A Fast Centralized Computation Routing Algorithm for Self-Configuring NoC Systems

Francisco Triviño, Francisco J. Alfaro, José L. Sánchez
Universidad de Castilla-La Mancha
Campus Universitario, s/n 02071, Albacete, Spain
Emails: {francisco.trivino, fco.alfaro, jose.sgarcia}@uclm.es

José Flich
Universitat Politècnica de València
Department of Computer Engineering
Email: jflich@disca.upv.es

Abstract

As technology evolves, networks-on-chip will need to survive to manufacturing faults in order to sustain yield. An effective configuration strategy implies the design of an efficient routing infrastructure, that enables a fast and efficient configuration of the NoC system to go around faulty links and switches. The strategy must minimize the overhead in resources and guarantee the entire system to be deadlock free. A centralized approach, through a monitoring controller is appealing as will get global network visibility.

This paper proposes a centralized routing configuration strategy that meets the requirements by means of a fast configuration algorithm for the most common failure patterns. The strategy is designed towards the goals of reduced configuration time and high coverage support (maximum number of supported failure patterns). No extra resources (virtual channels) are needed for the effective final configuration of the system. Results show the effectiveness of the proposed configuration algorithm.

1. Introduction

With nanoscale silicon technologies advancing, designing a fault-tolerant chip multi-processor (CMP) system is becoming a key issue to sustain both manufacturing yield and system lifetime. Manufacturing defects and wear-out faults will need to be supported. More specifically, the Network-on-Chip (NoC) (the communication backbone of the chip) will need to work around one or more component failures to preserve connectivity and chip usability.

The capability of surviving manufacturing and wear-out faults in the network closely depends on the flexibility of the routing algorithm [1]. Some common routing algorithms such as dimension-order routing are very simple and even ensure deadlock freedom, but they assume a regular 2D mesh structure. However, faults either at manufacturing or during the life time of the chip will impact network regularity. As a consequence, a defective node or link in the network will ruin the 2D mesh structure and lead to an irregular network that cannot be handled by the routing algorithm, thus rendering the network unusable.

Such failures in the network can be properly addressed by deploying a flexible routing algorithm able to deal with any of the topologies derived from a 2D mesh with some link or switch failures[1]. Both source routing (the complete path is coded in the packet header) and distributed routing (the destination of the packet is coded in the packet header and at every switch the output port is computed) can implement flexible fault-tolerant routing algorithms. However, such flexibility is mostly obtained by using forwarding tables either at sources (source routing) or at switches (distributed routing).

Recently, a new method of implementing routing algorithms for NoCs has been proposed. Logic-Based Distributed Routing (LBDR) [16] allows to implement algorithms by using an small set of bits at every switch. With LBDR the algorithm is translated into routing and connectivity bits and those bits are distributed over the switches. Most of the topologies derived from a 2D mesh when failures change the topology can be supported in LBDR, as demonstrated in [16].

However, the time required to compute those bits for a certain routing algorithm and topology can be high and may affect the effectiveness of the entire system. This is the case when non-minimal paths need to be supported in LBDR (by using the so-called deroute bits). This could be the case of a system that includes a NoC with some components failed. When the system is powered on the topology needs to be discovered (to find faulty components) and the routing algorithm is applied (the set of paths or the LBDR bits are computed). As this may take time (even in the order of milliseconds), the system is unavailable during that process.

This paper aims at overcoming the manufacturing faults and the onset of wear-out faults in a effectively way. A fast method to compute the routing algorithm is proposed, thus reducing significantly the system unavailability. We use the LBDR approach to implement the routing algorithm. This enables a compact and efficient representation of the routing info that needs to be distributed over the switches (as opposed to large forwarding tables).

We assume a centralized configuration mechanism where a global controller with full visibility of the network state is in charge of computing the new LBDR bits for the routing algorithm once the location of the failed components in the network is known.

For this purpose, we engineer the LBDR bits that will be deployed for every possible failure combination. The algorithm must be flexible enough in order to support most of the typical failure patterns. Moreover, it needs to be implemented in an efficient way avoiding the use of expensive resources (mainly storage resources). The configuration algorithm will be executed by the global controller and will be based on the

1. In this paper we focus on initial 2D mesh topologies for CMP systems.

978-1-4577-1950-9/11/$26.00 ©2011 IEEE

Segment-Based routing algorithm (SR) [14]. With SR, a set of initial paths is computed for a fault-free 2D mesh network and the proper LBDR bits are set.

We extend the SR algorithm in order to support failure patterns. In particular, we deduce the failure support properties of the SR algorithm and deduce the minimal number of LBDR bits that need to be changed in the face of failures. The algorithm is able to tolerate any possible number of link failures (compatible with the capability of the routing mechanism) but we optimize the specific case of either one or two link failures achieving a very fast configuration time for those cases, since in practice they are more likely to occur. Indeed, we derive the proper changes in selective LBDR bits that make the overall network fault-tolerant. This enables us to provide a fast configuration method of the NoC infrastructure.

The remainder of this paper is organized as follows: Section 2 presents the related work. In Section 3 we briefly describe the LBDR routing mechanism. Then, in Section 4 we present the configuration algorithm used to support network configuration. Section 5 details the performance evaluation and shows the obtained results. Finally, Section 6 presents some conclusions.

2. Related Work

Configurable routing for tackling the problem of fault tolerance in the NoC domain has been proposed in several forms. Unfortunately, most of them utilize distributed approaches that call for additional resources as well as the need to higher computational costs.

In [10], a heuristic search algorithm for re-routing in on-chip network is presented. The algorithm searches for a valid configuration of the routing tables in the network. Unfortunately, the algorithm cannot guarantee that every generated configuration is valid, therefore, a manual check is required. The configuration routing strategy we present in this work supports the most common failure patterns and reaches high coverage support.

Authors in [7] present another re-routing algorithm which does not guarantee 100% deadlock freedom, requires flooding strategies and increases the transmission time due to ping-pong re-routing. Furthermore, such approach demands for considerable buffering resources in order to keep a history of all the packets traversing a switch. Our configuration algorithm only needs the location of the failed components in the network.

In [2] a distributed reconfiguration algorithm for NoCs is proposed. Authors claim that no virtual channels are required and 100% coverage is achieved for a number of failures up to 10% of links. Moreover, 99.99% of failure combinations are supported for larger number of failed links. Although the claimed result is impressive, the paper does not detail the basic routing algorithm adopted to deal with *rule checking* for deadlock freedom. Moreover, special failure patterns have to be addressed with complex communication structures like broadcast. The paper does not state the internals of the switch. Moreover, the justification regarding deadlock freedom is weak. Authors also indicate that the reconfiguration algorithm is computed off-line, which is not compatible with a local reconfiguration mechanism. The computation time required (of utmost importance), is not provided, though. Our configuration algorithm is designed in order to be executed in a global controller with full visibility for a local configuration. For this reason, we have developed the configuration algorithm as fast as possible in order to minimize the system unavailability.

In [6], authors propose a custom methodology, based on packet rerouting, to handle data transfers upon power management events or system faults. However, their approach is not general and is actually working around faults in the attached cores, not in the NoC itself. Deterministic routing is characterized by its simplicity and minimal overhead; it can be easily configured to avoid deadlocks [4] and natively guarantees in-order delivery. Unfortunately, deterministic routing does not adjust to the system evolution over time; on the contrary, dynamic routing has been proposed to achieve goals such as bypassing faulty nodes and minimizing congestion [5]. Dynamic routing needs decentralized decision processes and is therefore often achieved with dedicated logic in every router [3].

In contrast with previous works, this paper proposes a fast configuration algorithm to compute the routing algorithm for fast execution for the common failure patterns. The use of the LBDR approach to implement the routing algorithm allows us to enable a compact and efficient representation of the routing info that needs to be distributed over the switches instead of the traditional large forwarding tables. For this reason, our method minimizes the overhead of routing info that needs to be distributed. This fact allows us to significantly reduce the system unavailability by materializing fast configuration in practice.

Finally, a distinctive feature of this work is the accuracy in the validation of the proposed configuration algorithm, evaluated by executions on a virtual platform with cycle accuracy using real applications and by actual synthesis runs in 65nm technology to evaluate the configuration time.

3. The LBDR Routing Mechanism

In this section we describe the switch routing mechanism to be programmed in order to support network configuration. In an attempt to provide better scalability of switch area and delay to large network sizes, logic-based distributed routing (LBDR) is more appealing to our approach than a full table-based routing mechanism. Although register-based implementations of memory macros could be a competitive solution for small networks, this technique does not scale. However, the LBDR complexity just depends on the number of switch I/O ports and not on the network size [8]. At the same time, this choice would reduce the amount of signaling between NoC switches and a global controller, since configuration of the routing logic depends on just a few configuration bits.

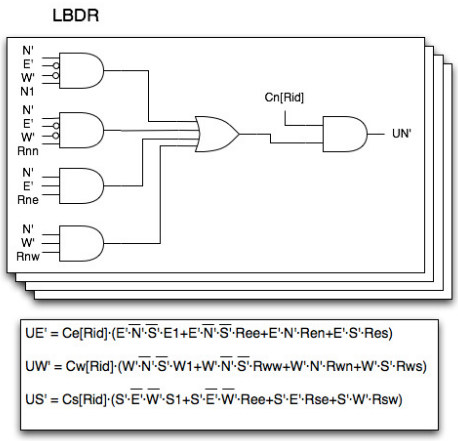


Fig. 1. Route computation logic.

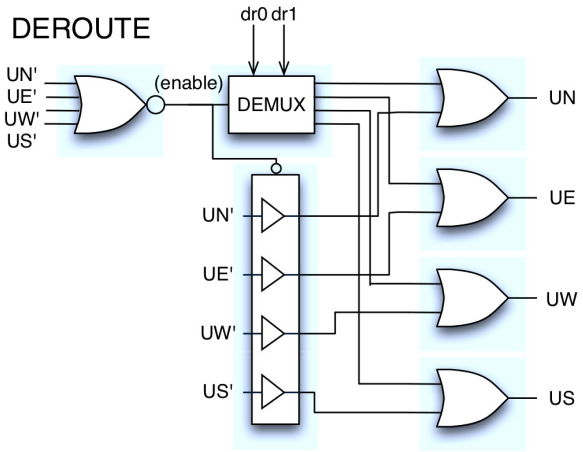


Fig. 2. Deroute logic.

The possibility to implement logic-based distributed routing, while retaining the flexibility of forwarding tables has been recently demonstrated in [9]. In practice, a route selection logic is located at each input port of a NoC switch (regardless of the specific switch architecture). LBDR consists of a selection logic of the target switch output port relying on a few switch-specific configuration bits (namely routing R_{xy}, connectivity C_x and deroute bits dr). The number of these bits (26 in the LBDR variant of this paper) is orders of magnitude smaller than the size of a forwarding table, what makes the reconfiguration mechanism quicker and less area demanding.

The core of the LBDR logic is illustrated in Fig. 1, showing the conditions that select the output port north UN' for packet routing. The routing decision is taken based on:

- the quadrant $N'/S'/W'/E'$ of packet destination (whose ID is embedded into the incoming packet);
- the routing restrictions posed by a deadlock-free routing algorithm (coded by the *routing bits*);
- the switch connections with the rest of the network (coded by the *connectivity bits*).

For some failure patterns, LBDR may not be able to find a route for the packet. In that case, a couple of additional *deroute bits* feed a logic (Fig. 2) which computes a valid output port to reach the destination. This extends the fault coverage of LBDR, which in [9] was proved able to work in roughly 60% of the irregular topologies derived from a 2D mesh. A larger coverage would require the NoC to revert to virtual cut-through switching and is therefore left for future work. More details on LBDR logic can be found in [9].

It is worth noting that LBDR is a method to implement a routing algorithm. DOR, up*/down*, or SR [14] are examples of routing algorithms that can be used. Based on the topology, the routing algorithm is computed, the paths for every pair of communicating flows along the location of the routing restrictions are obtained, and then, the LBDR bits are computed accordingly. Fig. 3 depicts the three stages needed to effectively apply LBDR and a routing algorithm on a topology. In the first stage the algorithm is computed and the routing restrictions are positioned in the topology. The complexity of this stage depends on the algorithm but mainly on the irregularity of the topology. An irregular topology requires a topology-agnostic routing algorithm (e.g. up*/down* and SR). For instance, in up*/down* a spanning tree is built and all the unidirectional links are labeled as *up* or *down*. Routing restrictions are placed in *down* $\rightarrow$ *up* transitions. We can term this complexity as mid since it is not a recursive process but neither as simple. In the second stage, the LBDR routing and the connectivity bits are obtained. This is a straightforward process since there is a direct relationship between the routing restrictions and the routing bits.

However, in an irregular topology LBDR also requires in some cases some non-minimal path support through the use

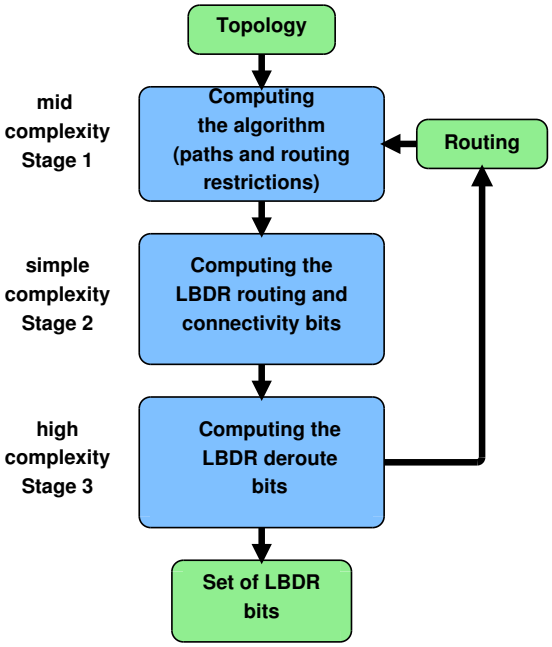


Fig. 3. Complete process to compute LBDR bits from a topology.

of deroutes. The way to compute such bits (third stage of the process), is costly as it needs to check connectivity for every possible communicating flow. If a flow is unconnected, then deroute options are placed and tested along the path in order to restore connectivity. This is a lengthy process since relies on a recursive search.

More important is the issue that a given topology, in combination with a routing algorithm, may not be suitable for LBDR. This is the case when deroutes are not enough to allow a valid path for a communicating flow. In that case, the LBDR approach can resort to a different routing algorithm in search of success. In order to provide such additional flexibility, the SR algorithm is a good routing candidate since it allows many instances of the SR methodology (ultimately, different routing instances that can be tested). However, it is easy to deduce that such approach comes with a significant processing overhead at the global controller and therefore should be taken just as an extreme course of action against hard-to-manage network failure patterns.

In this paper we will overcome the complexity of the previous process with a fast method to compute the LBDR bits of most of the usual failure patterns. The three stages and the routing algorithm (SR) will be embedded in a compact transition table. The table will be computed at design time and will provide the set of LBDR bits that need to be changed in order to support all the 1-link failure combinations and most of the n-link failure combinations. Therefore, the global controller will embed the transition table and upon notification of a topology change will compute the proper set of LBDR bits to be modified. The time required for the configuration algorithm will be reduced to accessing the transition table. This will be described later in the paper.

The configuration algorithm will need the list of failed links to recompute the configuration bits for obtaining a correct routing with the available communication resources.

4. The Configuration Algorithm

In this section we describe an effective, practical and fast method for configuring a NoC routing mechanism. As described previously, the algorithm assumes the use of the Segment-Based routing (SR [14]) algorithm, implemented with the LBDR mechanism. Basically, the configuration algorithm determines the LBDR bits to be changed in order to support failure patterns. As shown later on, one and two link failure patterns are very efficiently supported by this method.

Fig. 4 shows the steps followed to obtain the configuration algorithm. The figure can be compared with Fig. 3 where the steps to compute LBDR bits were depicted. As a first step, with the first stage, a set of initial paths are computed for a fault-free mesh network. For instance, Fig. 5.(a) shows the result of applying the SR algorithm to a 4×4 mesh network. Although there are many instances that can be derived from the SR algorithm, we segment the network and assign the restrictions in a zig zag manner from top to bottom. This method has proved to provide good performance and alternative paths [13].

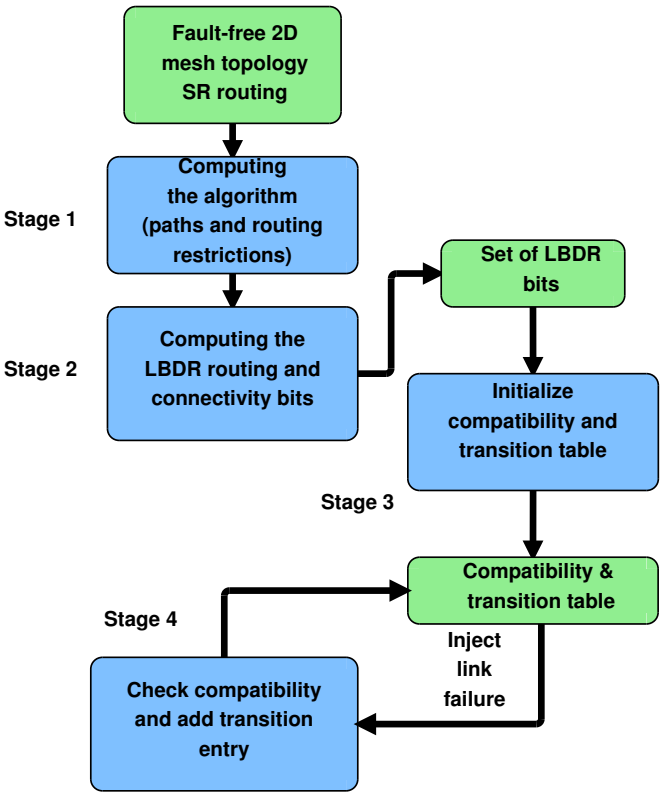
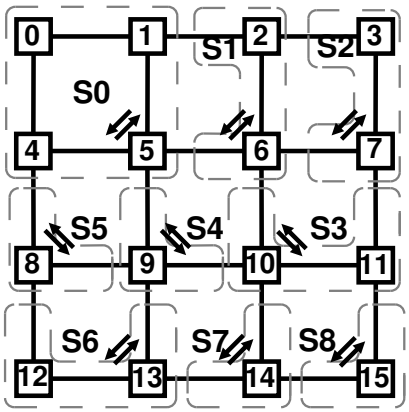


Fig. 4. Computation process for the configuration algorithm.

Once the set of routing restrictions (represented by arrows in Fig. 5.(a)) are obtained, the LBDR configuration bits, located on every switch, are computed in the second stage. Indeed, those bits can be deduced in a straightforward way from the location of the routing restrictions and the connectivity of the network. As an example, Fig. 5.(b) shows the bits for the topology. These bits form the basic configuration of the switches for our base system. Notice that those bits are set at design time, before any failure is detected. Notice also that no deroutes are needed in the original fault-free configuration.

Assuming the previous routing configuration, and once the system boots, we need to identify, the real topology. In order to keep the normal NoC operation, when a topology change is detected, the SR routing restrictions must be reviewed, and the LBDR configuration bits updated, if necessary. Fig. 6 shows a configuration with the failure of the link connecting switches 1 and 5. As can be seen, the restriction in the segment including the failure has been removed, and this implies that some LBDR bits need to be changed. Moreover, at switch 5 a deroute option needs to be added to restore connectivity with switch 1.

Therefore, we face two important challenges when detecting a topology change. The first one is computing the new configuration bits that need to be changed. This is quite challenging as it is not straightforward to understand how a routing algorithm should transition from an initial topology to the real one. Indeed, the most challenging aspect is how to restore connectivity and guaranteeing deadlock freedom. The second



(a)

Router	Cn	Ce	Cw	Cs	Rne	Rnw	Ren	Res	Rwn	Rws	Rse	Rsw
0	0	1	0	1	0	0	0	1	0	0	1	0
1	0	1	1	1	0	0	0	1	0	1	1	0
2	0	1	1	1	0	0	0	1	0	1	1	0
3	0	0	1	1	0	0	0	1	0	1	0	0
4	1	1	0	1	1	0	0	1	0	0	0	0
5	1	1	1	1	1	1	0	1	1	1	0	1
6	1	1	1	1	1	1	0	1	1	1	0	1
7	1	0	1	1	1	1	0	1	1	1	0	1
8	1	1	0	1	1	0	1	1	0	0	1	0
9	1	1	1	1	1	1	1	1	0	1	1	0
10	1	1	1	1	1	1	1	1	0	1	1	0
11	1	0	1	1	1	1	0	1	0	1	0	0
12	1	1	0	0	1	0	0	0	0	0	0	0
13	1	1	1	0	1	1	0	0	1	0	0	0
14	1	1	1	0	1	1	0	0	1	0	0	0
15	1	0	1	0	1	1	0	0	1	0	0	0

(b)

Fig. 5. (a) Segment-based routing with zig-zag restrictions combination and (b) Routing and connectivity bits for the SR algorithm on a 4x4 2D mesh (deroutes are not necessary).

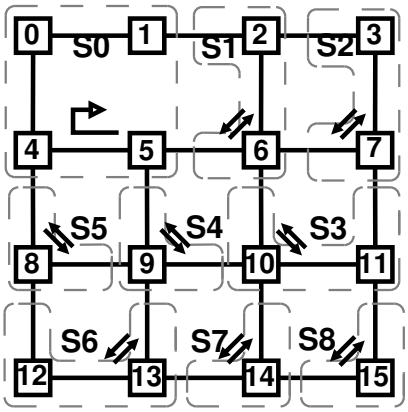


Fig. 6. Deroute option configuration.

challenge is how to perform such operation in a fast manner, minimizing processing time and required resources amount. In the next sections we address both challenges when assuming the use of the SR algorithm and the LBDR implementation. Both are addressed in Stage 4 of the computation process of the algorithm shown in Fig. 4.

4.1. Deadlock and Connectivity

The SR routing algorithm works by segmenting the network (links and switches) into different disjoint segments and applying a bidirectional routing restriction on every segment. This guarantees deadlock-freedom and connectivity. However, it is not clear how such algorithm should evolve if suddenly a link fails (changing the topology) while still guaranteeing deadlock-freedom and connectivity. This is the core problem we face in this paper.

Fortunately, the provided SR algorithm has been analyzed and a key property has been found. Indeed, in this paper we deduce the SR algorithm is able to keep connectivity among all the end nodes through the network whenever a maximum of a link failure is found per segment. To achieve such property, the routing restriction in the segment with a failed component has to be removed. Indeed, the network still provides connectivity in a network segmented with S segments in a link failure pattern with as much as S failed links, each failed link isolated in a segment.

In [14] it is demonstrated that on every cycle of the network at least one segment is fully included in the cycle. A segment is formed by a trail of links and switches. Therefore, by enforcing a bidirectional routing restriction on every segment it is guaranteed the cycle is broken, and thus, SR is deadlock-free. Moreover, in [14] it is demonstrated that the connectivity of the network is guaranteed. The reasoning behind this is that both ends of a segment are connected with a path not crossing the segment. That is, both ends are connected outside the segment.

These two properties allow us to derive a valid deadlock-free routing algorithm that supports a failure on every segment[2]. Upon detecting a link failure, the algorithm basically removes the routing restriction initially placed in the segment with the failed link. This simple action keeps deadlock-freedom and connectivity property untouched. Indeed, the algorithm can remove as many restrictions as failed links, only restricted by the fact that failures affect at maximum one link per segment. Fig. 7 shows an example with three missing links but with a compatible routing algorithm. Notice that three routing restrictions have been eliminated, all belonging to the affected segments.

Demonstration of deadlock freedom is achieved by contradiction. Imagine a topology with a failed link. Based on the SR demonstration, we can still hold that every cycle includes at least one entire segment. If the failed link does not affect that

2. It is worth mentioning that two link failures in the same segment break the segmentation procedure of SR and thus connectivity is not guaranteed. In this case, we rely on a new segmentation process (applying the SR algorithm again) on the real topology, thus leading to a different but valid set of routing restrictions. The algorithm in this case becomes complex and thus requires a high computation time.

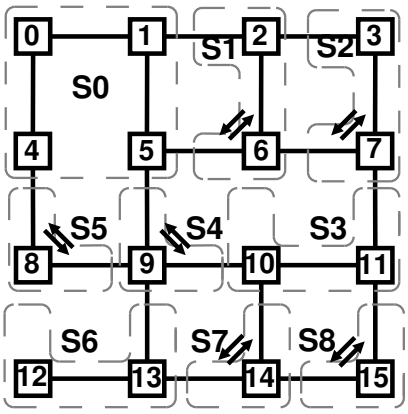


Fig. 7. Three missing links with a compatible routing algorithm.

cycle then the cycle is still avoided by the routing restriction of the segment. If, however, the failed link affects the segment, then we break the cycle with the failed link. Indeed, there is no cycle. Moreover, the routing restriction is removed as the failed link enforces packets to not cross the segment. Therefore, the algorithm is deadlock-free.

Connectivity is demonstrated similarly as in SR. Imagine a segment formed by the following four switches (links are omitted) $s_a - s_b - s_c - s_d$, and there is a routing restriction in the segment at switch s_c. Connectivity is guaranteed between s_a and s_d out of the segment. If a failure affects the segment, then connectivity is still guaranteed between switches s_a and s_d outside the affected segment. As the routing restriction is removed, switches within the segment can connect to one of the end switches and thus, still keeping the connectivity.

4.2. Fast Configuration Algorithm

With the previous method, we are ready to compute the new LBDR configuration bits to support failure patterns. Indeed, the initial LBDR bits need to be updated. Moreover, all the failure combinations lead to the need of implementing non-minimal path support. Fortunately, this situation is supported in LBDR with the so-called deroute bits that allow enforcing non-minimal routes. For instance, in Fig. 6 switch 5 needs a deroute option to allow messages from switch 5 to switch 1 to reach their destination.

As commented previously, failure patterns with more than one failed link per segment need a new segmentation process. In this section, however, we focus our attention in providing a fast and effective method for all the failure combinations compatible with the algorithm. Moreover, we focus on one-link and two-link failure patterns. In summary, when a link failure occurs the algorithm is able to remove the SR restriction of the segment that presents the link failure and configure the LBDR deroute bits in the required switches in order to comply with the new algorithm. This is a manual process that happens in stages 3 and 4 (Fig. 4).

At stage 3, two small tables are initialized: compatibility and transition tables. The compatibility table will indicate the combination of link failures that are compatible. A combination is compatible if the configuration algorithm supports that combination. The transition table will set the LBDR bits that need to be changed for every possible failed link. These two tables are filled at stage 4 by injecting link failures and checking their compatibility and computing the LBDR bits that need to be modified. In order to cope with the extreme complexity and the large amount of combinations, we have developed a tool that tests all the new configuration bits that need to be changed at each failure combination. Note that this process is still performed at design time and in advance to any possible failure detection. The goal is to use the tables precomputed later at run time and in the face of a failed link notification.

In Fig. 8 we list all the configuration bits that need to be changed in order to support all the one link failure patterns (24 cases). Since there are 24 links in a 4x4 mesh, we have 24 different possibilities of one link fails. For this reason, we have labeled the combinations in the figure from F0 to F23. Moreover, for each case we show the affected link (by the corresponding switch IDs and the output ports) and the bits that will be changed. For instance, for case F0 (east link of switch 0) the connectivity bit C_e at switch 0 will be set to zero, the C_w bit at switch 1 will be also reset. For routing bits, the R_{en} bit at switch 4 is set and the R_{sw} bit is also set (as the routing restriction is removed). Moreover, a deroute bit is set at both switches 0 and 1 in order to take the south direction. The case represented in Fig. 6 corresponds to the fail F4 in Fig. 8.

Based on the *one failure per segment* property of the algorithm we optimize the configuration process for one and two failure cases in a simple and straightforward way. It consists of applying the algorithm for one link failure two times, each one focused on each failure.

FAIL	SW	OP	SW	OP	BITS TO BE CHANGED				
F0	0	E	1	W	Ce[0]=0	Cw[1]=0	Ren[4]=1	Rsw[1]=1	DR[0,1]=S
F1	1	E	2	W	Ce[1]=0	Cw[2]=0	Ren[5]=1	Rsw[2]=1	DR[1,2]=S
F2	2	E	3	W	Ce[2]=0	Cw[3]=0	Ren[6]=1	Rsw[3]=1	DR[2,3]=S
F3	0	S	4	N	Cs[0]=0	Cn[4]=0	Ren[4]=1	Rsw[1]=1	DR[0,4]=E
F4	1	S	5	N	Cs[1]=0	Cn[5]=0	Ren[4]=1	Rsw[1]=1	DR[1,5]=W
F5	2	S	6	N	Cs[2]=0	Cn[6]=0	Ren[5]=1	Rsw[2]=1	DR[2,6]=W
F6	3	S	7	N	Cs[3]=0	Cn[7]=0	Ren[6]=1	Rsw[3]=1	DR[3,7]=W
F7	4	E	5	W	Ce[4]=0	Cw[5]=0	Ren[4]=1	Rsw[1]=1	DR[4,5]=N
F8	5	E	6	W	Ce[5]=0	Cw[6]=0	Ren[5]=1	Rsw[2]=1	DR[4,5,6]=N, Ree[4]=0
F9	6	E	7	W	Ce[6]=0	Cw[7]=0	Ren[6]=1	Rsw[3]=1	DR[5,6,7]=N, Ree[5,6]=0
F10	4	S	8	N	Cs[4]=0	Cn[8]=0	Rse[4]=1	Rwn[9]=1	DR[4,8]=E
F11	5	S	9	N	Cs[5]=0	Cn[9]=0	Rse[5]=1	Rwn[10]=1	DR[5,9]=E
F12	6	S	10	N	Cs[6]=0	Cn[10]=0	Rse[6]=1	Rwn[11]=1	DR[6,10]=E
F13	7	S	11	N	Cs[7]=0	Cn[11]=0	Rse[6]=1	Rwn[11]=1	DR[7,11,3]=W, Rww[11]=1, Rss[3]=0
F14	8	E	9	W	Ce[8]=0	Cw[9]=0	Rse[4]=1	Rwn[9]=1	DR[8,9,10,11]=N, Rww[10,11]=0
F15	9	E	10	W	Ce[9]=0	Cw[10]=0	Rse[5]=1	Rwn[10]=1	DR[9,10,11]=N, Rww[11]=0
F16	10	E	11	W	Ce[10]=0	Cw[11]=0	Rse[6]=1	Rwn[11]=1	DR[10,11]=N
F17	8	S	12	N	Cs[8]=0	Cn[12]=0	Ren[12]=1	Rsw[9]=1	DR[8,4]=E, DR[12]=N, Rss[4]=0
F18	9	S	13	N	Cs[9]=0	Cn[13]=0	Ren[12]=1	Rsw[9]=1	DR[9,13]=W
F19	10	S	14	N	Cs[10]=0	Cn[14]=0	Ren[13]=1	Rsw[10]=1	DR[10,14]=W
F20	11	S	15	N	Cs[11]=0	Cn[15]=0	Ren[14]=1	Rsw[11]=1	DR[11,15]=W
F21	12	E	13	W	Ce[12]=0	Cw[13]=0	Ren[12]=1	Rsw[9]=1	DR[12,13]=N, Ree[12]=0
F22	13	E	14	W	Ce[13]=0	Cw[14]=0	Ren[13]=1	Rsw[10]=1	DR[12,13,14]=N, Ree[13,14]=0
F23	14	E	15	W	Ce[14]=0	Cw[15]=0	Ren[14]=1	Rsw[11]=1	DR[13,14,15]=N

Fig. 8. LBDR bits to be changed on fault detection.

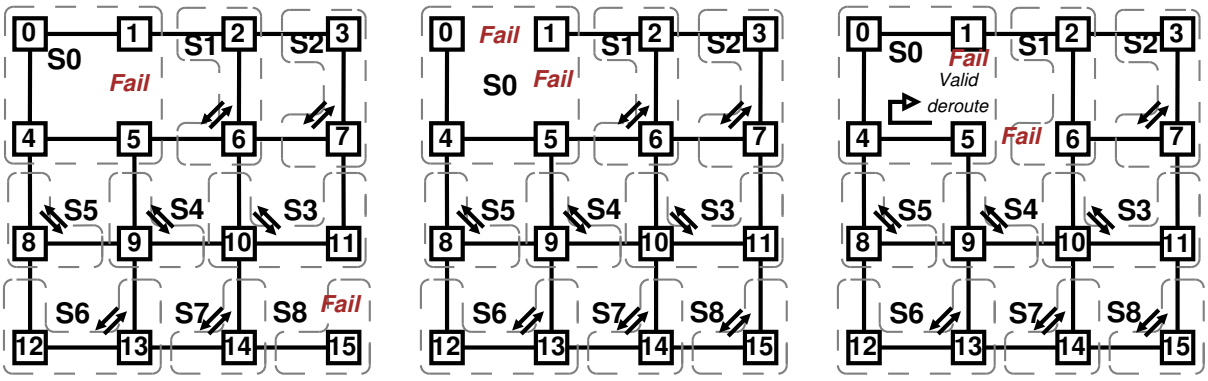


Fig. 9. Examples of link failure combinations: (a) Supported (2FC), (b) unsupported (2FI), and (c) special case (2FS).

Given a 2D-mesh network, the number of different topologies with two link failures is very high. For example, in a 4×4 mesh network there are 24 different topologies with one failed link, but 276 different topologies with two failed links. This number grows exponentially with network size. We have carefully analyzed all the combinations of one and two failed links. We classify all the combinations in four large sets: 1F set, 2FC set, 2FI set, and 2FS set. 1F refers to all the combinations with only one failed link. All these combinations are supported by the algorithm. The set 2FC refers to all the patterns with 2 link failures where the two failures are *compatible*. We say two failed links are compatible if the algorithm can treat them separately. These cases are also supported. 2FI means combinations that have two failed links and the two failed links are incompatible among them. In this case, there is not support for them. Two failed links are incompatible if the addition of individual actions by the algorithm does not apply (this is the case of two failed links in the same segment). Finally, 2FS represents the case of 2 failed links that are special, in the sense that the algorithm can add new changes in order to make both failed links compatible. Figures. 6, 9.(a), 9.(b), and 9.(c) show an example of each set, respectively.

The number of combinations having the two link failures in the same segment depends on the number of segments and the length of segments used (e.g., in a 4×4 mesh the number of segments is 9, ending up in 18 combinations). However, note that, some combinations belonging to the 2FS set can still be supported. An example is depicted in Fig. 10 where there are two failed links in the S0 segment. This specific case is fully supported by the LBDR mechanism. We will see in Section 5.1 the coverage for different topologies, where the unsupported combinations represent a minimal percentage.

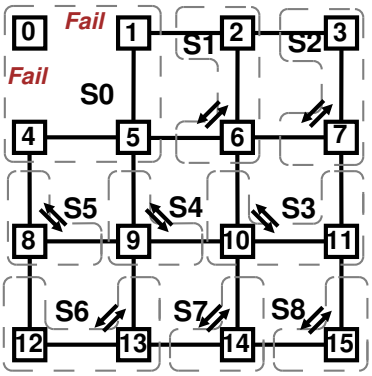


Fig. 10. Supported although the faults are in the same segment

Support for the 2FS set is enforced by extra deroute options placed in specific switches. Fig. 9.(c) shows an example where the links connecting switches 1-5 and 5-6 failed. As a consequence of the failure of the link connecting switches 1-5, the switch 5 needs a *west* deroute. Moreover, the failure of the link connecting switches 5-6 would also need a *north* deroute at switch 5. In such a situation we have a special case: although the second failure forces a *north* deroute in switch 5, it is not a valid deroute because the north link of switch 5 also failed. For this specific case, the solution is to configure a *west* deroute at switch 5. For this reason, this combination needs an extra-change that must be covered by our configuration algorithm.

For all the failure patterns, the tool uses a compatibility table indexed by the failed links. The table is of $L \times L$ being L the number of links in the network. Each entry in the compatibility table has a code that identifies the set the combination belongs to (0 for 2FI, 1 for 1F, 2 for 2FC, and 3 for 2FS). In the case of the 2FS set the algorithm needs to allocate further deroutes to support the combination. The diagonal of the table is set by the 1F cases.

Once all the possible link failures are computed and checked (stage 4 in Fig. 4), the tables are ready for their use at the central controller at run time. The configuration algorithm is depicted in Fig. 11. When a topology change is detected and one/two failures occur, the compatibility matrix is checked. If the algorithm finds the corresponding entry labeled as 0 the topology is not supported and the configuration process would need to trigger the slow configuration process that recomputes a new SR algorithm from scratch (60% failure coverage is achieved, as shown in [9], see Fig. 3). If the entry is

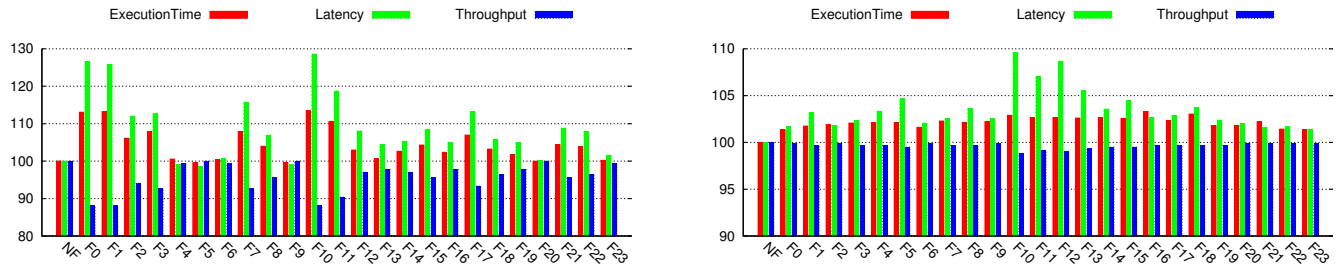


Fig. 12. Performance degradation for (a) BC application and (b) SC application.

labeled as 1 or 2 then the algorithm applies the corresponding changes (located in the transition table). Finally, if the entry is labeled as 3, the algorithm applies the appropriate changes as if the entry was 2, plus the extra changes (located in an extra transition table).

5. Experimental Results

In this section we show the configuration algorithm coverage for all the possible combinations from one up to eight link failures. Then, we evaluate the performance degradation for all the combinations when both one link and two links failures occur. Moreover, we show the configuration time for the different sets supported by the algorithm. Finally, we analyze how the algorithm could be extended to a larger set of link failures.

5.1. Coverage and Performance of the Configuration Algorithm

As we have already mentioned in the introduction section, our algorithm offers full support for all the combinations with one link failure. In the case of two link failures, there are several topologies that are not supported, which occurs when the two failures coincide in the same SR segment. TABLE 1 shows the coverage for all the combinations of two failures.

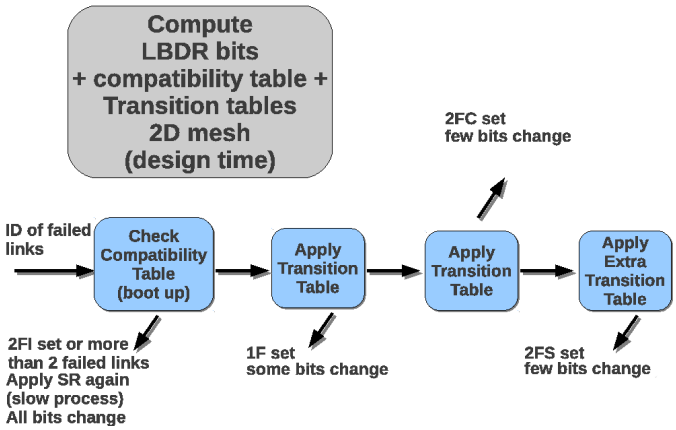


Fig. 11. Configuration algorithm.

TABLE 1. Coverage percentage for two link failures.

Mesh topology	Supported 2FC+2FS	Unsupported 2FI	% Coverage
4x4	282	18	93.5
5x5	791	29	96.3
6x6	1788	42	97.6
7x7	3513	57	98.4
8x8	6254	74	98.8
9x9	10347	93	99.1
10x10	16176	114	99.3

As can be seen, for all the topologies checked the coverage is very high and only a few combinations are not supported. As the network size increases, the coverage also increases reaching 99.3% for 10x10 mesh. This is due to the fact that the percentage of failed combinations that end up with segments with more than one failed link is lower. The larger the network the higher the number of segments in the network and, thus, the lower the likelihood of two failures hitting the same segment.

In the following we present performance results for all the topologies with just 1 failure[3] compared with the fully connected 4x4 mesh (we called this topology as NF). We plot the execution time, the average network latency, and the average network throughput.

We use a full-system simulator based on Simics-GEMS [11], [12]. Regarding the on-chip network, we use a mesh topology which has all the links among switches of the same width. Specifically, input port buffers are 4-flit deep, and packets are 8-flit long for control messages and 72-flit long for data messages. Flit size is set to one byte.

As workload, we have used the PARSEC v2.1 benchmark suite [15]. This benchmark suite contains emerging applications and commercial solutions that cover a wide area of working sets and allows current CMP technologies to be studied more effectively. Although we have used all the applications from the PARSEC v2.1 benchmark, we only show here results for five applications: Blackscholes (BC), Streamcluster (SC), Swaptions (SW), Bodytrack (BT), Fluidanimate (FL), and x264 (X). In all cases, Simsmall input set has been used.

3. Since there are 24 links in a 4x4 mesh, we have 24 different topologies, each one with one link failure that is labeled in the figures from F0 to F23

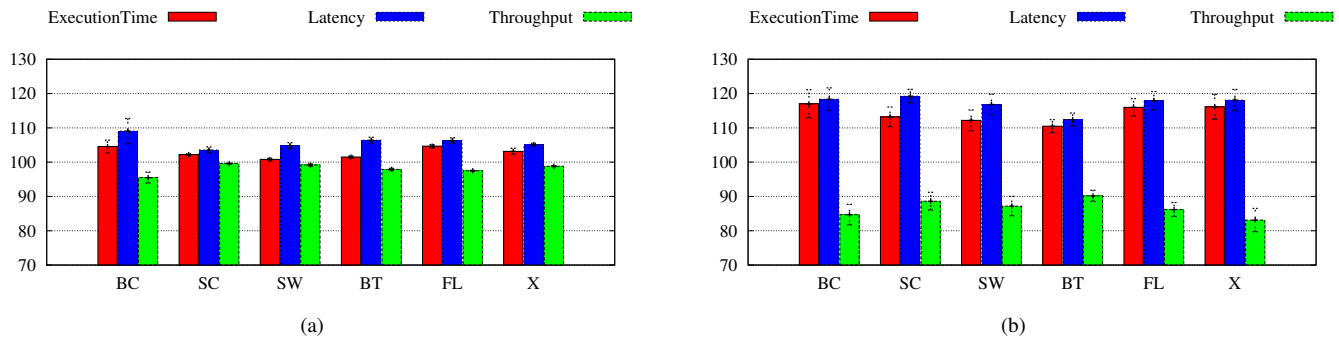


Fig. 13. Average results for (a) all the 1F combinations and (b) 2FC + 2FS combinations.

Figures 12.(a) and 12.(b) show detailed results for all the evaluated topologies. In these figures all the results are shown in normalized terms with respect to the NF topology. The x-axis depicts the NF topology (a fully connected 4x4 mesh without link failures), and all the combinations of 1 link failure (from F0 to F23). As we can see, the performance degradation is minimal for all the cases. The obtained results are influenced by both the communication traffic pattern and the injected traffic of each application during the execution. Specifically, for the Streamcluster application the results are very stable, while the Blackscholes application presents more differences among the topologies.

Fig. 13 shows results for all the selected PARSEC applications (x-axis) for all the supported topologies of 1 fail (Fig. 13.(a)) and 2 failures (Fig. 13.(b)). Each value shown in the figures is the average of different simulations varying the topology. The figures also show the performance variation of the different topologies represented by the confidence interval (which has been set to 95%). As we can see, the values are very stable for the 1F combinations and no topology exceeds 16% of execution time overhead for the executed applications. Regarding 2 failures, obviously the degradation and the variation is higher, but no topology exceeds 25% of execution time overhead.

Finally, Fig. 14 shows the configuration time for the different sets supported by the algorithm. The algorithm was coded in C and executed on the instruction set simulator of an ARM7 processor core. For 1F sets the number of processor cycles is significantly low (186 cycles). For 2FC and 2FS cases, we can see a logical increment, however, execution time in cycles is always lower than 350 cycles. With a 700MHz operating frequency assumed for the processor core running the global controller, this means that the configuration time is lower than 0.497 microseconds, which is quite very low.

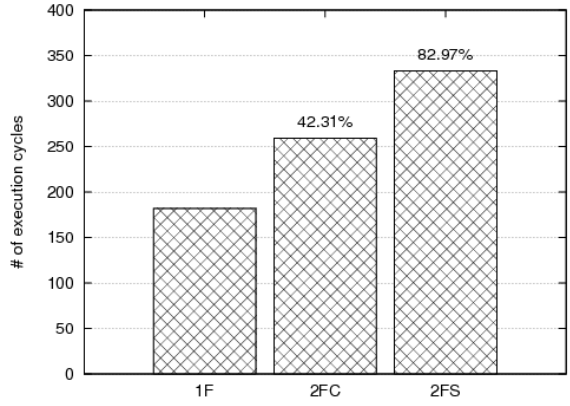


Fig. 14. Execution cycles count.

5.2. Beyond Two Link Failures

As we have seen in the previous sections, the configuration algorithm is able to deal successfully with 1- and 2-link failures. In this final evaluation section we analyze how the algorithm could be extended to a larger set of link failures. Indeed, we have checked the potential of the property of SR that allows one link failure per segment. TABLE 2 shows the coverage for 3-failed links by dealing them in a sequential manner and by using the same transition tables as used for the 2-failed links. 3FC and 3FS mean supported cases whereas 3FI means unsupported ones. The uncovered cases correspond to combinations where two failed links lie in the same SR segment. As we can see, percentage keeps low for those cases. To solve them, as said before, we should rely on a complete and new re-segmentation process of the topology.

However, for those cases that are not covered by the algorithm we can also re-adapt the topology and remove some components for the sake of reverting the case to a supported one. Indeed, many topology combinations (due to failed patterns) can get simplified when disabling components (e.g. lamb nodes) and grouping failed links into square regions. All the components in the square are disabled and the irregularity of the topology is reduced. By doing this, many of the

TABLE 2. Coverage percentage for three link failures.

Topology mesh	Supported 3FC+3FS	Unsupported 3FI	% Coverage
4x4	1640	424	79.5
5x5	8798	1148	88.4
6x6	31756	2504	92.7
7x7	90638	4768	95.0
8x8	219812	8264	96.4
9x9	474174	13364	97.3
10x10	935608	20488	97.9

topology combinations that currently are unsupported will be successfully handled. One example can be seen in Fig. 9.(b) where nodes 0 and 1 could simply be removed. Now, most of the network is regular and SR can be applied successfully using the same segmentation process shown in the figure. This is, however, left for future work.

Finally, we show in Fig. 15 a more extensive characterization of coverage of the configuration algorithm with multiple link failures for topologies that varies from 4x4 to 10x10 mesh size. As can be seen, the coverage of our configuration algorithm is very high. As the network size increases, the coverage also increases reaching 82.1% for 8 failures in a 10x10 mesh size. This is due to the fact that the percentage of failed combinations that end up with segments with more than one failed link is lower. Therefore, the larger the network the higher the number of segments in the network and, thus, the lower the likelihood of two failures hitting the same segment.

Finally, future work concerns techniques that embed failed links into rectangular regions to be compatible with current SR routing segmentation, and therefore, maximize the coverage.

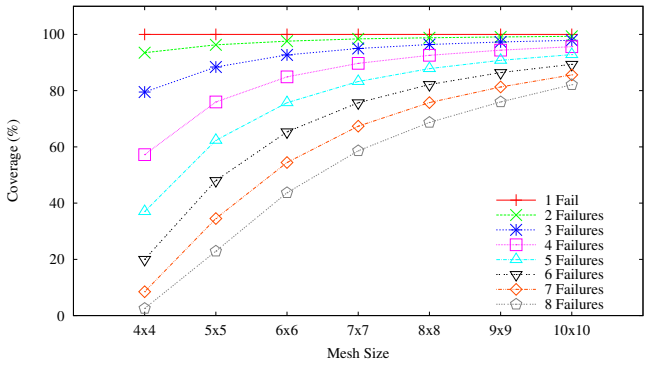


Fig. 15. Up to 8 failures.

6. Conclusions

This paper proposes a fast centralized computation routing algorithm for self-configuring NoC infrastructures, capable of reducing the configuration time (less than 1 microsecond) while maintains a high coverage support (up to 99.3% for two link failures in a 10x10 mesh). The key idea consists of using a centralized controller for efficient routing reprogramming.

The proposed configuration algorithm has been optimized to minimize the overhead in resources and to handle the most common failure cases in a very fast time, although retaining the possibility to handle adverse failure patterns in a much longer configuration time (through re-segmentation of the SR algorithm, which can take in the order of seconds to execute).

Experimental results show the effectiveness of the proposed configuration algorithm in terms of configuration time and coverage. The configuration algorithm is evaluated for all the combinations of both 1 and 2 failure patterns on a virtual platform with cycle accuracy using real applications.

Acknowledgements

This work was supported by the Spanish MEC and MICINN, as well as European Comission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. It was also partly supported by JCCM under Grants PCC08-0078-9856 and POII10-0289-3724, and by the project NaNoC (project label 248972) which is funded by the European Commission within the FP7 Research Programme. We thank Dr. Davide Bertozzi for discussions and contributions to this paper and special thank to Dr. Daniele Ludovici for providing the synthesizer for evaluate the configuration time.

References

[1] M. E. Gomez, N. A. Nordbotten, J. Flich, P. Lopez, A. Robles, J. Duato, T. Skeie, O. Lysne, "A Routing Methodology for Achieving Fault Tolerance in Direct Networks", IEEE Transactions on Computers, Volume 55 Issue 4, April 2006.

[2] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, D. Blaauw, "A highly Resilient Routing Algorithm for Fault-tolerant NoCs", Proceedings of Design Automation and Test in Europe (DATE), 2009.

[3] E. Beigne, F. Clermidy, P. V. A. Couard, M. Renaudin, "An asynchronous NoC Architecture providing Low Latency Service and Its multi-level Design Framework", Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems, pp. 54–63, 2005.

[4] D. Starobinski, M. Karpovsky, L. A. Zakrevski, "Application of Network Calculus to General Topologies using Turn-Prohibition", IEEE/ACM Transactions on Networking, vol. 11, no. 3, pp 411–421.

[5] M. Ali, M. Welzl, S. Hellebrand, "A Dynamic Routing Mechanism for Network-on-Chip", Proceedings of the 23rd NORCHIP Conference, pp. 70–73, 2005.

[6] F. Angiolini, D. Atienza, S. Murali, L. Benini, G. De Micheli, "Reliability Support for On-Chip Memories Usign Networks-on-Chip", Proceedings of ICCD, 2006.

[7] A. Alaghi, M. Sedghi, N. Karimi, M. Fathy, Z. Navabi "Reliable NoC Architecture Utilizing a Robust Rerouting Algorithm", Proceedings of the East-West Design and Test Conference.

[8] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, J. Duato, "Efficient Implementation of Distributed Routing Algorithms for NoCs", IET-CDT, pp.460-475, vol.3, issue 5, 2009.

[9] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, J. Duato, "Addressing Manufacturing Challenges with Cost-Effective Fault Tolerant Routing", Proc of NOCS 2010, pp.35-32, 2010.

[10] N. Hornarmand, A. Shahabi, Z. Navabi, "A Heuristic Search Algorithm for Re-routing of On-Chip Networks in the Presence of Faulty Links and Switches" Proc. of IEEE EWDTS, 2007.

[11] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[12] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's genera execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[13] Andres Mejia Gomez, "Design and Implementation of Efficient Topology Agnostic Routing Algorithms for Interconnection Networks", PhD dissertation, University of Valencia, 2008.

[14] A. Mejia, J. Flich, and J. Duato, "On the Potentials of Segment-Based Routing for NoCs," in *Proceedings of the 2008 37th International Conference on Parallel Processing (ICPP) '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 594–603.

[15] C. Bienia and K. Li, "PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[16] A. Mejia, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, May 3-6, 2010.