

Essential Fault-Tolerance Metrics for NoC Infrastructures

Cristian Grecu¹, Lorena Anghel², Partha P. Pande³, André Ivanov¹, Resve Saleh¹

¹University of British Columbia, ²TIMA Laboratory, ³Washington State University

¹{grecuc,ivanov,res}@ece.ubc.ca, ²lorena.anghel@imag.fr, ³pande@eecs.wsu.edu

Abstract

Fault-tolerant design of Network-on-chip communication architectures requires the addressing of issues pertaining to different elements described at different levels of design abstraction – these may be specific to architecture, interconnection, communication and application issues. Assessing the effectiveness of a particular fault-tolerant implementation can be a challenging task for designers, constrained with tight system performance specifications and other requirements. In this paper, we provide a top-down view of fault-tolerance methods for NoC infrastructures, and present a range of metrics used for estimating their quality. We illustrate the use of these metrics by simulating a few simple but realistic fault-tolerant scenarios.

1. Introduction

Global on-chip interconnects are becoming a serious bottleneck for meeting performance and power consumption requirements of complex, multi-processor System-on-Chip (MP-SoC). The problem of global interconnects is recognized as a challenging design constraint both in industry and academia. Among the proposed solutions, the use of packet-based on-chip interconnection networks, commonly referred to as network-on-chip (NoC) [1] [2], is a promising alternative that addresses the issues of increasing interconnect complexity. The NoC design paradigm addresses the ever-increasing delay and power dissipation of global on-chip interconnects by offering structured interconnect fabrics consisting of wire segments and intelligent routing blocks that ensure a certain level of Quality of Service (QoS) [3] [4]. The QoS represents a collection of requirements on the NoC performance expressed by the achievable throughput (bandwidth), latency, power dissipation, and reliability.

The major cause affecting the reliability of the VLSI global interconnects is the shrinking of the feature size, which exposes them to different faults of permanent, transient or intermittent nature. Among the failure mechanisms, we can enumerate factors such as crosstalk, electromigration, electromagnetic interference, alpha particle hits, and cosmic radiation. These phenomena can alter the timing and functionality of the NoC fabrics and thus degrade their QoS characteristics or, eventually, lead to failures of the whole NoC-based system. Providing resilience from such faults is mandatory for the operation of NoC-based chips.

Traditionally, error detection and correction mechanisms are used to protect communication subsystems against the effects of transient malfunctions. Designers must carefully weigh the hardware cost of implementing such mechanisms for the on-chip data communication infrastructures against the potential benefits they can bring [5] [6]. Complex error

detection and correction (EDC) schemes may require additional energy dissipation and area overhead, and can affect the performance of SoC communication architectures in terms of throughput and latency.

Other approaches for fault-tolerant, on-chip communication include stochastic communication, adaptive routing, and different hybrid schemes that combine spatial and temporal redundancy for achieving fault tolerance.

Standard metrics for fault-tolerant systems are well-established and have been used extensively in design of distributed computing systems. Recent research in fault-tolerant NoC fabrics proposed metrics that are specific to message-passing on-chip communication systems for evaluating the effectiveness of new and existing fault-tolerance methods. In the absence of widely-accepted metrics, it is difficult for NoC designers to assess the fault-tolerant capabilities in a quantitative manner.

One option for evaluating the fault tolerance (FT) of a system is to measure the degree of hardware and software redundancy (both spatial and temporal), which is an inherent property of most NoC architectures. While redundancy by itself is a useful measure, it is incomplete in the sense that different systems can exploit redundancy in more or less efficient ways. It is therefore preferable to have metrics that can measure the effective fault tolerance as it influences the required performance in the main task of transporting information across the NoC fabric. Based on the above considerations, the scope of this paper is to present traditional FT metrics in the novel context of NoC systems, and use them in conjunction with newly-proposed metrics in order to measure the *effective fault tolerance* in the context of overall system performance for NoC subsystems.

The rest of this paper is organized as follows. In Section II we first present a review of fault-tolerant methods for NoCs and the performance figures used to assess their usefulness. Section III presents a hierarchical view of fault tolerance in NoC architectures and defines a set of metrics that we consider relevant with respect to performance of FT implementations. In Section IV we present some experiments that illustrate the utility of the proposed metrics. Section V presents the concluding remarks.

2. Related work

The quest for on-chip fault-tolerant communication started well before the NoC paradigm emerged as a solution for integrating large MP-SoCs. Solutions for fault-tolerant on-chip busses are found in [7], where a particular form of duplication is used to detect and correct crosstalk and transient faults, and [8], where different error recovery mechanisms from simple retransmission to correction/retransmission are analyzed in terms of power/area figures.

The NoC infrastructures are characterized by more complex topologies (relative to bus-based topologies), with higher degrees of connectivity. Moreover, in the NoC paradigm, data is transferred across the chip by employing some form of packet switching, where sent data is tagged with additional flow control information [16]. The additional hardware and/or information redundancy offers significant potential for implementing different fault-tolerant strategies. Hardware redundancy can be exploited by using multiple paths to transport data between source and destination cores, e.g., in the case of adaptive routing methods [9], or stochastic communication [10] [11]. In [10], the total time to complete the application (a Fast Fourier Transform algorithm) in presence of faults is another performance measure of the FT method.

General-purpose metrics used in [10] and [11] are *energy consumption* and *area cost*, which are fully usable and relevant.

In practice, temporal and spatial data redundancy is often used to achieve the fault-tolerant goals. Ultimately, designers must assess the effectiveness of the FT implementation in the context of the NoC performance specification. This information must be readily available and quantifiable such that, if a specific FT implementation cannot be accommodated while still meeting the performance specification of the NoC medium, it can be eliminated from the set of potential FT realizations at early stages of the design process.

Most of the previously published works on the issue of FT in NoCs present methods for achieving FT at the algorithm or circuit level, and then proceed to assess them relative to an *ad-hoc* set of parameters that may be more or less relevant to the specifics of an on-chip data transport mechanism.

In [10] and [11], the authors propose probabilistic communication schemes based on probabilistic broadcast and random walk, respectively. In these schemes, multiple copies of the same message are transmitted following different paths, selected randomly from the set of possible routes between communicating pairs of cores. Eventually, from the set of redundant messages, one that is error-free may reach the destination, completing a successful transmission. *Message latency* is correctly identified in both works as an important measure for evaluating the impact of the FT solution. However, there are many factors that can affect message latency, other than the specific FT implementation, such as traffic distribution (spatial and temporal), buffer size and buffer management, flow control, etc. The contributions of all these factors towards message latency are generally inter-dependent, and it is quite difficult to separate their individual effects.

In [18] a new metric to characterize fault-tolerant NoCs is proposed, called *message arrival probability* (MAP). For a pair of tasks (processes) that exchange messages across the NoC, MAP is defined as the fraction of successfully transmitted messages. This metric is useful for assessing NoC performance when tight bounds are imposed on its capability of transmitting error-free messages.

A different category of FT metrics relates to topology characteristics that can potentially offer FT properties. In particular, *connectivity* is used to express the ability of an interconnection network to offer multiple paths among its

communicating nodes [17]. This figure of merit is useful, but only when combined with the flow control mechanism that can exploit the degree of connectivity in a more or less efficient manner.

When dedicated spare routers or links are employed for achieving fault tolerance, the amount of spare elements can give an estimate of the NoC's FT capabilities. As for the connectivity, the *amount of spares* is only offering information about the NoC's potential fault tolerance, rather than its actual ability.

Various coding schemes were proposed for providing error resilience to on-chip communication infrastructures [6] [12]. A comprehensive comparison and evaluation is carried out in [13] for detection-only and single-error correcting codes, combined with retransmission for data recovery. The effect of locating the recovery points in the NoC topology (e.g., end-to-end or switch-to-switch recovery) is also studied and evaluated. The metrics used in [12] [13] quantify the effect of error recovery schemes on NoC parameters such as data latency, area overhead, and energy consumption, under specific conditions of NoC operation.

These types of approaches for determining the efficiency of FT schemes based on their effect on NoC performance parameters, under specific operating conditions (traffic distributions), have the drawback that they only deliver information for a set of inter-dependent operating conditions (i.e., a given NoC hardware instance, specific traffic distributions and flow control protocols).

2.1 Existing FT metrics

Traditional engineering methods that address design of fault-tolerant systems use reliability and availability analysis to characterize these systems and their components. *Reliability* is defined as the probability with which a system will perform its task without failure under specified environmental conditions over a specified time duration. Thus, the MTBF (Mean Time Between Failures) parameter is used as a representation of the average time to the next failure, and is calculated as:

$$MTBF = \frac{\text{Total operating time}}{\text{No. of failures encountered}} \quad (1)$$

Another metric used for evaluation of fault-tolerant systems with repair/recovery capabilities is MTTR (Mean Time To Repair), defined as:

$$MTTR = \frac{\text{Time spent for repairs}}{\text{No. of repairs}} \quad (2)$$

Based on MTBF and MTTR, the *availability* of a system can be used to measure the impact of failures on an application, and is defined as:

$$\text{Availability} = \frac{MTBF}{MTBF + MTTR} \cdot 100\% \quad (3)$$

While useful at system level, these metrics may overlook important properties of fault-tolerant NoC subsystems. One such property that is misrepresented by use of MTBF is the capability of a NoC medium to rapidly recover from failures. Even in the case when the number of failures is high (which indicates a low, undesired MTBF), if the recovery can be performed quickly (e.g., through flit-level recovery [13] [14]), the impact of failures may be minimal and, therefore, it may

not affect the application at all. For the same reason, the availability of the NoC subsystem in such condition is misrepresented by Eqn. (3).

Another drawback of these generic metrics is that they represent average values. In the case of NoC fabrics that must meet tight quality of service (QoS) requirements in the presence of failures, the average values are not useful since the performance constraints (in terms of guaranteed latency per message or available throughput) have to be met for *all* possible instances, not only on an average basis.

While there is little doubt that fault tolerance is a desirable and useful property of NoCs, designers need simple, readily available, and self-sufficient metrics to be able to characterize FT methods in the context of the specific NoC implementation. We introduce these metrics relative to the NoC's ability to *detect* the occurrence of faults and *recover* from failures. The metrics we address in this work aim to address the issue of characterizing the effectiveness of fault-tolerance schemes for NoC communication subsystems in the context of the specific QoS requirements that designers face in their implementation. They are not intended to substitute the existing metrics, but to complement them by offering a more detailed view of the properties of different fault-tolerance methods. An exhaustive analysis of the classic fault-tolerance metrics is outside the scope of this paper. Instead, we choose to illustrate how the metrics defined here can help designers gain more insight on the actual performance of fault-tolerant implementations related to NoC architectures.

3. Fault-tolerance metrics for NoC subsystems

Before defining the set of metrics forming the objective of this work, we need to differentiate between different hierarchical abstraction levels for achieving resilience to failures. There are five key elements in a comprehensive approach to fault-tolerant design: *avoidance*, *detection*, *containment*, *isolation*, and *recovery*. Ideally, these are implemented in a modular, hierarchical design, encompassing an integrated combination of hardware and software techniques. Moreover, the fault-tolerant techniques can be applied at different layers from the set of ISO/OSI layers [15] that the NoC may implement, resulting in numerous possibilities for fine tuning the performance of the FT implementation by combining the (sub) set of FT elements with the (sub) set of NoC layers. As an example, error detection may be implemented in the data layer, and recovery may be realized either in the data layer (e.g., if an error correcting code is used) or at the application layer. In a more generic approach, the partitioning and derivation of requirements, and the partitioning and implementation of fault/failure management techniques must be realized in a hierarchical fashion. For each hierarchical level, the existence of appropriate metrics allows the designers to have full control and understanding of the implications that a particular fault-tolerant implementation will have on the operation of a NoC subsystem.

For illustrating the set of metrics discussed in this work and their usability, we consider a simple example of an application running on a NoC-based multi-processing system. The application employs two processes P_1 and P_2 on two different processing cores, as shown in Fig. 1. Processes P_1 and P_2 use

the NoC subsystem to communicate with each other along the path shaded in grey.

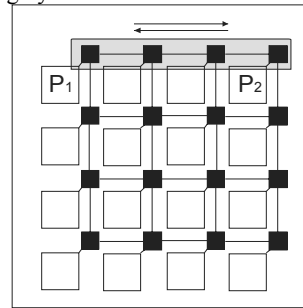


Figure 1: Processes communicating across a NoC fabric.

Using a layered representation of the data communication in a NoC-based system, and considering a subset of the standard OSI layers, Fig. 2 shows the propagation of faults from the physical level (faults affecting low level device functionality) to the application level (faults affecting the software application running on the NoC-based system). At each level in the hierarchy, faults can be characterized by type, source, frequency of occurrence, and impact. At the lowest level (physical), it is assumed that a fault results in a degraded operation of the respective component. If it is not always cost effective to detect and recover at the lowest level, the fault manifests itself as a local error/failure which propagates to the next higher level, where the corresponding FT technique is evaluated again relative to performance and cost effectiveness. The hierarchical approach can provide back-up at higher levels for faults which, for any reason, are not handled at lower levels. Generally, the higher the level in the hierarchy, the longer it takes to contain and/or recover from the effect of a failure, but there are certain advantages with respect to area cost and power dissipation. For real-time NoC systems, time is the critical factor for specifying the performance of the FT implementation. Designers must decide on the effectiveness of a particular method by knowing how quickly faults must be detected, how quickly they have to recover from the occurrence of a fault, how long an error can exist in the NoC infrastructure without impairing/compromising system performance.

Based on the example application in Fig. 1 and considering a hierarchical implementation as in Fig. 2, we define metrics for the five elements of comprehensive fault-tolerant methods.

(a) Avoidance

Fault avoidance techniques can be realized through information redundancy (by means of error correcting codes for NoCs) or hardware redundancy (n -modular redundancy being a typical example).

Depending on the targeted number of errors to correct, coding/decoding hardware blocks may require a certain number of cycles to perform their operation. The associated time overhead adds to the total latency of the data being transported across the NoC fabric.

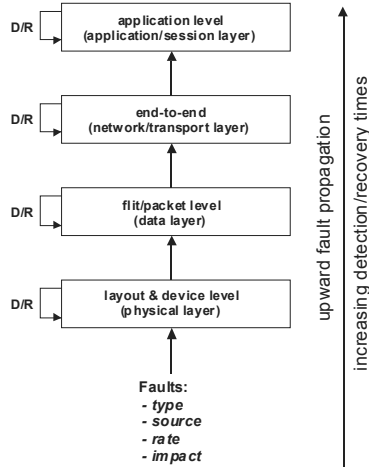


Figure 2: Hierarchical partitioning for fault-tolerant NoC designs.

We define $T_{av, ov}$ as the time overhead of an avoidance scheme, and compute it as the difference between data latency with (Lat_{av}) and without (Lat) fault avoidance:

$$T_{av, ov} = Lat_{av} - Lat$$

The difference between various implementations of this concept can be significant relative to this metric. In the example in Fig. 1, if coding/decoding functions are implemented at each switch on the path between P_1 and P_2 (switch-to-switch avoidance), the resulting time overhead will be significantly higher than in the case where only end-to-end avoidance is implemented (i.e., data is encoded at the source and decoded at destination).

(b) Detection

The next hierarchical level in a fault-tolerant design is the detection of faults that were not handled by the avoidance mechanism. Detection is built in most error-correcting codes, which generally can provide information regarding the number of un-corrected faults, when the correction mechanism fails (or is not even present in the particular implementation). Fault detection is then used to assess the need for recovery from potentially uncorrected fatal failures. The quicker the detection mechanism signals the presence of un-corrected faults, the quicker the recovery can be initiated. We define the detection latency T_{lat} as the amount of time between the moment a fault occurs and the moment it is detected. Going back to our example in Fig. 1, fault detection may be performed by the processes P_1 and P_2 whenever data is received (end-to-end detection), at the input ports of the intermediate switches (switch-to-switch detection), or at each switch input/output port (code-disjoint detection).

(c) Containment

Fault containment is concerned with limiting the impact of a fault to a well-defined region within the NoC. Error containment refers to avoiding the propagation of the consequences of a fault, the error, out of this defined region. Fault containment regions (FCR) may be defined with variable resolutions, directly correlated with the quality and resolution of the fault detection mechanism. For the case of Fig. 1, and assuming an end-to-end detection mechanism, the fault containment region can be defined as the entire shaded route between the cores where processes P_1 and P_2 are being

executed. It is essential that FCR are independent, in the sense that a fault occurring in a FCR does not affect a different FCR. In this respect, if two routes between cores/processes P_1 and P_2 can be found that are on independent FCRs, and a fault is detected on one of the routes, the other route can be used to provide an alternative path between processes P_1 and P_2 .

(d) Isolation

The independency of fault containment regions can only be achieved if an effective isolation method can be provided, which can guarantee that the effect of a fault occurring in a FCR does not propagate to another FCR. At the physical layer, in the case of permanent faults, isolation can be accomplished by marking or disconnecting the faulty NoC components (links, switches, routes) and avoiding their use until, eventually, hardware recovery/repair can be performed through reconfiguration. At higher layers, erroneous data packets can be dropped on the fly or at the destination process, such that they are not allowed to interfere with the rest of the data and propagate at application level.

(e) Recovery

The ultimate goal of fault-tolerant schemes is to provide means to recover from occurrence of failures. For fault-tolerant and QoS constrained NoCs, it is important to recover from failures within the time budget allowed by the QoS specifications. Late recoveries, even when successful, are not acceptable, since they lead to out-of-specification behavior of the NoC subsystem. Consequently, we define *recovery time* (T_{rec}) as the amount of time that passes between the detection of a fault and recovery from the corresponding failure. A simple form of attempting recovery of erroneous data flowing between processes P_1 and P_2 is to provide a hardware correction at lower layers, or a retransmission mechanism at higher layers where, upon detection of an error, an automated retransmission request (ARQ) is generated and an error-free copy of original data is re-sent from the source process.

4. Experiments and discussion

We illustrate the use of the metrics defined in this work by running simulations of a few detection and recovery schemes on a 4x4 NoC fabric as the one in Fig. 1. A cycle-accurate, flit-level simulator is used, with messages injected with equal probabilities by all NoC cores (i.e., uniformly distributed random traffic). Messages are organized in 16 flits each, and are transferred across the NoC medium following the dimension-order (*e-cube*) routing algorithm, where messages are first sent along the *x* direction of the topology, then along the *y* direction towards the destination core. The injection rate is varied uniformly. Faults are injected randomly, on a flit-per-cycle basis, i.e., in each cycle, flits may be marked as erroneous with equal probabilities, regardless of their current location in the NoC fabric

In this set of experiments, we are mainly interested in detection and recovery estimation. We consider the following cases for fault detection scenarios:

- end-to-end (*e2e*) detection: the errors are detected at the destination cores. Each flit is checked for errors upon reception and, if found erroneous, the detection latency is calculated as the difference between the index of the fault injection cycle and the index of the reception cycle.

- switch-to-switch (*s2s*) detection: error checking is performed at each input port of the switches, for each flit traveling across the NoC.

- code-disjoint (*cdd*) detection: flits are checked for errors at both input and output ports of the NoC switches.

We simulate two recovery methods, both based on retransmission of erroneous messages. The first scheme is a message-level, equal-priority retransmission. Upon detection of an erroneous flit, an ARQ message is generated at the destination core. If more flits in a message are erroneous, only one ARQ message is generated. The ARQ is sent towards the source of the erroneous message and handled as any other regular message by the NoC environment. We do not consider here the case where ARQ messages themselves are affected by errors. There are three types of messages in this scheme: regular messages, ARQ messages, and retransmitted messages. They all have equal priorities, and are treated similarly when traversing the NoC. We call this method equal priority recovery (*epr*).

For speeding up the recovery process, we consider a second retransmission-based recovery scheme, where ARQ and retransmitted messages have higher priority than the regular messages. When contention arises, regular messages will wait for ARQ and retransmitted messages to be processed. This allows faster transport of the latter message types, and therefore speeds-up the recovery process. We call this method priority-based recovery (*pbr*).

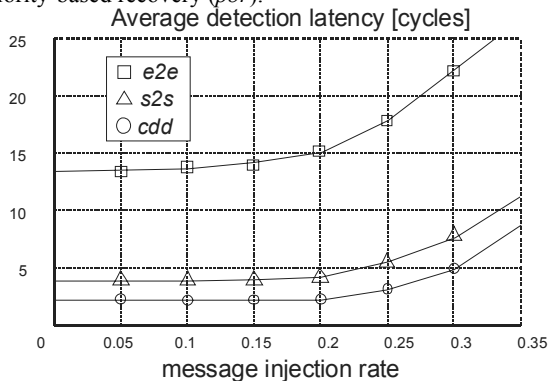


Figure 3: Average detection latency for end-to-end (*e2e*), switch-to-switch (*s2s*), and code-disjoint detection (*cdd*).

Fig. 3 shows the average detection latency for the three detection mechanisms considered here. Flit error rate is set at $e=10^{-10}$, i.e., in each cycle, for each flit, the probability of being flagged as erroneous is 10^{-10} . The *e2e* detection is the poorest performer relative to detection latency, since the messages must travel all the way from source to destination before an eventual error is detected. The *cdd* mechanism performs best, due to the fact that error detection is performed at each input/output of the switches.

The second experiment measured the recovery latency for a few combinations of detection/recovery schemes. Ideally, we would like to isolate the detection and recovery mechanisms, and report individual performance figures for recovery phase only. In practice, this is difficult due to the two processes being inter-dependent. In our experiments, the inter-dependency is caused by the need to manage the erroneous messages in a realistic fashion, i.e., upon detection of an erroneous flit, the corresponding message continues to

be transported toward its destination and, eventually, interact with ARQ and retransmitted messages.

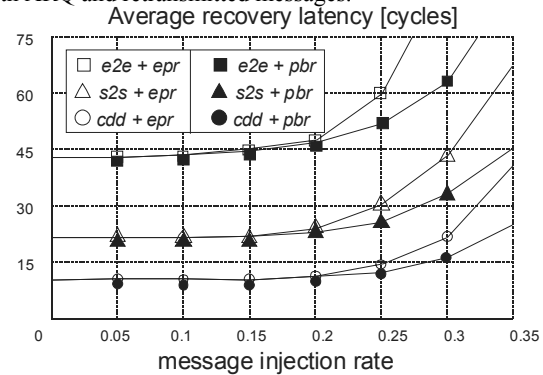


Figure 4: Average recovery latency for equal priority (*epr*) and priority based (*pbr*) recovery methods.

The average recovery latencies for the two recovery methods considered in our experiments are shown in Fig. 4. The flit error rate is set to $e=10^{-10}$ for all simulation runs. Note that the *pbr* method offers better recovery latency due to preferential processing of ARQ and retransmitted messages. The advantage of priority based recovery is more apparent at higher injection rates, where the ARQ and retransmitted messages are able to travel faster than the regular messages waiting in queue buffers for available resources. Also, the *pbr* scheme can support tighter QoS requirements in terms of minimum latency in presence of errors.

In the last experiment, we study the effect of varying the flit error rate e on the average recovery latency for the *pbr* scheme. In this experiment, the message injection rate was set constant to a non-saturating value of 0.1 flits/cycle/core. A high flit error rate will cause the generation of numerous ARQ and retransmitted messages, which will effectively cause an increase of the NoC traffic. Therefore, we ran the simulations at a low, non-saturating injection rate, at which the variation of message recovery latency is principally affected by the error rate e . The results shown in Fig. 5 indicate that the system can reliably recover from failures at flit error rates up to approximately $e=10^{-9}$.

The advantage of using the detection and recovery latencies as defined in this work becomes apparent by observing that using experiments similar to the ones presented here, designers are able to isolate the performance figures of fault-tolerant methods from the environmental conditions (represented by traffic conditions in our experiments).

As an example, the design specification of a NoC fabric such as the one in Fig. 1 may require an average recovery latency of 30 cycles for a message injection rate of up to 0.1 flits/cycle/core, at a flit error rate of up to $e=10^{-10}$.

Based on Fig. 4 and 5, the end-to-end (*e2e*) schemes can be eliminated from the set of potential candidates for FT implementations, since their average recovery latency is greater than the required value of 30 cycles.

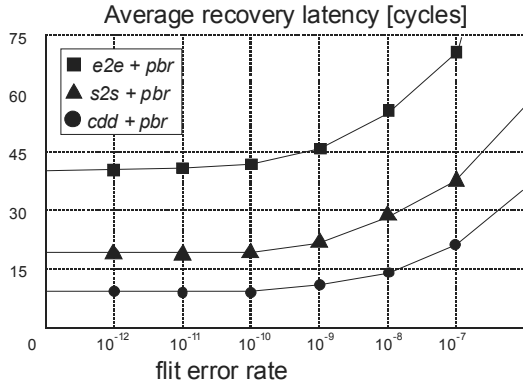


Figure 5: Average recovery latency for *pbr* scheme with variable flit error rate.

Finally, we estimate the message arrival probability (MAP) associated with the three detection scenarios for the pair of communicating processes P_1 and P_2 . In Fig. 6, we plot the average message latency versus bound on MAP under traffic assumptions stated earlier in this section. In this set of experiments, the flit error rate was fixed at $e=10^{-6}$, at a message injection rate close to network saturation.

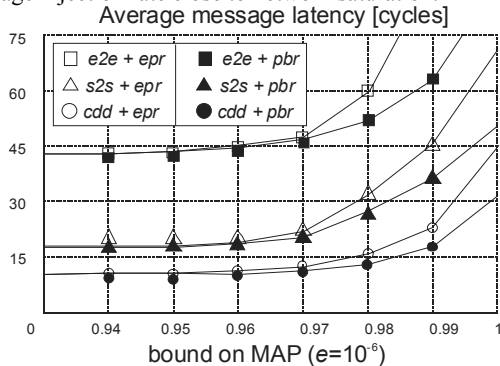


Figure 6: Average message latency vs. bound on MAP

Note that when the required MAP is increasing towards 1, the average message latency increases significantly. This indicates that, when the application requires a MAP close to 1, the simple recovery techniques presented here may not suffice, and more advanced FT techniques need to be employed. Also, a complete characterization of particular FT techniques would involve additional application metrics under application-specific traffic conditions.

5. Conclusions

The need for fault-tolerant features in NoC communication fabrics is recognized by academic and industrial communities, and different implementations were proposed to address the challenges of providing fault-tolerant communication in NoC subsystems. It is, however, difficult for designers to choose a particular fault-tolerant scheme due to lack of metrics that can express the intrinsic effectiveness of a FT method, and not only its effect on generic performance parameters such as latency, throughput, and power consumption.

We have presented a unified view of fault-tolerant methods for NoC subsystems, with hierarchical partitioning that makes possible mapping the elements of a FT implementation on the generic layered structure of network-based communication

systems. We presented metrics that assess the capability of a NoC subsystem to quickly detect the presence of a fault and recover from failures in a speedy manner. The use of these metrics was illustrated with a set of experiments.

6. References

- [1] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, Jan. 2002, pp. 70-78.
- [2] P. Pande, C. Grecu, A. Ivanov, R. Saleh, G. De Micheli, "Design, Synthesis and Test of Networks on Chip", *IEEE Design and Test of Computers*, Vol. 22, No. 5, 2005, pp. 404-413.
- [3] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "QNoC: QoS architecture and design process for Network on Chip", Special issue on Networks on Chip, *The Journal of Systems Architecture*, Dec. 2003
- [4] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, P. Wielage, "An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming", *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, 2004, Vol: 2, pp: 878- 883.
- [5] D. Bertozzi, L. Benini, G. De Micheli, "Error Control Schemes for On-Chip Communication Links: The Energy-Reliability Tradeoff", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 6, June 2005, pp. 818-831.
- [6] S. R. Sridhara, and N. R. Shanbhag, "Coding for System-on-Chip Networks: A Unified Framework", *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, Vol. 13, No. 6, June 2005, pp. 655-667.
- [7] M. Lajolo, "Bus guardians: an effective solution for online detection and correction of faults affecting system-on-chip buses", *IEEE Transactions on VLSI Systems*, Vol. 9, Issue: 6, Dec. 2001, pp: 974-982.
- [8] D. Bertozzi, L. Benini, G. De Micheli, "Low power error resilient encoding for on-chip data buses", *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 4-8 March 2002, pp: 102-109.
- [9] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks - An Engineering Approach*, Morgan Kaufmann, 2002.
- [10] T. Dumitras, S. Kerner, and R. Marculescu, "Towards on-chip fault-tolerant communication", in *Proceedings of ASP-DAC*, Jan. 2003. pp: 225-232.
- [11] M. Pirretti, G. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, "Fault Tolerant Algorithms for Network-On-Chip Interconnect", *Proceedings of IEEE ISVLSI 2004*, pp: 46-51.
- [12] P. P. Pande, A. Ganguly, B. Feero, B. Belzer, C. Grecu, "Design of low power and reliable networks on chip through joint crosstalk avoidance and forward error correction coding", *Proceedings of IEEE DFT Symposium, DFT'06*, 2006, pp:466-476.
- [13] S. Murali, T. Theodorides, N. Vijaykrishnan, M. J. Irwin, L. Benini, G. De Micheli, "Analysis of error recovery schemes for networks on chips", *IEEE Design and Test of Computers*, Sept.-Oct. 2005, Vol: 22, Issue: 5, pp: 434- 442.
- [14] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, P. P. Pande, "On-line fault detection and location for NoC interconnects," *Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS'06)*, 2006, pp. 145-150.
- [15] International Standards Organization, Open Systems Interconnection (OSI) Standard 35.100, www.iso.org.
- [16] A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, Kluwer Academic Publishers, 2003.
- [17] D. Ferrero, C. Padró, "Connectivity and fault-tolerance of hyperdigraphs", *Discrete Applied Mathematics* 117 (2002), pp: 15-26.
- [18] S. Manolache, P. Eles, Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC", *Proceedings of Design Automation Conference, DAC 2005*, pp: 266 - 269.