mechanism; and (ii) the *NoC Driver*, which contains the routines to convert logical addresses of the application message to physical addresses of the network, and vice versa. Additionally, this driver makes the fault-tolerant mechanism to the PE's operating system transparent because control packets are exchanged between the *Control Module* and NoC Interface without the knowledge of the PE's operating system.

The *Global Fault Table* stores the status (i.e., whether a link is tested, faulty, with fault tendency or operating properly) for all of the NoC links. This table is a global copy of all of the router *Fault Tables* (refer to Section 5.1). The *Control Module* writes/reads this table to synchronize the fault link knowledge of the *OsPhoenix* of all of the PEs.

The *Scenarios Processing Module*, which is the key aspect of this work, computes routing tables according to the fault or tendency of fault on links when commanded by the *Control Module*. It uses the fault links information that is provided by the *Global Fault Table* together with new fault information to search a previous

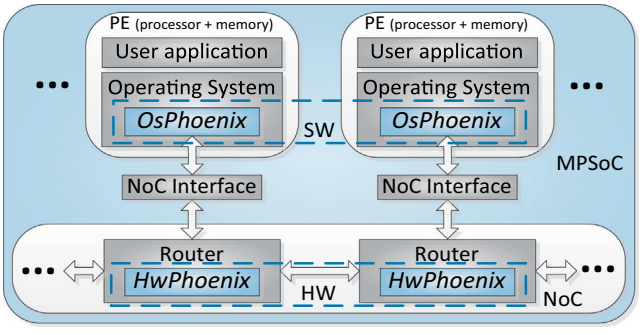


Fig. 1. Phoenix distributed architecture [20].

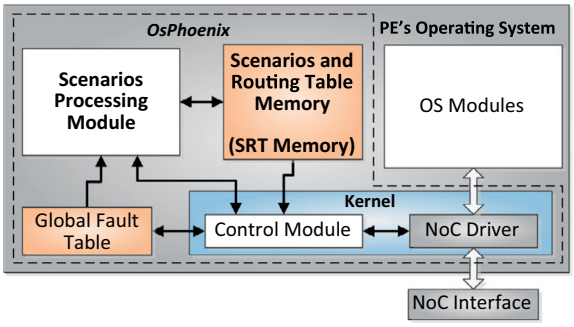


Fig. 2. Block diagram of the *OsPhoenix* architecture. The dashed line encloses *OsPhoenix* and contains five main elements (i.e., the memory, tables and processing modules), and the black arrows show how the elements are interrelated.

computed scenario that covers this new fault situation, in the *Scenarios and Routing Table Memory* (*SRT Memory*). If a candidate scenario is found, then the associated *Routing Table* contains a new routing and can be transmitted to the *HwPhoenix*. Otherwise, this module preprocesses a new fault-tolerant scenario and its associated *RBR Table*.

4. Scenario processing flow

The Phoenix NoC starts with all of the local routing tables configured to operate with the XY routing algorithm. As soon as *OsPhoenix* is loaded, the preliminary link tests are commanded. If all of the links are working properly, the network starts operating. In the case of detecting the faults, the *OsPhoenix* and *HwPhoenix* perform several steps on all of the PEs and routers, to establish a new routing algorithm [20] that allows deadlock-free communication. Aiming to capture dynamic faults during the network operation, the FPM modules search for permanent faults and fault tendencies in all of the NoC links. The information on these faults is transmitted to *OsPhoenix* for appropriate handling. Fig. 3 illustrates a Message Sequence Chart (MSC) that contains the steps that are taken after an FPM module detects a fault on a 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*, which propagates this event to the *OsPhoenix* through a control packet that contains the *Fault Table*.

Inside the *OsPhoenix* kernel, the *NoC Driver* module captures the control packet and transmits it to the *Control Module*, which checks if the fault is already annotated in the *Global Fault Table*. In the case of positive checking, the *Control Module* considers that the problem is already known and treated and concludes the fault processing, i.e., no other message is generated. Otherwise, the *Control Module* updates the *Global Fault Table*, and through a control packet, requests to the *Fault Control Machine* the broadcast propagation (i.e., to all neighboring routers) of the control packets that contain the fault information. Simultaneously, the *Control Module* commands the *Scenarios Processing Module* to proceed with the next fault-tolerant steps (e.g., to process a new fault coverage scenario).

The control packets are propagated among neighboring routers until all of the *OsPhoenix* are notified. Any neighboring router that receives a control packet with the fault information propagates the same information to its *OsPhoenix*, following the same steps described above. The control packet propagation generates extra communication traffic. However, because the control packet is normally short (i.e., a 5-flit length in our implementation) and it is transmitted only in the presence of new fault detection, they do not impact significantly on the overall data communication. Additionally, each *OsPhoenix* contains a timing mechanism to define a maximum time for network stabilization, which is reached when all of the *OsPhoenix* receive the same fault information. This mechanism is used when the network is partitioned by a sequence of faulty links that precludes transmitting control packets to all of the routers [20].

Fig. 4 illustrates the fault-tolerant 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 whether the fault is already covered by some scenario that was previously computed. In case of a positive answer, no action is taken. Otherwise, aiming to enable a fast

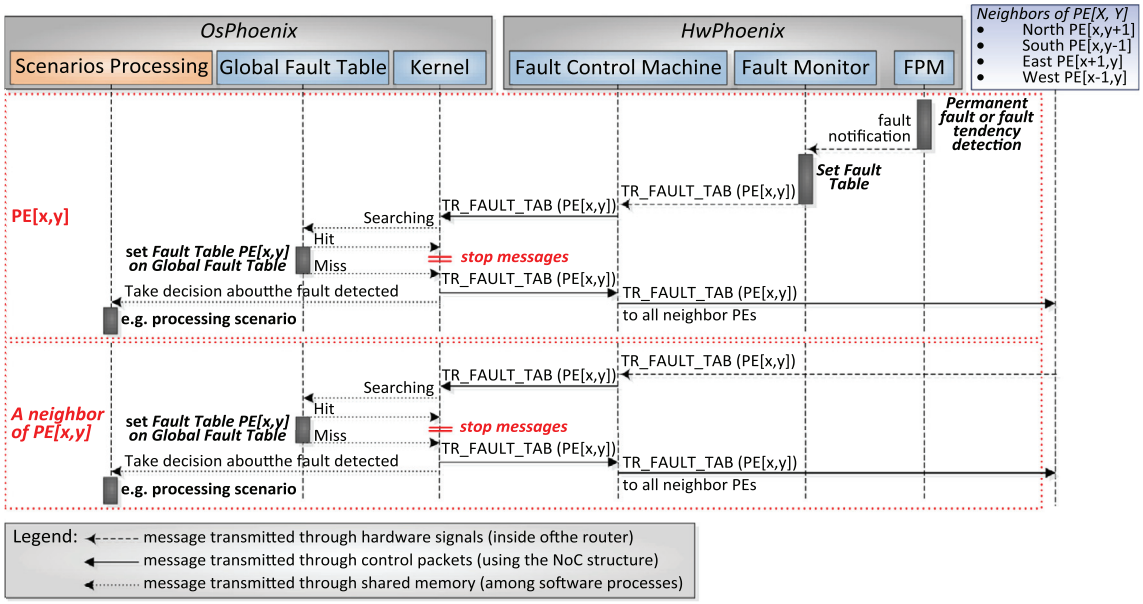


Fig. 3. MSC of the fault processing mechanism when a faulty link or a fault tendency is detected.

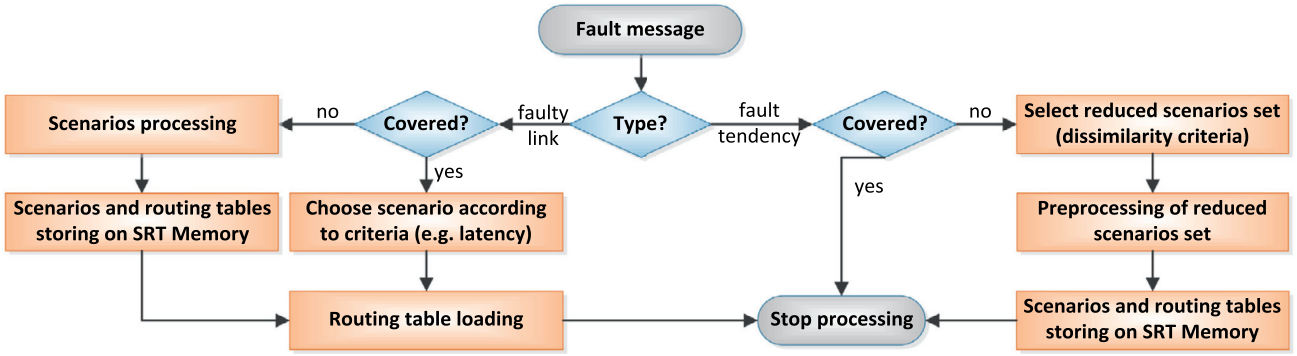


Fig. 4. *Scenarios Processing Module* operation according to the fault type.

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 number of fault scenarios rises exponentially with the quantity of faulty links. To address this complexity, this paper proposes to preprocess a limited set of scenarios based on a dissimilarity method that uses the cross-correlation of fault matrices to better meet the applications requirements. The preprocessing approach can be employed to fulfill several applications requirements (e.g., to reduce power dissipation and to achieve homogeneous thermal distribution). Nevertheless, this work uses latency minimization as an application requirement and employs the Average Routing Distance (ARD) as a metric for fast latency estimation. The dissimilarity method and ARD metric are explained next in Section 5.

When a message of a faulty link is received, the decision about network recovery is preeminent because the FPM module can no longer recover defective packets, which prevents network communication. The *Scenarios Processing Module* checks whether the fault is covered by the scenarios that are stored in *SRT Memory*. If the fault is covered, then *OsPhoenix* can perform the fast routing reconfiguration. The *Scenarios Processing Module* searches the one that better fulfills the application requirement and sends a control packet to the *Control Module* that contains 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, it configures the routing table, as described above.

The Phoenix platform accounts for the premise that "*all PEs have the same algorithm to generate scenarios and routing paths*". This premise allows for each *OsPhoenix* to have its own *Global Fault Table* and its own *SRT Memory*, and all of the *OsPhoenix's* (i.e., one per PE) operate independently and in a distributed way. This arrangement also avoids having it be necessary for routing tables to be propagated through the NoC because each router has a *Routing Table* that is set by its local *OsPhoenix*.

5. Fundamentals of the fault scenarios preprocessing

The high variability of the most recent technologies applied to the manufacture of CMOS circuits makes these circuits susceptible to transitory changes, for example, voltage fluctuations and temperature variations, which can be perceived by the monitoring system as a dynamic operation fault. In the context of a system that has many and frequent dynamic faults, a preprocessed scenarios approach is important to meet the requirements of the performance, latency and throughput. Additionally, the greater the number of processed scenarios, the greater is the fault coverage. Unfortunately, preprocessing a large number of scenarios is a very complex problem, which consume both time and memory. This section describes the fundamentals of the scenario preprocessing and how the number of scenarios can be reduced without decreasing the quality of the fault-tolerant solution.

5.1. HwPhoenix fundamentals

Phoenix NoC employs a direct 2D mesh topology with m lines and n columns, which consists of $m \times n$ routers with bidirectional links to connect with the other routers and PEs. The NoC employs routing tables for the source routing decisions, and the *OsPhoenix* performs routing algorithms to fill the routing table according to the positions of the PEs and the faulty links. Furthermore, the Phoenix NoC implements wormhole switching, which divides the packets into flits (the flit size of the Phoenix is equal to the phit size), demanding only small buffers for data storing. Additionally, the Phoenix NoC uses credit-based flow control to reduce the number of clocks that are required for the flit transmissions.

Fig. 5 shows the Phoenix router architecture, which includes mechanisms for packet routing and fault-tolerance. The basic packet routing mechanism of Phoenix router encompasses four set of components, which are described next: (i) Four bidirectional ports (north, south, east and west) dedicated to interconnect routers and a bidirectional port (local) that enables communication between the router and its local PE. All of the input links contain configurable buffers that are 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 packet routing and arbitration according to the packet header and *Routing Table* content. The arbitration follows a dynamic rotating policy to ensure that all of the incoming requests are processed, which avoids the starvation phenomenon.

Phoenix NoC employs distributed routing in which routes are computed according to the *Routing Table*, which is initialized with XY routing. Nonetheless, depending on the occurrence of faults, a new deadlock-free routing is provided by *OsPhoenix* that modifies the *Routing Table*. The NoC routing algorithm is similar to RBR [6], which groups target addresses into regions 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. If the *Routing Table* size increases, then *OsPhoenix* provides an alternative for the minimum path using, e.g., heuristic algorithms.

The fault-tolerance mechanism implemented in each router of the *HwPhoenix* includes three types of circuits: (i) a fault detection and correction module that contains a Hamming Encoder (HE), Hamming Decoder (HD) and *Fault Prediction Module* (FPM), which is placed in each one of the bidirectional links that interconnects the routers; (ii) a *Fault Monitor* that communicates with the FPM

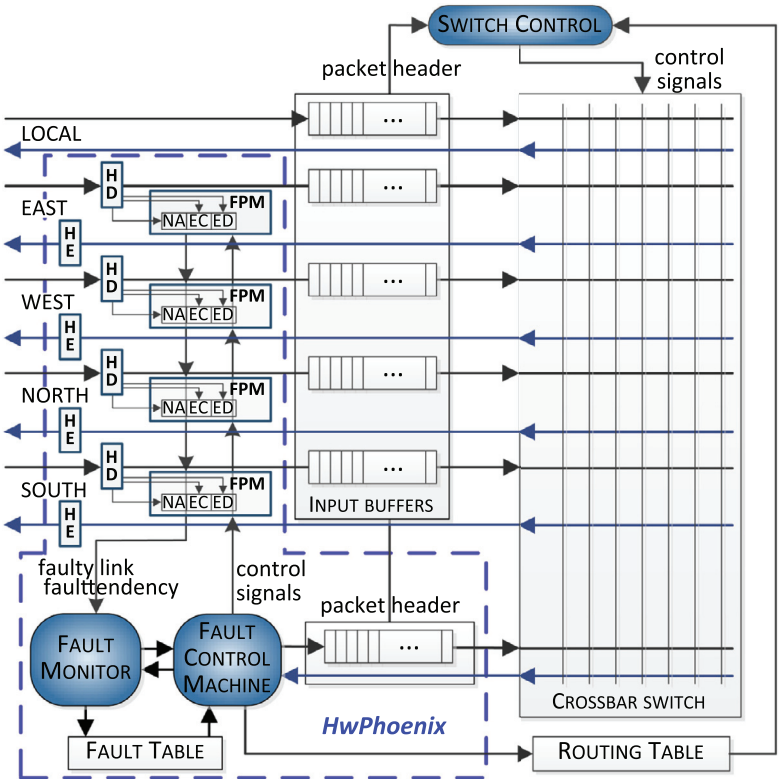


Fig. 5. The basic components of Phoenix router architecture. The main components of *HwPhoenix* are bounded by dashed lines.

to set the status of the links on the *Fault Table* according to a two-level fault model; and (iii) a *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.

Fig. 6 depicts the two-level fault model that is implemented in each Phoenix router. The first level is a 4-field vector in which each field stores the operation status of the north, south, east and west links, which contains 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 complements the first level, providing extra information about the quality of the link. This second level, which is physically placed inside each FPM, has 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 Fig. 6, the counters have different lengths to implement a window of events to capture the probabilities of NE, CE and DE.

The Phoenix implements two mechanisms for testing the link qualities: a static and a dynamic mechanism, which are independent but complementary. The static link test starts with *OsPhoenix* sending, through the router local port, the test_links control packet, to the *HwPhoenix*. The *Fault Control Machine* interprets this command as broadcasting a predefined test packet to all of the output ports except for the local output port. 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 *Fault Control Machine*, which sends the tr_fault_tab control packet containing the *Fault Table* to the *OsPhoenix* [20], with a higher priority assigned to minimize the control packet latency. The static mechanism sets only the Level 1 of the fault model with a “faulty” or “operating properly” status. Typically, the *OsPhoenix* produces a test_links control packet when the system is started or asynchronously by a high-level command (i.e., provided by the application layer – this procedure is not discussed here). Note that although the dynamic mechanism does not interrupt the network operation, the static mechanism must stop the router communication when not all of the links are verified.

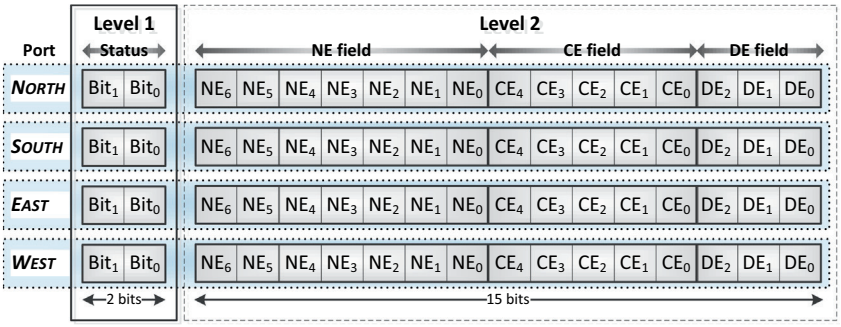


Fig. 6. The two-level fault model (i.e., Level 1 – Fault Table; Level 2 – Link status counter).

To implement the dynamic link test, each bidirectional link contains an HE and HD to perform a strategy that is similar to the strategy used in [30], which identifies fault tendencies using circuits based on a threshold. The HD, which is placed on the input of each buffer, 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, and thus, the module informs the communication status by the signals NE, EC and ED. The HE module generates the redundancy bits according to the flit that is transmitted. Additionally, associated with each data link between routers, there are control signals to request the retransmission of faulty packets. However, this mechanism is omitted here because this aspect is not the focus of the present paper.

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, which is placed inside the *Fault Monitor*, measures the density of the errors that are corrected and deduces a link fault tendency. This fault tendency information is propagated to the *OsPhoenix*, which makes inferences to permanent errors or tendencies of errors. According to these inferences, *OsPhoenix* can set, on Level 1 of the *Fault Table*, the bidirectional link to be faulty (e.g., a permanent error) and/or could start the preprocessing of a new routing scenario that avoids the use of a link with fault tendency. When a link is marked as faulty, the HE and HD modules are turned off and remain in this status until the *OsPhoenix* requires a new link evaluation.

5.2. On-demand processing versus preprocessing approach

One of two approaches, illustrated in Fig. 7, is employed to reconfigure a NoC from a fault occurrence: on-demand processing or preprocessing of fault scenarios. Three intervals of time compose the on-demand processing: (i) *scenario computation* (ΔSc), which consists of the calculation of the routing tables when new faults are detected; (ii) *NoC reconfiguration* (ΔNr_1), which includes stopping the network operation, a possible discard of packets, reconfiguring the new routing tables, and resynchronizing the network for the next operation period; and (iii) *system operation* (Δt_1), where the network provides new communication paths for data traffic.

Let ΔRi be a time interval between two *Reconfiguration Commands* (RCs), which depends on a random fault event. Let ΔNr_1 be the time interval for the NoC reconfiguration, which is nearly constant for each NoC size. Let $\Delta Ri = \Delta Sc + \Delta Nr_1 + \Delta t_1$ be the equation that defines the composition of ΔRi for the on-demand processing approach. Then, reducing the routing algorithm interval for new scenario computation (i.e., ΔSc) increases the time remaining for the system operation (i.e., Δt_1). However, reducing ΔSc reduces the efficiency of the routing algorithm, thereby increasing the communication latencies. Thus, the on-demand processing approach implies a tradeoff between ΔSc and Δt_1.

Four intervals of time compose a reconfiguration approach that employs the preprocessing of fault scenarios: (i) *scenarios preprocessing* (ΔSp), which implies the routing table computation for each fault scenario; (ii) *scenario selection* (ΔSs), which implies selecting the coverage scenario that provides the most efficient communication of all of the stored scenarios; (iii) *NoC reconfiguration* (ΔNr_2); and (iv) *system operation* (Δt_2). These two last intervals of time have the same meaning as those described in the on-demand processing approach. The ΔSp can be performed throughout all of the system operation, i.e., ΔSp can be less than or equal to ΔRi, and no extra processing time is required in the occurrence of an RC. Thus, the routing algorithm has much time to search for efficient routing paths. The ΔSs is not time-consuming because the scenarios can be stored and ordered to facilitate the scenario selection. However, this stage is important because a bad scenario selection can compromise the system performance. Aiming to evaluate the quality of this task, we adopt the concept of *coverage penalization*, which is the average latency measured by the difference between the latency achieved with an optimum solution (e.g., a scenario whose routing tables produce the minimum average latency) and the latency achieved by the selected coverage scenario. Section 8 describes the experimental results on the *coverage penalization*.

Both approaches are performed during the same ΔRi, such that $\Delta Sc + \Delta Nr_1 + \Delta t_1 = \Delta Ss + \Delta Nr_2 + \Delta t_2$. Considering that the *NoC reconfiguration* is independent of the reconfiguration approach, $\Delta Nr_1 = \Delta Nr_2$. In addition, to achieve efficient routing, the on-demand processing approach requires $\Delta Sc \gg \Delta Ss$, and then, $\Delta t_2 > \Delta t_1$. Consequently, the preprocessing approach also increases the time that is available for the system operation, which emphasizes another improvement of the approach adopted here.

5.3. Definition of coverage scenarios

In a given set of scenarios, some of the scenarios cover others, which enables a reduction in the number 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 (i.e., the covered scenario) when all of the communication allowed in s_a is allowed in s_b, but the opposite is not necessarily true. Consequently, a coverage scenario is equally restrictive or more restrictive than the covered scenario, and then a coverage scenario can be used instead of the covered scenario.

Fig. 8(a) exemplifies this situation, where a 5×5 mesh NoC has four links (l_1, l_2, l_3 and l_4) that have the tendency to fail. To cover all of the fault situations would be necessary to preprocess 16 scenarios. Fig. 8(b) shows the best coverage scenario for the faults on links l_1 and l_3. However, the scenarios presented in Fig. 8(c and d) can cover this same fault combination. We remark that these two coverage scenarios contain an extra faulty link (i.e., l_4 or l_2). Therefore, the preprocessing mechanism must choose which one of these scenarios provides better performance to the system operation. When a given scenario covers others, the set of routing tables applied to the coverage scenario can be employed to the routers of the covered scenario. Thus, in the fault occurrence situation, 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*, which was already computed.

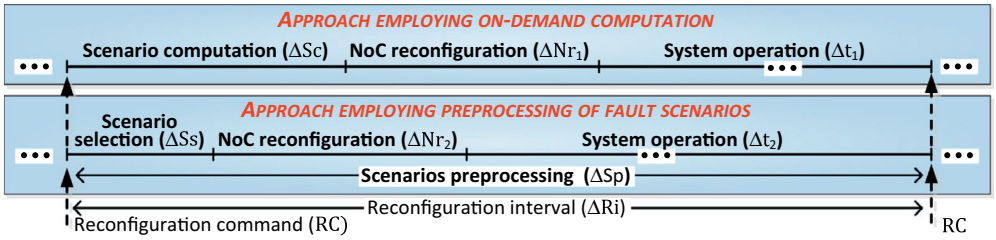


Fig. 7. Stages for a fault-tolerant NoC that employs on-demand processing or preprocessing approaches.

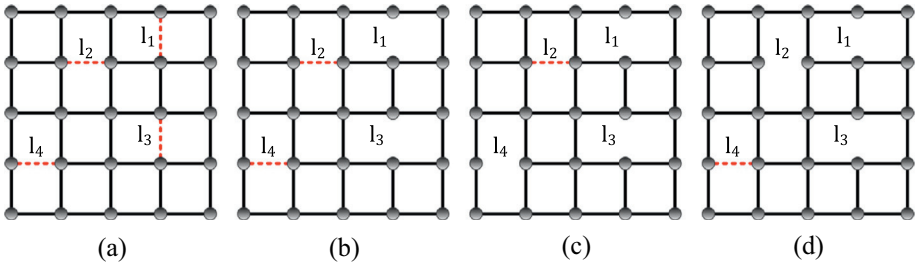


Fig. 8. A 5 × 5 mesh NoC with four links with fault tendency (i.e., dashed lines) and some coverage scenarios.

5.4. Level of similarity with a cross-correlation method

The selection method should search for coverage scenarios that do not degrade the communication, i.e., those that keep the latency of the packets as low as possible. Aiming to attain this goal, this work uses two approaches that are based on the 2D cross-correlation method [31] and that enable searching for sound coverage scenarios based on dissimilarity in the fault scenarios.

The 2D cross-correlation of an $M \times N$ matrix A and a $P \times Q$ matrix B is a matrix X of size $M + P - 1$ by $N + Q - 1$, such that $X = A * B$. Eq. (1) computes each element of X, which is a weighted sum of neighboring elements.

$$X(k,q) = \sum_{m=1}^{M}\sum_{n=1}^{N} A(m,n) \times B(m-k, n-q)$$
$$\forall \; -1P+1 \le k \le M-1, \; -Q+1 \le q \le N-1 \qquad (1)$$

Let M and N be the lines and columns of a 2D mesh NoC, respectively. We define the following matrices that are depicted in Fig. 9 (a and b): (i) R represents the routers; (ii) H and V represent the horizontal and vertical links, respectively; and (iii) C represents a composition of links.

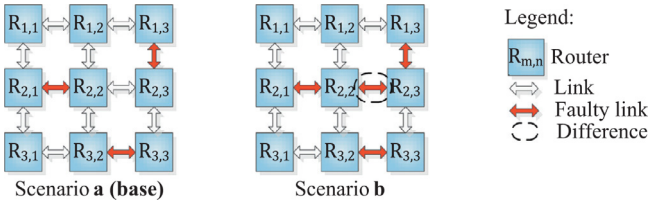
$R_{m,n}$ is a router, and $C_{m,n}$ is a composition of links, and both have coordinates m and n, $\forall\ 1 \leqslant m \leqslant M$ and $1 \leqslant n \leqslant N$. In addition, $H_{m,n}$ is a horizontal connection between $R_{m,n}$ and $R_{m,n} - 1\ \forall\ 1 \leqslant m \leqslant M$ and $1 \leqslant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

This work uses 2D cross-correlation to compare scenarios to find dissimilarities. We define LsHV (Level of similarity on Vertical and Horizontal links) as a method that employs cross-correlation on the matrices H and V, and LsC (Level of similarity on links Composition) as a method that employs cross-correlation while accounting for the joint effect of all of the 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 the 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. Eqs. (2) and (3) illustrate LsHV and LsC, respectively, for the two synthetic scenarios a and b.

$$\mathrm{LsH_{ab}} = \frac{||H_a * H_b||}{||H_a * H_a||}, \qquad \mathrm{LsV_{ab}} = \frac{||V_a * V_b||}{||V_a * V_a||},$$
$$\mathrm{LsHV_{ab}} = \frac{\mathrm{LsH_{ab}} + \mathrm{LsV_{ab}}}{2} \qquad (2)$$

$$\mathrm{LsC_{ab}} = \frac{||C_a * C_b||}{||C_a * C_a||} \qquad (3)$$

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



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

$$C_a * C_b = \begin{bmatrix} 4 & 12 & 17 & 12 & 4 \\ 12 & 34 & 48 & 34 & 12 \\ 17 & 48 & 68 & 48 & 17 \\ 12 & 34 & 48 & 34 & 12 \\ 4 & 12 & 17 & 12 & 4 \end{bmatrix}$$

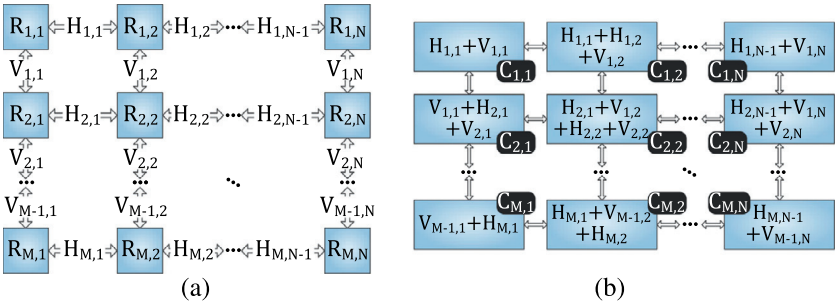


Fig. 9. Matrices employed in the cross-correlation method: (a) illustrates matrices R, H and V, which are represented in a NoC fashion (routers are rectangles, and links are double arrows); (b) describes matrix C, where each rectangle contains the sum of all of the links that are directly connected.

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

5.5. Reduction of coverage scenarios

Eq. (4) computes the quantity of links (Q_L) that connect all of the $M \times N$ NoC routers. For example, $Q_L = 112$ links in an 8×8 mesh NoC

$$Q_L = M \times (N-1) + N \times (M-1) \quad (4)$$

Accounting for the scenario in which all of the links and/or each individual link could be faulty, Eq. (5) computes the maximum quantity of fault scenarios (Q_S), which is the combination of all of the possibilities of faulty links.

$$Q_S = 2^{QL} \quad (5)$$

As the number of scenarios grows exponentially with the number of faults, computing all of the possible scenarios is both time and memory consuming, even for a small percentage of links, which makes this approach unfeasible. For example, if 10% of the faulty links were considered in a 9×9 mesh NoC (i.e., 15 of the faulty links), it would be necessary to preprocess $Q_S = 2^{15}$ scenarios. 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 according to the way they were detected. Faults classified as static determine an irregular NoC topology that is perceived by the *Scenarios Processing Module* as a basic NoC topology. Consequently, independent of the quantity of static faults, there will be only a single scenario that represents the basic NoC topology. Over this basic topology, only the dynamic faults can perform temporary path changes, which implies computing new fault scenarios.

Although the total number of fault scenarios grows exponentially with the total number of faults, the number 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 that are previously known, and let Q_N be the quantity of new fault tendencies currently detected; then, Eq. (6) calculates the quantity of new scenarios to be computed (Q_{SC}). For example, if the system has 4 links with fault tendency already known ($Q_F = 4$) and the *Fault Monitors* detects two new links with fault tendency ($Q_N = 2$), then the *OsPhoenix* must preprocess 48 more scenarios (i.e., $2^{(4+2)} - 2^4$), to assure that the system will provide all of the fault scenarios.

$$Q_{SC} = 2^{(QF+QN)} - 2^{QF} \quad (6)$$

Identifying the most dissimilar scenarios that cover the same fault scenario allows us to save a reduced set of scenarios with a wider coverage. This wider and reduced coverage is achieved by sorting the new scenarios according to the dissimilarity level and storing only a percentage of the most dissimilar scenarios. When the percentage of storage is low, only the most 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 target architecture size and the fault quantity. We consider this percentage to be a selection criterion, and we provide analysis of this concept in Section 8.

6. OsPhoenix and HwPhoenix implementation results

This section describes the synthesis of the Phoenix architecture while targeting a mesh NoC-based MPSoC whose tiles contain a plasma processor and a local memory. This architecture was applied to collect experimental results on the quality of the analytical metrics and the costs associated with scenario preprocessing, which are described in Sections 7 and 8, respectively.

6.1. OsPhoenix implementation on the plasma processor

Each MPSoC tile consists of a 32-bit Plasma processor running at 100 MHz, whose source code is based on the processor available in the OpenCore site [32], with 256 KB of program memory and 256 KB of data memory. Running on the Plasma processor, the *OsPhoenix* works like a device-driver of the Hellfire Operating System [33], enabling a distributed and fault-tolerant operation of a network, which is transparent to the OS. Table 1 shows the data and code memory footprint for the Hellfire OS and for the main modules of *OsPhoenix* (i.e., *Kernel* and *Scenarios Processing Module*), in addition to the memory area that is left for user applications.

Hellfire OS and *OsPhoenix* are designed for low memory consumption, allowing approximately 80% of the code memory and more than 85% of the data memory for user applications. It is important to note that the *Scenarios Processing Module* consumes 32 KB of the data memory, which is almost all available to store the data tables that contain preprocessed scenarios. This consumption depends on the number of scenarios that the designer wants to store.

Fig. 10(a) illustrates the memory consumption of the *Scenarios Processing Module* during the execution of the preprocessing scenarios algorithms with respect to the NoC size variation, on average. Additionally, Fig. 10(b) shows the Plasma processing cost (in clock cycles and in seconds) for these same experiments.

The main implementation complexity of *OsPhoenix* is in the *Scenarios Processing Module* due to the network segmentation algorithm (which provides deadlock-free routes) and the regions calculation algorithm (which generates the routing tables for the NoC reconfiguration). Fig. 10(a) illustrates that during the execution of these algorithms, the data memory consumption of the Scenarios *Processing Module* grows proportionally with the NoC size, thereby reducing the data memory that is available for user applications. Although not shown in Fig. 10(a), the data memories of the Hellfire OS and the kernel of *OsPhoenix* only change slightly with the NoC size.

Fig. 10(b) shows that the processing time of these algorithms grows more than linearly with the NoC size. When employing the scenario preprocessing approach for these same experiments, the total processing time is reduced to 5500 cycles, on average. These figures support the feasibility and importance of the preprocessing approach because the identification of the scenario faults allows us to reconfigure the network approximately a thousand times faster.

Table 1
Average occupancy of data and code memories for each MPSoC tile.

Memory occupation	Hellfire OS	*OsPhoenix*		User application
		Kernel	*Scenarios Processing Module*	
Code in KB (%)	24 (9.38%)	4 (1.56%)	25 (9.77%)	203 (79.30%)
Data in KB (%)	4 (3.13%)	1 (0.78%)	32 (12.50%)	219 (85.54%)

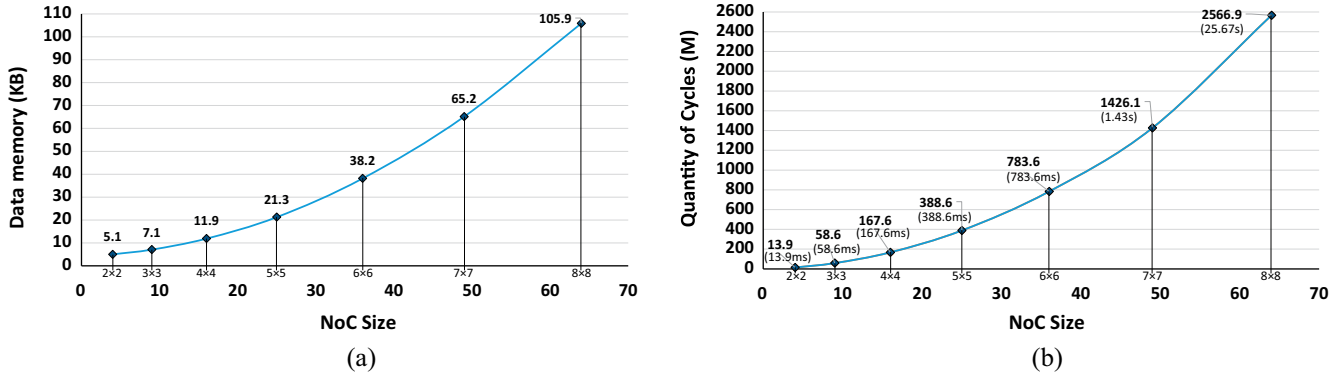


Fig. 10. Memory consumption of the *Scenarios Processing Module* and the CPU cost for scenarios preprocessing while accounting for seven NoC sizes. The CPU cost is given in millions of clock cycles and in seconds (Plasma operating at 100 MHz).

Table 2
Area and power reports of the Phoenix router with 16-flit depth buffers (synthesis with Encounter [34] using STM 65 nm CMOS and 100 MHz).

Characteristic	Router	*Fault Control Machine*	*Fault Monitor*	FPM (per channel)	HE (per channel)	HD (per channel)
Power (μW)						
Leakage	626.36	136.73	24.00	3.84	1.08	3.02
Dynamic	7112.25	1206.44	758.49	52.49	3.08	93.91
Total (%)	7738.61 (100%)	1343.17 (17.36%)	782.49 (10.10%)	56.33 (0.73%)	4.16 (0.05%)	96.92 (1.25%)
Area (nm^2)						
Cell area	58,159	12,003	1460	497	127	353
Net area	49,430	9309	2627	495	58	418
Total (%)	107,589 (100%)	21,312 (19.81%)	4087 (3.80%)	992 (0.92%)	185 (0.17%)	771 (0.72%)

6.2. HwPhoenix synthesis details

All of the fault-tolerant mechanisms of Phoenix are implemented within each NoC router, in the static monitoring module (i.e., *Fault Monitor*), in the modules for dynamic fault monitoring and correction (i.e., FPM, HE and HD), and in the central machine that coordinates these mechanisms and performs the communication with the *OsPhoenix* (i.e., *Fault Control Machine*). Table 2 presents the area and power reports of Phoenix router, highlighting the fault-tolerant modules, in a standard cell implementation (STM 65 nm CMOS) for a 100 MHz operation frequency and 16-flit depth buffers.

In the experimental setup, we employed a [16,5] Hamming code (i.e., 16 data bits and 5 bits of data redundancy), which was implemented using fixed masks that operate data and parity in a combination of blocks designed to affect minimally the operation frequency of the NoC links. Table 2 illustrates that HE consumes an insignificant portion of the router area and power. The ability to perform bit error detection and correction makes the HD circuit much more complex than the HE circuit. Consequently, HD has four times more area consumed and 20 times more power dissipated than HE. However, this increase in the complexity does not imply a significant portion of the area and power of the router. The FPM has the same magnitude of the HD circuit, consuming little more area due to the internal fault tables but dissipating less dynamic energy. As a consequence, the dynamic fault monitoring and correction mechanism dissipates only approximately 2% of the router power and consumes less than 2% of the router area, per channel.

While HD, HE and FPM are replicated on each channel between routers, there is a single centralized *Fault Monitor* to perform the static monitoring of all of the channels between routers. This monitoring is started by a control packet sent by *OsPhoenix* via a local link to the *Fault Control Machine*, which in turn controls the *Fault Monitor* to make a loopback test on all of the channels between routers. To accomplish this task, the *Fault Monitor* implements a low-complexity finite state machine whose area consumption is almost the same as the sum of all of the dynamic fault monitoring and correction mechanisms, but the power dissipation is almost double.

The *Fault Control Machine* is responsible for sending and receiving control packets, monitoring the status of the static and dynamic fault mechanisms and updating the *Fault Table*. This machine also updates the *Routing Table* according to the control packets. These features make the *Fault Control Machine* the higher power- and area-consuming circuit, occupying almost 20% of the NoC router area and dissipating more than 17% of the router power. Finally, all of the fault-tolerant circuits of Phoenix dissipate approximately 35% of the router power and consume less than 30% of the router area, which shows that the fault-tolerant model that is adopted (which implements part of the functionality in software and part in hardware) enables the production of low-cost and efficient fault-tolerant hardware.

7. Exploration of analytic metrics for runtime latency estimation

7.1. Experimental setup

The fabrication process variability of VLSI circuits increases during every scale-down of new, deep submicron technologies, due to phenomena such as imprecise impurity deposition and non-uniformity in the lithography exposure field. This variability can deviate the circuit from its nominal specification or even prevent its partial or total operation [35]. 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. [36] for generating the fault scenarios for the experimental results presented here. This model considers the effect of

Table 3
Expansion of fault scenarios for 48 irregular NoC mesh topologies.

NoC size	Distribution fault		Amount of fault channels (3 samples)	Amount of scenarios expanded (3 samples)	Total amount of scenarios
	σ	λ			
5 × 5	0.05	1.2	4, 4, 4	15, 15, 15	45
	0.18	1.2	4, 4, 4	15, 15, 15	45
	0.05	0.4	6, 6, 6	63, 63, 63	189
	0.18	0.4	6, 6, 6	63, 63, 63	189
6 × 6	0.05	1.2	5, 5, 5	31, 31, 31	93
	0.18	1.2	5, 5, 5	31, 31, 31	93
	0.05	0.4	7, 7, 7	127, 127, 127	381
	0.18	0.4	7, 7, 7	127, 127, 127	381
7 × 7	0.05	1.2	5, 5, 5	31, 31, 31	93
	0.18	1.2	5, 5, 5	31, 31, 31	93
	0.05	0.4	9, 9, 9	511, 511, 511	1533
	0.18	0.4	8, 8, 8	255, 255, 255	765
8 × 8	0.05	1.2	4, 4, 4	15, 15, 15	45
	0.18	1.2	5, 5, 5	31, 31, 31	93
	0.05	0.4	11, 11, 10	2047, 2047, 1023	5117
	0.18	0.4	10, 10, 10	1023, 1023, 1023	3069
Total			48 NoC topologies	12,224 scenarios	12,224

variability on the switch-to-switch link delay, which employs two variation parameters: the *link-delay variability* σ and the *spatial correlation variability* λ.

Aiming to explore scenarios with 65 nm and 22 nm manufacturing processes, the link-delay variability was set to 5% ($\sigma = 0.05$) and 18% ($\sigma = 0.18$), respectively, as predicted by the ITRS roadmap [37]. Additionally, experiments were produced with $\lambda = 0.4$ and $\lambda = 1.2$, which represent high and low strengths of the spatial correlation variability, respectively. These values are representative of the typical correlation that is induced by fabrication processes [36]. The experiments encompass four NoC sizes (5 × 5, 6 × 6, 7 × 7 and 8 × 8). Table 3 shows each NoC size combined with the link-delay and spatial correlation parameters, which produces 16 fault scenarios. Aiming to explore the randomness of the variability model, we generated each of these scenarios three times, resulting in 48 irregular NoC mesh topologies. Finally, an in-house tool expands these 48 topologies into 12,224 scenarios by combining all of the possibilities of faulty links.

Fig. 11 describes how the simulation scenarios are composed and applied to achieve the experimental results. For each of the 12,224 scenarios, the in-house tool performs the following two steps: (i) segmentation of the network using the segment routing approach to generate a restriction file. This file contains all of the forbidden directions, to avoid deadlock situations; and (ii) computation of minimal paths using the restriction file information, which allows us to generate the set of virtual regions for the RBR approach.

When there is an occurrence of a new fault, the *OsPhoenix* must select, out of the available preprocessed set of scenarios, the one that minimizes the overall system latency. Aiming to choose a sound runtime metric, we employed an RTL simulation with synthetic traffic to evaluate all of the scenarios. The simulation results were compared with the results of three analytic 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; (ii) Link Weight (LW), which is the number of communications that

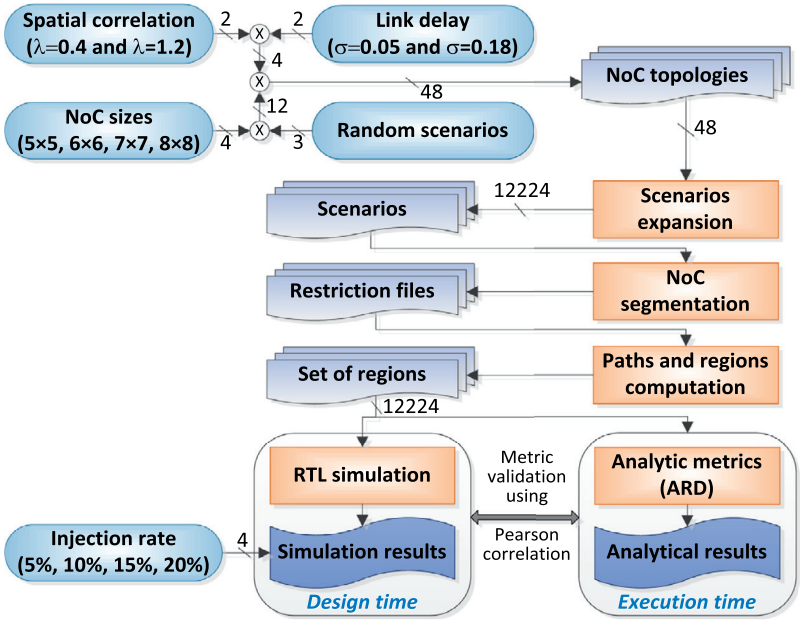


Fig. 11. Setup of experimental results.

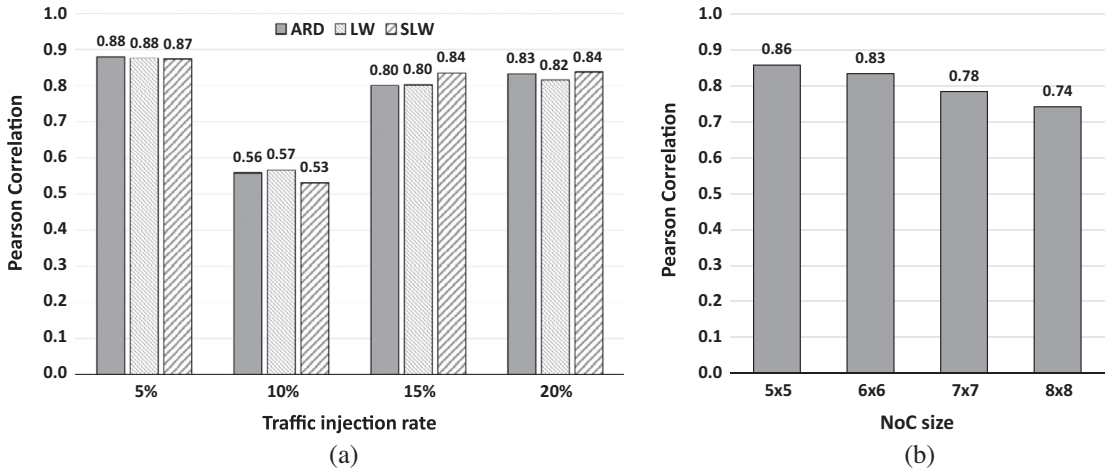


Fig. 12. Pearson correlation for the average latency: (a) comparison of 3 analytic metrics (i.e., ARD, LW and SLW) with uniform traffic simulations (injection rates of 5%, 10%, 15% and 20%); and (b) comparison of ARD with uniform traffic simulations, at an injection rate fixed at 5% and with 4 NoC sizes (5×5, 6×6, 7×7, 8×8).

crosses each link, considering the links direction; and (iii) Standard deviation of LW (SLW). Aiming to evaluate these metrics, we employ the Pearson correlation analysis [38] between the average latency simulated and the estimates provided by the analytic metrics. According to the Pearson correlation, the closer the correlation is to 1, the better the metric that is used. The experimental results are discussed next.

7.2. Experimental results for analytic metric selection

The experimental results were obtained using uniform traffic with four injection rates (5%, 10%, 15% and 20%) and for all of the 12,224 fault scenarios in which each PE sends 50 packets of 100 flits. Fig. 12(a) shows that except for the 10% injection rate, all of the analytic metrics presented positive, significant and strong correlations (i.e., closer to 0.9).

The highest correlation values are achieved with a 5% injection rate because the NoC resources are not overloaded, which minimizes the packet contentions. At almost 10% of the injection rate, the NoC reaches the saturation point, where the traffic behavior is unpredictable, which presents enormous variability in the communication latency. Consequently, the Pearson correlation is reduced for all of the analytic metrics. Finally, after the saturation point, the Pearson correlation increases again, which shows that the proposed analytic metrics can capture the behavior of a congested traffic scenario. Assuming ARD to be the better analytic metric for traffic with low injection rates, Fig. 12(b) shows the experimental results of the Pearson correlation according to the NoC sizes. This figure highlights that the increase in the NoC size reduces the Pearson correlation, which is justifiable due to the increase in the additional paths between the communication pairs and the increase in random packet collisions.

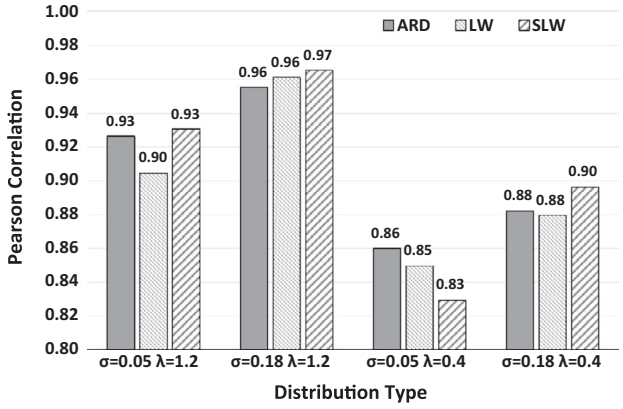


Fig. 13. Pearson correlation for the average latency: comparison of 3 analytic metrics (i.e., ARD, LW and SLW) under uniform traffic simulations, with an injection rate fixed at 5% and 4 types of fault distribution.

Fig. 13 illustrates the quality of the analytic metrics (i.e., ARD, LW and SLW) for estimating the average latency according to the type of fault distribution, i.e., the link-delay variability considering CMOS manufacturing technology ($\sigma = 0.05$ / 65 nm, $\sigma = 0.18$ / 22 nm) associated with the strength of the spatial correlation variability (i.e., a high or low strength for $\lambda = 0.4$ or $\lambda = 1.2$, respectively). Accounting for the spatial correlation variability, the highest correlations between the analytic metrics and the average latency are obtained in networks with $\lambda = 1.2$ because this spatial correlation produces the lowest degree of severity faults, the greatest fault dispersion and a lower percentage of faults. A high variability in the spatial correlation ($\lambda = 0.4$) reduces the correlation degree between the analytic metrics and the average latency because it produces scenarios with a high quantity of faults that are highly grouped. Nevertheless, the experimental results display a strong correlation for $\lambda = 0.4$, which shows that the analytic metrics can be used to estimate the average latency even with these aggressive fault distributions. Finally, the link-delay variability shows that the analytic metrics have a stronger correlation with $\sigma = 0.18$ than $\sigma = 0.05$, which displays a tendency to achieve sound results for the most recent technologies.

Fig. 14 illustrates the quality of the analytic metrics according to the number of fault links. It is evident that there is a reduction in the Pearson correlation with an increase in the number of faulty links. This finding can be explained because occurrences of faulty links imply a reduction in minimal paths. Consequently, more communications share the same paths, which increases the concurrency for the NoC resources (i.e., the input buffers and links of the router). This circumstance implies more packet contention and more unpredictable latencies, which are not captured by the estimates of the analytic metrics.

The simulation results show that the three analytic metrics produce similar and satisfactory estimates for the end-to-end average latency. However, the ARD is less susceptible to an increase in the number of faulty links. Thus, we chose to employ this metric in the *Scenarios Processing Module* of *OsPhoenix* for the execution of all of the experiments.

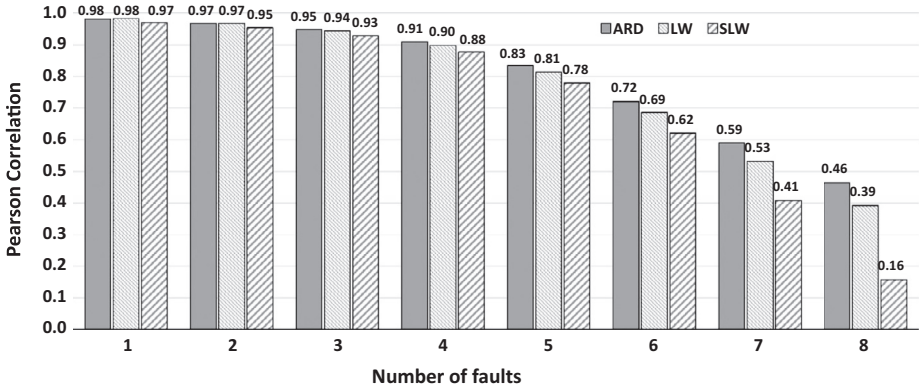


Fig. 14. Results of the Pearson correlation when comparing the average latencies estimated by analytic metrics with the RTL simulation with a 5% uniform traffic injection rate and random faults ranging from 1 to 8.

8. Analysis of methods and costs of the preprocessing approach

This section displays the experimental results that are employed to quantify the *coverage penalization*, as defined in Section 5.1, and analyze the efficacy and efficiency of both the LsHV and LsC cross-correlation methods in searching for a suitable reduced set of coverage scenarios.

8.1. Experimental setup to estimate the coverage penalization

Fig. 15 shows the flow that is employed in each experiment, which encompasses the 48 basic mesh topologies detailed in Table 3 and two traffic injection rates (5% and 20%). Additionally, for each cross-correlation method, we use three different percentages of stored fault scenarios (10%, 30% and 50%).

As described in Section 5.5, a given set of faults alters a mesh NoC into an irregular mesh topology, which is considered here to be a basic topology. Each basic topology encompasses Q_{SC} possible scenarios. For example, a 6×6 NoC with 7 faulty links produces 127 fault scenarios (i.e., $2^7 = 128$, but one scenario does not contain faulty links). The set of 48 scenarios with its corresponding quantity is surrounded by a rounded rectangle in Fig. 15. From one of this set of scenarios ①, we can apply the LsHV and LsC cross-correlation methods, selecting a percentage P of the most relevant scenarios ②. The remaining scenarios $(1 - P)$ are used to evaluate the *coverage penalization*, because none of these scenarios are

NoC size	σ	λ	Q_{SC}
5×5	0.05	1.2	15, 15, 15
	0.18	1.2	15, 15, 15
	0.05	0.4	63, 63, 63
	0.18	0.4	63, 63, 63
6×6	0.05	1.2	31, 31, 31
	0.18	1.2	31, 31, 31
	0.05	0.4	127, 127, 127
	0.18	0.4	127, 127, 127
7×7	0.05	1.2	31, 31, 31
	0.18	1.2	31, 31, 31
	0.05	0.4	511, 511, 511
	0.18	0.4	255, 255, 255
8×8	0.05	1.2	15, 15, 15
	0.18	1.2	31, 31, 31
	0.05	0.4	2047, 2047, 1023
	0.18	0.4	1023, 1023, 1023

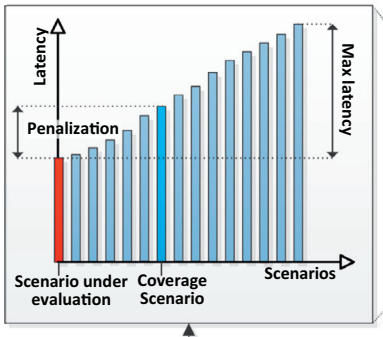


Fig. 15. Flow for the cross-correlation method and cost analysis, which is applied to each set of scenarios that compose each one of the 48 basic mesh NoC topologies.

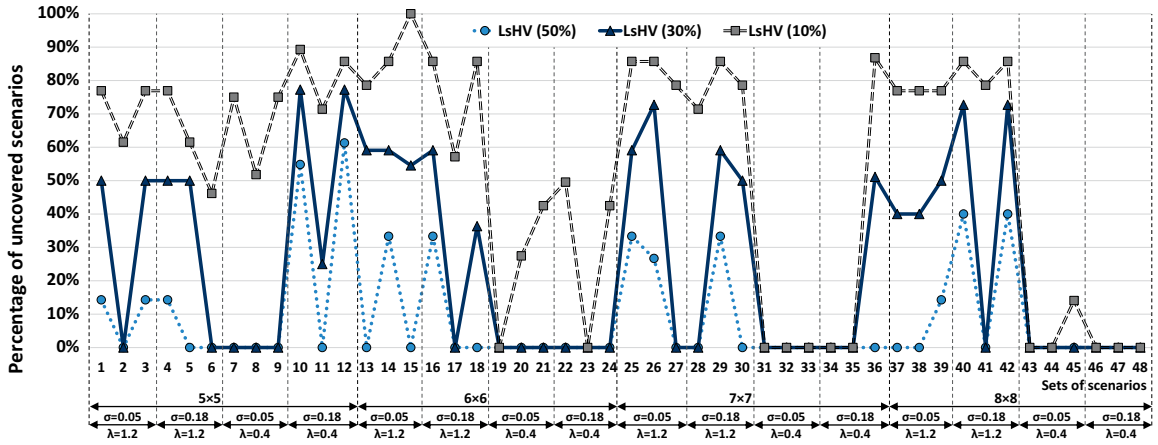


Fig. 16. Percentage of uncovered scenarios when applying the LsHV dissimilarity method. The experiment evaluates all 48 basic NoC topologies with three percentages of scenario selection (10%, 30% and 50%).

stored at runtime ③. Once a scenario is selected to be evaluated ④, the flow, which reflects the *OsPhoenix* operation, checks whether at least one stored scenario covers the selected one ⑤. In the negative case, the flow marks this non-success to evaluate the efficacy of the LsHV and LsC methods ⑥ (at runtime, the network must stop waiting for the processing of the new routing tables that cover this fault situation, which is a timing-consuming task). Otherwise, the flow selects, from the set of coverage scenarios, the scenario that produces the lower latency ⑦ (at runtime, the *OsPhoenix* uses this scenario for NoC reconfiguration, and the scenario latency is estimated using ARD). Thus, the selected scenario and the scenario under evaluation are compared in terms of the latency – i.e., the *coverage penalization* is quantified ⑧. These latency results, which are computed by Eq. (6), are stored for statistical analysis ⑨. The flow performs these previous steps until all of the remaining scenarios are evaluated ⑩. Once all of the scenarios are evaluated, the flow stops ⑪, and a new experiment can be evaluated.

8.2. *Experimental results for* coverage penalization *and correlation methods analysis*

The first set of results explores the efficacy of both cross-correlation methods in searching for sound coverage scenarios. The results determined that the LsC method did not present any uncovered scenarios, which shows that it is efficacious in finding the best scenarios according to the coverage criterion. This finding occurs because LsC selects the most dissimilar scenarios, which provides large coverage possibilities. On the other hand, Fig. 16 shows that the LsHV is less efficacious, because this method produced a large quantity of uncovered scenarios, even storing 50% of the scenarios. Additionally, Fig. 16 shows three other aspects about the efficacy of the LsHV method: (i) it is independent of CMOS manufacturing technology (i.e., $\sigma = 0.05/65$ nm, $\sigma = 0.18/22$ nm), which highlights the potential of the approach for other CMOS technologies; (ii) it increases with an

$$\textit{Coverage penalization} = \frac{\text{Average Latency}_{\text{Selected Scenario}} - \text{Average Latency}_{\text{Scenario Under Evaluation}}}{\text{Average Latency}_{\text{Scenario Under Evaluation}}}\% \quad (7)$$

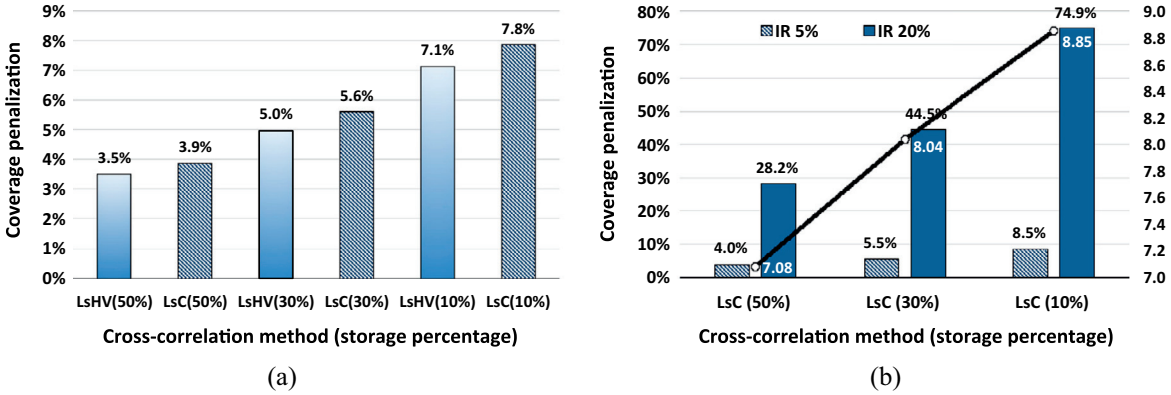


Fig. 17. Average of the *coverage penalization* for a 10%, 30% and 50% fault scenario reduction: (a) the LsHV and LsC methods under a 5% Injection Rate (IR); and (b) LsC method under a 5% and 20% IR.

increase in the NoC size. This relationship occurs because the increase in the NoC size sparsifies the faults, making the capture of dissimilarities easy; (iii) it increases with a reduction in λ because a reduction in λ means an increase in the spatial correlation that produces scenarios with more faulty links. In turn, the increase in the faulty links allows it to find more dissimilarities between the scenarios.

Fig. 17 (a) presents the *coverage penalization*, which compares the efficiency of the LsHV and LsC methods, considering only a 5% traffic injection rate and only the scenarios whose coverage was found by both methods. Although the LsHV method does not show high efficacy in finding sound coverage scenarios, the experiments highlight that (i) LsHV presents a lower *coverage penalization*, but the increase in the *coverage penalization* for the LsC method is not meaningful; and (ii) the *coverage penalization* increases slightly with the reduction in the scenarios storage, which indicates that the cross-correlation to identify dissimilarities is a promising technique for the scenarios reduction approach.

Fig. 17(b) depicts the average *coverage penalization*, accounting for two traffic injection rates (IR 5% and IR 20%), four NoC sizes (5×5, 6×6, 7×7 and 8×8) and only the LsC method for all of the 48 scenarios in the set of scenarios. It shows that the *coverage penalization* increases significantly with the increase in the network traffic. For example, with 50% of the storage scenarios, the increase from IR 5% to IR 20% implies an increase from 4.0% to 28.2% in the *coverage penalization*. This finding can be explained because the coverage scenario has fewer operational links, which increases the traffic competition for the same NoC resources. Furthermore, the increase in the network traffic that is associated with the reduction in the coverage scenarios produces an additional increase in the *coverage penalization*. For example, with 50% of the storage scenarios, the increase from IR 5% to IR 20% implies 7.08 times more *coverage penalization*, while with 10% of the storage scenarios, the same increase in IR implies 8.85 times more *coverage penalization*. This last situation argues against the employment of the approach of coverage scenario reduction. However, in practice, only a few sets of IO-bounded applications produce large IRs, on average. Additionally, our synthetic experiments produce large quantities of data without compromising CPU utilization, and they reached only 25% IR, even using a DMA channel.

Fig. 18 shows the variation in the *coverage penalization* with an increase in the NoC size while considering the LsC method and IR 20%, i.e., the same experiment as depicted in Fig. 17(b) – column LsC (50%) + IR 20%, but identifying the average of the *coverage penalization* for each NoC size.

These last results show that the *coverage penalization* decreases when the NoC size increases. This relationship occurs because larger NoCs contain many more links, enabling them to produce efficient routing paths even in the presence of unused links. Accounting for recent and future systems that contain large quantities of PEs, which require large NoCs and in which the traffic injection rate is almost less than 20%, the *coverage penalization* tends to be less significant, which justifies an even larger preprocessing approach usage.

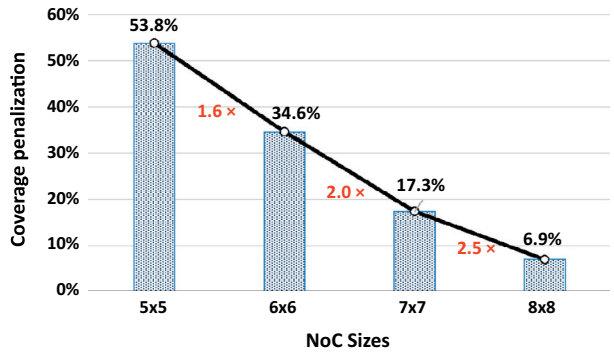


Fig. 18. Average of the *coverage penalization* for 50% of the fault scenario reductions, when applying the *LsC* methods under a 20% traffic injection rate and considering four NoC sizes.

9. Conclusions

This paper proposes a hardware/software reconfiguration approach that is based on preprocessing fault scenarios. The software is a small part of the operating system kernel called *OsPhoenix*, which preprocesses fault scenarios as soon as a fault prediction monitor (monitors are placed on each link of each router) detects a fault tendency. The hardware part is a fault-tolerant mesh NoC, which employs a region-based routing mechanism.

The preprocessed scenarios approach reduces the time that the network is halted, waiting for the computation of the routing algorithm that enables the NoC operation in the occurrence of new faults. The quantity of scenarios grows exponentially with the quantity of faults, which implies that there is a large area of memory and processing time to compute all of the scenarios in the set of scenarios. Aiming to minimize this problem, this work employs three strategies: (i) differential treatment for static and dynamic faults, (ii) incremental processing of fault scenarios, and (iii) a dissimilarity approach, which enables finding the most dissimilar scenarios based on the 2D cross-correlation method.

We concluded that the preprocessed scenarios approach, in conjunction with the analysis of fault tendency detection, allows us to preprocess sound coverage scenarios, enabling a faster reconfiguration and reducing the time that the network is halted waiting for the computation of the routing algorithm that handles topology modifications. The preprocessing of scenarios implies a *coverage penalization*, which is the difference between the latency that is achieved with an optimum solution and the latency that is achieved by the selected coverage scenario. However, according to the experimental results, the *coverage penalization* is not meaningful and tends to be slower when the traffic injection rates are reduced and when the NoC size is increased. Furthermore, when compared with on-demand processing approaches, the preprocessing of fault scenarios approach enables a larger CPU time to compute the routing algorithm, which enables communication paths to be selected that minimize the overall latency and increase the time for the system operation, which highlights the efficiency of the preprocessing approach proposed in this work.

References

[1] International Technology Roadmap for Semiconductors (ITRS), ITRS 2013 ed. <www.itrs.net/reports.html> (captured in April 2014).
[2] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, J. Duato, Cost efficient on-chip routing implementations for CMP and MPSoC systems, IEEE Trans. Computer-Aided Des. Integr. Circuits Syst. (TCAD) 30 (4) (2011) 534–547.
[3] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, Y. Hoskote, Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives, IEEE Trans. Computer-Aided Des. Integr. Circuits Syst. (TCAD) 28 (1) (2009) 3–21.
[4] F. Arnaud, L. Pinzelli, C. Gallon, M. Rafik, P. Mora, F. Boeuf, Challenges and opportunity in performance, variability and reliability in sub-45 nm CMOS technologies, Microelectron. Reliability 51 (9–11) (2011) 1508–1514.
[5] E. Ioannidis, S. Haendler, C. Theodorou, S. Lasserre, C. Dimitriadis, G. Ghibaudo, Evolution of low frequency noise and noise variability through CMOS bulk technology nodes from 0.5 μm down to 20 nm, Solid-State Electron. 95 (2014) 28–31.
[6] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. Lopez, R. Holsmark, J. Duato, Region-based routing: a mechanism to support efficient routing algorithms in NoCs, IEEE Trans. Very Large Scale Integr. VLSI Syst. 17 (3) (2009) 356–369.
[7] W. Dally, H. Aoki, Deadlock-free adaptive routing in multicomputer networks using virtual channels, IEEE Trans. Parallel Distributed Syst. (PDS) 4 (4) (1993) 466–475.

[8] F. Chaix, D. Avresky, N.-E. Zergainoh, M. Nicolaidis, Fault-tolerant deadlock-free adaptive routing for any set of link and node failures in multi-cores systems, in: IEEE International Symposium on Network Computing and Applications (NCA), 2010, pp. 52–59.
[9] M. Valinataj, S. Mohammadi, J. Plosila, P. Liljeberg, A fault-tolerant and congestion-aware routing algorithm for networks-on-chip, in: IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010, pp. 139–144.
[10] C. Jackson, S. Hollis, A deadlock-free routing algorithm for dynamically reconfigurable Networks-on-Chip, Microprocess. Microsyst. 35 (2) (Mar. 2011) 139–151.
[11] S. Pasricha, Y. Zou, NS-FTR: a fault tolerant routing scheme for Networks on Chip with permanent and runtime intermittent faults, in: Asia and South Pacific Design Automation Conference (ASPDAC), 2011, pp. 443–448.
[12] M. Ebrahimi, M. Daneshtalab, J. Plosila, H. Tenhunen, MAFA: adaptive fault-tolerant routing algorithm for Networks-on-Chip, in: Euromicro Conference on Digital System Design (DSD), 2012, pp. 201–207.
[13] C. Glass, L. Ni, Fault-tolerant wormhole routing in meshes without virtual channels, IEEE Trans. Parallel Distributed Syst. (PDS) 7 (6) (Jun. 1996) 620–636.
[14] K.-H. Chen, G.-M. Chiu, Fault-tolerant routing algorithm for meshes without using virtual channels, J. Inf. Sci. Eng. (JISE) 14 (4) (1998) 765–783.
[15] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, T. Skeie, Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori, in: International Parallel and Distributed Processing Symposium (IPDPS), 2006, pp. 1–10.
[16] K. Aisopos, A. DeOrio, L.-S. Peh, V. Bertacco, ARIADNE: agnostic reconfiguration in a disconnected network environment, in: International Conference on Parallel Architectures and Compilation Techniques (PACT), October 2011, pp. 298–309.
[17] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, J. Hu, G. Chen, A reliable routing architecture and algorithm for NoCs, in: IEEE Transaction on computer-aided design of integrated circuits and systems (TCAD), vol. 31, no. 5, May 2012.
[18] M. Palesi, S. Kumar, R. Holsmark, A method for router table compression for application specific routing in mesh topology NoC architectures, in: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2006, pp. 373–384.
[19] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, Routing table minimization for irregular mesh NoCs, Design, Automation & Test in Europe Conference & Exhibition, (DATE), 2007, pp. 16–20.
[20] C. Marcon, A. Amory, T. Webber, T. Volpato, L. Poehls, Phoenix NoC: a distributed fault tolerant architecture, in: International Conference on Computer Design (ICCD), 2013, pp. 7–12.
[21] M. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. Nordbotten, O. Lysne, T. Skeie, An efficient fault-tolerant routing methodology for meshes and tori, Comput. Architecture Lett. 3 (1) (2004) 1–4.
[22] C.-T. Ho, L. Stockmeyer, A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers, IEEE Trans. Comput. (TC) 53 (4) (2004) 427–439.
[23] A Strano, D. Bertozzi, F. Triviño, J. Sánchez, F. Alfaro, J. Flich, OSR-lite: fast and deadlock-free NoC reconfiguration framework, in: International Conference on Embedded Computer Systems (SAMOS), 2012, pp. 86–95.
[24] D. Fick, A. DeOrio, J. Hu; V. Bertacco, D. Blaauw, D, Sylvester, Vicis: a reliable network for unreliable silicon, in: ACM/IEEE Design Automation Conference (DAC), 2009, pp. 812–817.
[25] F. Triviño, J. Sánchez, F. Alfaro, J. Flich, Network-on-Chip virtualization in chip-multiprocessor systems, J. Syst. Architecture (JSA) 58 (3–4) (2012) 126–139.
[26] Y. Fukushima, M. Fukushi, I. Yairi, A region-based fault-tolerant routing algorithm for 2D irregular mesh Network-on-Chip, J. Electron. Test. 29 (3) (2013) 415–429.
[27] R. Holsmark, M. Palesi, S. Kumar, Deadlock-free routing algorithms for irregular mesh topology NoC systems with rectangular regions, J. Syst. Architecture (JSA) 54 (3-4) (2008) 427–440.
[28] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, J. Duato, Efficient implementation of distributed routing algorithms for NoCs, IET Comput. Digital Tech. 3 (5) (2009) 460–475.
[29] C. Feng, Z. Lu, A. Jantsch, M. Zhang, Z. Xing, Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router, IEEE Trans. Very Large Scale Integr. VLSI Syst. 21 (6) (2013) 1053–1066.
[30] L. Dai, D. Shang, F. Xia, A. Yakovlev, Monitoring circuit based on threshold for fault-tolerant NoC, Electron. Lett. 46 (14) (Jul. 2010) 984–985.
[31] B. Pan, K. Qian, H. Xie, A. Asundi, Two-dimensional digital image correlation for in-plane displacement and strain measurement: a review, Meas. Sci. Technol. 20 (6) (2009) 1–17.
[32] Open Cores, Plasma most mips i(tm) opcodes. <http://opencores.org/project,plasma> (captured in April 2015).
[33] A. Aguiar, S. Johann, F. Magalhaes, T. Casagrande, F. Hessel, Hellfire: a design framework for critical embedded systems' applications, in: International Symposium on Quality Electronic Design (ISQED), March 2010, pp. 730–737.
[34] Cadence, Encounter RTL compiler. <www.cadence.com/products/> (captured in April 2015).
[35] M. Shintani, T. Uezono, T. Takahashi, K. Hatayama, T. Aikyo, K. Masu, T. Sato, A variability-aware adaptive test flow for test quality improvement, IEEE Trans. Computer-Aided Des. Integr. Circuits Syst. (TCAD) 33 (7) (2014) 1056–1066.
[36] B. Hargreaves, H. Hult, S. Reda. Within-die process variations: how accurately can they be statistically modeled? in: Asia and South Pacific Design Automation Conference (ASP-DAC), 2008, pp. 524–530.
[37] International Technology Roadmap for Semiconductors (ITRS), ITRS 2007 ed. <http://www.itrs.net/reports.html (captured in April 2014).
[38] P. Lena, L. Margara, Optimal global alignment of signals by maximization of Pearson correlation, Inf. Process. Lett. 110 (16) (2010) 679–686.

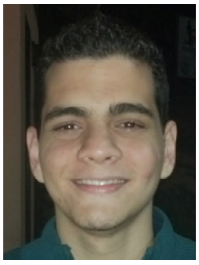
Jarbas Aryel Nunes Silveira, M.Sc. is an Assistant Professor at Teleinformatics Department (DETI), Federal University of Ceará (UFC), Brazil, since 2009. He received his M.Sc. in Teleinformatics Engineering from Federal University of Ceará, Brazil, in 2006. His research interests are in the areas of embedded systems on digital circuits, computer architecture, on-chip communication architectures, fault tolerance, and real-time systems. At UFC, he works at the Engineering Laboratory Computer Systems (LESC). Nowadays he is currently pursuing his doctorate in on-chip communication architectures at DETI/UFC, Brazil.

César Augusto Missio Marcon, Ph.D. is Associate Professor at the School of Computer Science of Pontifical Catholic University of Rio Grande do Sul, Brazil, since 1995. He received his Ph.D. in Computer Science from Federal University of Rio Grande do Sul, Brazil, in 2005. His research interests are in the areas of embedded systems on the telecom domain, computer architecture, on-chip communication architectures, partitioning and mapping application tasks, software & hardware testing, fault tolerance, parallel processing, real-time operating systems. Currently, he participates in five research projects and he works as advisor for M.Sc. and Ph.D. graduate students.

Paulo César Cortez, Ph.D. is a Professor with the Department of Teleinformatics Engineering, Federal University of Ceará (UFC), Brazil. He received the B.Sc. degree from the Federal University of Ceará in 1982, and the M.Sc. and Ph.D. degrees from the Federal University of Paraíba, João Pessoa, Brazil, in 1992 and 1996, respectively, all in electrical engineering. His fields of interest include image and signal analysis.

Giovani Cordeiro Barroso, Ph.D. received the M.Sc. degree in electrical engineering from Pontifical Catholic University of Rio de Janeiro, Brazil, in 1986 and the Ph.D. degree in electrical engineering from the Federal University of Paraíba, Brazil, in 1996. He became a Postdoctoral Fellow with the Institute National des Télécommunications—INT, Evry, France, in 2003. Currently, he is Titular Professor at the Federal University of Ceará, Brazil. He has experience in electrical engineering with emphasis on supervisory control, acting on the following topics: Petri nets, modeling and analysis, distributed systems, and discrete event systems.

João Marcelo Ferreira is a Student Member of the IEEE. He is currently pursuing an undergraduate course in electrical and computer engineering at Universidade Federal do Ceará (UFC), Brazil, since 2009. He is also undergoing another undergraduate course in mechatronics engineering at Instituto Federal do Ceará (IFCE). His research interests are in the areas of embedded systems on digital circuits, computer architecture, on-chip communication architectures and real-time system and his current research focuses on developing methodologies for traffic balancing in Networks-on-Chip. Currently, he participates in three research projects on communication architectures.



Rafael Gonçalves Mota is an undergraduate student at Teleinformatics engineering Department (DETI) of Federal University of Ceará (UFC), Brazil. At UFC, he works in the Laboratory of Computer Systems Engineering (LESC) since 2010. His researches interests are in embedded systems, computer architecture, on-chip communication architectures, real-time and fault tolerant systems. Nowadays, his researches are focused in networks-On-Chip (NoCs).