Blocking and Non-blocking Checkpointing and Rollback Recovery for Networks-on-Chip

Claudia Rusu[1], Cristian Grecu[2], Lorena Anghel[1]

[1] *TIMA Laboratory (CNRS-UJF-INPG),* [2] *University of British Columbia*

claudia.rusu@imag.fr, grecuc@ece.ubc.ca, lorena.anghel@imag.fr

Abstract

In this paper we propose a dynamically reconfigurable failure recovery scheme developed for Network-on-Chip (NoC) based systems. The recovery scheme is based on a checkpointing and rollback protocol and permits enhancing the system fault tolerance capabilities by exploiting information on traffic load and failure rate. The increased performance of the fault tolerance mechanism is achieved by simply switching between blocking and non-blocking checkpointing approaches. We analyze the effectiveness of the solution, considering different traffic loads and expected failure rates.

Keywords: network-on-chip, fault tolerance, rollback recovery, checkpoint, performance, failure rate.

1. Introduction

The use of packet-switched on-chip interconnection networks, commonly referred to as networks-on-chip [1] is a promising alternative that addresses the issues of increasing inter-core communication requirements. Thus, NoC is emerging as the new on-chip communication structure, where applications can be seen as sets of different communicating tasks running on several processing elements (PEs). Achieving fault tolerance of large, multi-core designs is recognized as a challenging design constraint both in industry and academia.

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 fault sources we can enumerate: process variation, crosstalk, electromagnetic interference and cosmic radiation hits. These phenomena can alter the timing and functionality of the NoC fabrics and thus degrade their characteristics, leading, eventually, to failures of the NoC-based systems. A failure, even if it only affects an element of the system, can lead to restarting the entire application from its initial state, which can be unacceptable, especially for critical applications.

Rollback recovery has been largely employed as a fault tolerant mechanism at system and application level, in distributed and parallel systems and can also serve a similar purpose in NoC-based systems. Figure 1 illustrates the rollback recovery principle vs. restart.

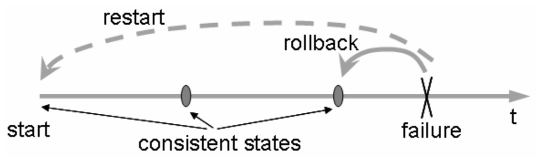


Figure 1. Rollback recovery vs. restart

The mechanism implies periodically saving (*checkpointing*), during the failure-free execution, of consistent states, on *stable storage*. The latter is a memory element whose content persists through the tolerated failures. A state is consistent if it could be reached following a correct and error-free execution of the application from the start point. In case of failure, a rollback is performed, i.e. the application resumes execution from the most recent consistent state.

Several approaches of rollback recovery have been proposed [2]. In distributed systems, pure uncoordinated checkpointing suffers from domino effect [3], caused by low probability to form a consistent state by individual checkpoints (it is assumed that each individual system forming a distributed system is able to take checkpoint). To avoid the domino effect, this method is usually complemented by message logging, at the price of higher overhead. Due to its simplicity and lower overhead when compared to other methods, coordinated checkpointing is preferable in practice [2]. Since these features are not only desirable, but rather required for systems-on-chip (SoC), we will use in this work the coordinated checkpointing.

During the failure-free execution, when taking a global checkpoint using the coordinated protocol, all tasks must synchronize with each other, in order to set a consistent global state. The coordinated checkpointing principle is depicted in Figure 2.

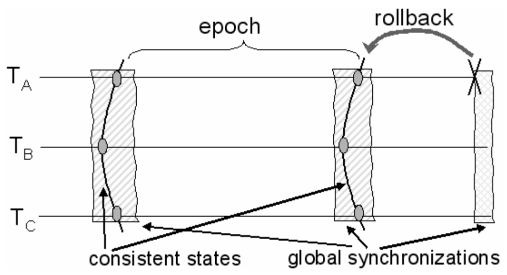


Figure 2. Coordinated checkpointing principle

The global synchronization coordinator is a dedicated task, which we call the recovery management unit (RMU).

During the global synchronization phases, synchronization messages are exchanged among tasks. Both blocking and non-blocking checkpointing methods were proposed. In the blocking approach, tasks block their normal execution during checkpointing and exchange only synchronization messages, while in the non-blocking one, tasks overlap their normal execution with the synchronization phase. Generally, the non-blocking approach is preferred due to its lower latency. Thus, there are many works dedicated to proposing new or improved non-blocking protocols for distributed and parallel systems.

For typical SoC applications, the nature of the communication pattern is mostly heterogeneous, with certain task sets exchanging data more frequently than others. Different applications with their own disjoint task sets can also run simultaneously on the NoC-based system [4]. Even if the tasks in different disjoint sets do not exchange data with each other, they may compete for NoC resources (routers, channels). Mapping the application tasks to PEs is an important step in the NoC design flow, and is generally performed with the objective of minimizing communication latency and power dissipation [5]. Also, it is desirable to adapt the checkpoint and rollback recovery method to the specifics of the communication pattern, and optimize the recovery mechanisms such that the latency and overhead they incur are reduced as much as possible.

One of the main limitations of the coordinated checkpointing protocol is the duration of the global synchronization. With the increase of the network size and/or traffic load, the checkpointing duration increases. As the protocol allows no overlapping of checkpointing phases, the possible interval of time between two consecutive checkpoints enlarges. In consequence, after a rollback in case of failure, re-bringing the application to the failure-free state incurs greater latency.

This situation becomes even more drastic when the synchronization latency cannot be reduced below the expected interval between two consecutive failures. In such cases, there is no sufficient time to take a new global checkpoint, so the application always rolls back to the last one. When the probability of occurrence of these conditions is high, the application stops advancing its execution.

In this work we show that this kind of situations are better solved using a blocking checkpointing approach. The main contribution of this work is the development of a checkpointing protocol that can run in either blocking or non-blocking mode, for NoC based systems. It can dynamically change its mode exploiting information on traffic load and failure rate in order to obtain an optimum performance. Moreover, as the same protocol is used, it offers the possibility to block only the execution of a subset of participating tasks, while the others checkpoint without blocking (useful for critical tasks or tasks that do not significantly overload the NoC resources).

The rest of the paper is organized as follows. In Section 2 we present related works regarding NoC fault tolerance and blocking/non-blocking checkpointing approaches. Section 3 describes the system and application model. Section 4 presents the checkpoint-rollback recovery scheme. Simulation results are presented in Section 5. Section 6 contains the concluding remarks and future work.

2. Related Work

Using the common layered representation of distributed networked systems, we can classify the currently proposed fault tolerant schemes for NoC based systems depending on the layer(s) onto which they are placed in the communication stack. At the lowest level we find fault tolerant approaches based on hardware redundancy (NMR – N modular redundancy) [6], where NoC links are duplicated.

At higher levels we find solutions based on error detection and correction, eventually combined with retransmission [7] [8]. Some of the coding-based solutions consider crosstalk as an important source of data corruption, and devise crosstalk-avoidance methods, sometimes combined with single or multiple error correction mechanisms [9]. These methods can be applied either inter-router or to end-to-end channels.

A few adaptive routing techniques were proposed for avoiding permanent faults in NoC components (links or routers). Adaptive routing has the disadvantage of not being able to handle transient errors, therefore must be combined with error detection/correction schemes when transient errors are a concern. A solution that can cope with transient errors without a need for error control schemes is stochastic communication [10], where data is forwarded from the source to destination cores on multiple paths, selected probabilistically, such that the probability of a message being received correctly can be significantly improved. Stochastic methods are, however, resource intensive and tend to saturate the network quickly due to multiple copies of the same message being transported through the network simultaneously.

Failure recovery through checkpointing and rollback complements the above mentioned fault tolerant methods by adding an extra level of safety when a failure occurs and it is not solved by any of the methods at lower levels.

Blocking [15] or non-blocking [17] checkpointing protocols were previously proposed for computer networks and super-computers. Different approaches were proposed to reduce the overhead of coordinated checkpointing algorithms, like minimizing the number of synchronization messages and the number of checkpoints, making the checkpoint non-blocking or combinations of these [11]. A quantitative comparison between two different blocking and non-blocking protocols for large scale MPI (message passing interface) systems was addressed in [16].

Another direction for enhancing the overall checkpointing performance is the communication-induced checkpointing (CIC). CIC protocols were proposed as an improved approach combining the coordinated and the uncoordinated checkpointing protocols. They attempt to reduce the number of tasks participating in the global checkpointing by a dynamical analysis of the traffic pattern. However, a more recent work [12] shows that CIC protocols do not scale well for larger number of tasks. Hierarchical checkpointing protocols have also been proposed for large clusters and cluster federations using coordinated checkpointing or combining coordinated checkpointing and CIC [13].

3. System and Application Model

In this work, the NoC-based multi-core system-on-chip is modeled as a set of PEs interconnected by a message-passing communication fabric. At any time, one or several computing tasks can be mapped onto each processing element. During execution, the tasks exchange data in form of packets using a wormhole switching technique, with messages organized in sets of basic flow control units (flits). The first flit of each message (the *header* flit) establishes the path between the corresponding source and destination PEs, and the subsequent *data* flits follow the header in a pipelined fashion. The last flit of the message, the *tail* flit, ends the transmission and frees the resources reserved by the header. As for routing, the XY algorithm is used.

For simulations we used a mesh direct-connected NoC, as the one depicted in Figure 3, with one task mapped onto each PE.

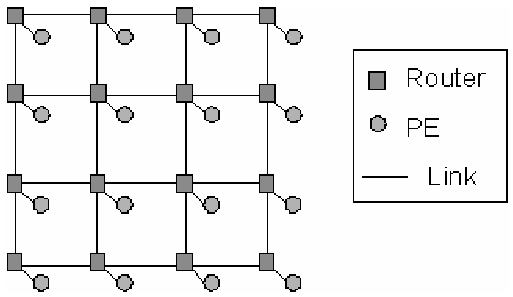


Figure 3. 4x4 mesh direct NoC

The stable storage is local to each PE, so the checkpoints do not overload the network when stored to the stable storage.

4. Recovery Scheme

4.1. Principle and Characteristics

In this sub-section we present the coordinated checkpointing protocol we developed and used in this work for NoCs.

We base our approach on classical protocols [14] [15] and stress some features required by systems-on-chip. In our protocol, any task can initiate a checkpoint, but the coordination is done each time by the same unique dedicated task (the RMU). Also, all rollback requests pass by the RMU. Having a unique RMU for a communicating task group presents the advantage of simplicity, lower traffic overhead and lower costs in terms of area and power consumption, as detailed subsequently. One of the advantages is that the checkpoint validation by broadcast can be avoided, reducing thus the traffic overload. Instead, a task knows whether its last checkpoint is valid or not upon the receipt of either a CK_REQ (checkpoint request message) or a rollback request, when the RMU also communicates the index of the respective checkpoint.

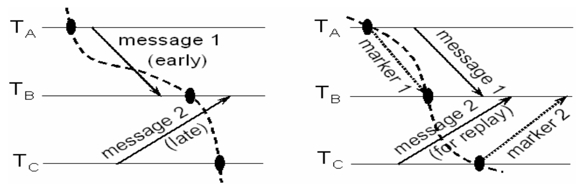
Besides, the unique RMU approach helps simplifying the protocol. It avoids the synchronization needed when several RMUs start a checkpointing phase, which also reduces the associated overload in the communication network. Moreover, passing rollback requests by the unique RMU avoids the situation when, in case of rollback request from a task, some tasks have already received from the former RMU the validation of the latest checkpoint and others have not. Finally, having only a RMU in a communicating PE group implies less hardware and less power compared with the situation when more (or all) PEs have to run an RMU task.

The proposed failure recovery mechanism tolerates single and multiple failures. In addition, several failures can be treated in a single recovery phase if they occur relatively close in time, i.e., if the time interval between two failures is comparable to the recovery protocol duration. Thus, for higher failure rates, treating several failures in the same recovery phase contributes to reducing the overall latency, since a rollback to an old checkpoint is performed only once instead of several times.

The coordinated protocol we developed requires little extra information to be stored by each task and does not use timestamps [2], therefore it can be used for both synchronous and GALS (Globally Asynchronous, Locally Synchronous) NoCs.

4.2. Consistent State. Early and Late Messages

Global inconsistent states (the dashed curve in Figure 4a) can appear as there may be messages flowing in the NoC when different task checkpoints are taken. These inconsistent states are avoided by treating *early* (appear as received, but never sent) and *late* (appear as sent, but not received) messages. Markers exchanged among application tasks serve either to avoid early messages (marker 1) or to end the log of late messages (marker 2), as depicted in Figure 4b.



a) Inconsistent global state with early and late messages

b) Consistent global state with forced checkpointing and replay list

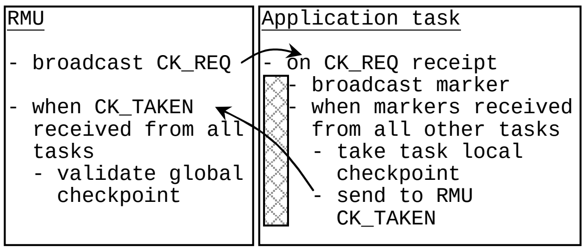
Figure 4. Inconsistent and consistent states using markers

Late messages are logged at the destination together with the task state and are to be replayed from this log after rollback. Thus, a task is considered to have taken its checkpoint only after both its state and all the late messages from all other tasks are on stable storage. Markers can be explicit dedicated messages or can be piggybacked on application messages (called checkpoint sequence numbers, CSNs).

4.3. Blocking/Non-blocking Protocol

Figure 5 presents the blocking/non-blocking protocol we developed for taking a coordinated checkpoint. CSN carried by application messages [14] are used, complemented with explicit markers [15]. Explicit markers contribute in significantly reducing the checkpoint duration when the application traffic (carrying CSNs) is scarce.

The RMU broadcasts a checkpoint request, CK_REQ, to the application tasks. Then, these exchange markers in order to establish a consistent state. After its local checkpoint has been taken, each task informs the RMU about this, by sending a CK_TAKEN message. Subsequently, the RMU can validate the global checkpoint, which is formed by the set of local checkpoints, after the CK_TAKEN receipt from all tasks. When a task knows its new local checkpoint is globally validated, it can remove its previous checkpoint from stable storage, so at any moment at most two checkpoints must be stored.



```
RMU                          Application task

- broadcast CK_REQ           - on CK_REQ receipt
                               - broadcast marker
- when CK_TAKEN                - when markers received
  received from all              from all other tasks
  tasks                          - take task local
  - validate global                checkpoint
    checkpoint                   - send to RMU
                                   CK_TAKEN
```

Figure 5. Coordinated checkpointing protocol

In the non-blocking approach, tasks continue normal execution during the checkpointing phase. The same sequence of actions presented above is executed for checkpointing when the blocking protocol is used, except that the task execution is blocked between the CK_REQ receipt and the CK_TAKEN send, as depicted in Figure 5. However, during the checkpointing, late messages are logged and will be part of the checkpoint. So, regardless of the blocking or non-blocking nature of the checkpoint, the protocol is the same and the checkpoint itself has the same content. Thus, each task can participate to the global checkpointing either by blocking or not its normal execution, regardless of the other tasks.

5. Simulations and Evaluation

This section presents the results we obtained by simulation using the blocking (B) and non-blocking (NB) protocol, under different traffic loads and failure rates. All the simulations are run on a 4x4 mesh direct-connected NoC as the one depicted in Figure 3.

The first results evaluate the average checkpoint duration (Figure 6) for the blocking and non-blocking approaches, considering different uniform traffic loads, from a very light one to one approaching the maximum throughput of the NoC. The traffic load is measured in messages/cycle/task and each message has a length of 64 bytes.

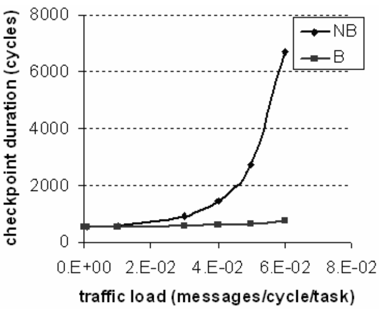


Figure 6. Checkpoint duration

We observe from Figure 6 that for very low traffic loads, the checkpoint duration of the blocking and non-blocking approaches are approximately the same. However, as the traffic load increases, the checkpoint duration in the non-blocking case significantly increases (the diamond-marked curve). In the blocking case (the square-marked curve), the checkpoint duration presents only a slight increase. This is possible because in the non-blocking case the recovery traffic is not significantly delayed by the application traffic (this one being blocked), as in the non-blocking case. However, the amount of application messages that flow through the NoC when the blocking starts is proportional to the traffic load, which explains the slight checkpoint duration increase with the traffic load, in the blocking approach.

Figure 7 depicts the checkpoint overhead in the same scenarios. The same significant ascending trend with the traffic load increase can be noticed for the non-blocking approach.

As the traffic load and the checkpoint duration increase, the probability of late messages becomes more significant, which explains the increase of the checkpoint overhead. On the other hand, the blocking approach keeps the overhead in very reasonable limits when compared to lower traffic load situations.

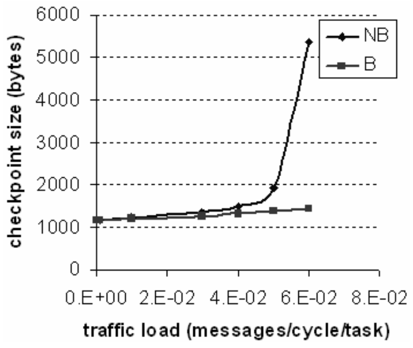


Figure 7. Checkpoint overhead

The next experiment studies the latency induced in the application execution. The latency performance is evaluated relative to the ideal case when there are no failures and no checkpoints are taken. Two traffic loads are considered: 0.01 and 0.03 messages/cycle/task (Figures 8 and 9, respectively) and different failure rates.

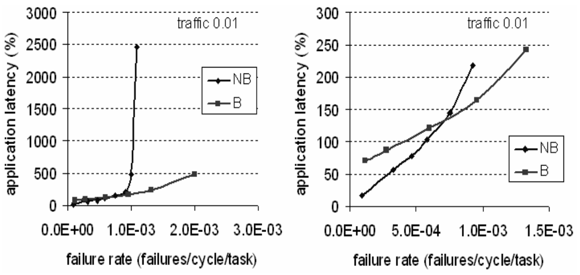


Figure 8. Application execution latency - traffic load of 0.01 (messages/cycle/task)

Normally, the application execution is delayed in the blocking approach by blocking the application during the checkpointing phases. This can be observed for relatively low failure rates (the failure rate is expressed in number of failures/cycle/task). However, for higher failure rates, we can observe (right side of Figure 8) that the blocking approach induces smaller latency than the non-blocking one (crosspoint at 7E-4 failures/cycle/task).

Moreover, the non-blocking approach becomes ineffective for higher failure rates, which is not the case for the blocking one. In fact, as the checkpointing duration is larger in the non-blocking case than in the blocking one, the time interval between two successive failures is not long enough to take a new non-blocking checkpoint, for higher failure rates. Thus, in the non-blocking approach, the probability to rollback to the same old checkpoint increases. Thus, parts of the application are re-executed several times, which leads to extremely high latency penalty.

For a traffic load three times higher (Figure 9), the same trend is maintained, but the crosspoint between the two approaches occurs for a lower value of the failure rate (3E-4 – see right side of Figure 9).

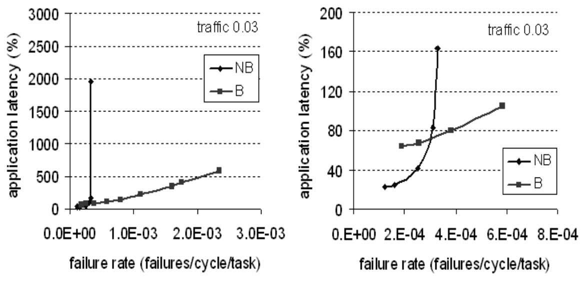


Figure 9. Application execution latency - traffic load of 0.03 (messages/cycle/task)

This is due to the fact that the difference between the checkpointing duration of the blocking and non-blocking approaches is larger for higher traffic loads (as seen in Figure 6).

Thus, if the expected failure rate is between the two traffic crosspoints, different approaches (blocking/non-blocking) are preferable for different traffic loads.

6. Conclusions and Future Work

In this work we analyzed and compared the effectiveness of blocking and non-blocking checkpointing for NoC based systems. We developed a protocol allowing both blocking and non-blocking modes, which can be dynamically set in depending on the actual traffic load and expected failure rate. Also, the protocol allows performing the same global checkpoint with subsets of blocking tasks and the rest non-blocking.

As future work, we plan to design and implement a decision algorithm that will dynamically switch between blocking and non-blocking mechanisms for each checkpointing phase of each task in the NoC-based system at run-time. Our initial experiments indicate that a decision based on the history of the checkpointing time would be well-suited for this purpose. We will also evaluate our dynamically-configured protocol on different types of application traffic patterns (not only uniform traffic) with non-static, bursty characteristics, and long-range dependencies.

7. References

[1] L. Benini, G. De Micheli. "Networks on chips: a new SoC paradigm". Computer, Vol. 35, Issue 1, Jan 2002, pp: 70-78.

[2] M. Elnozahy, L Alvisi, Y. M. Wang, D. B. Johnson. "A survey of rollback-recovery protocols in message-passing systems". ACM Computing Surveys, Vol. 34, No. 3, September 2002, pp: 375-408.

[3] B. Randell. "System structure for software fault tolerance". IEEE Trans. Software Eng., 1, pp: 220-232, 1975.

[4] A. Hansson, M. Coenen, K. Goossens. "Undisrupted Quality-of-Service during reconfiguration of multiple applications in networks on chip". Proc. of Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, pp: 954-959.

[5] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, F. Hessel. "Exploring NoC mapping strategies: an energy and timing aware technique". Proc. of Design, Automation and Test in Europe, March 2005, pp: 502 - 507 Vol. 1.

[6] B. Joshi, D. Pradhan, J. Stiffler. "Fault-tolerant computing". Wiley Encyclopedia of Computer Science and Engineering, January 15, 2008.

[7] S. Murali, G. De Micheli, L. Benini, T. Theocharides, N. Vijaykrishnan, M. Irwin. "Analysis of error recovery schemes for networks on chips". IEEE Design & Test of Computers, Vol. 22, no. 5, pp: 434-442, 2005.

[8] D. Rossi, P. Angelini, C. Metra. "Configurable error control scheme for NoC signal integrity". International On Line Testing Symposium (IOLTS), 2007, pp: 43-48.

[9] A. Ganguly, P.P. Pande, B. Belzer, C. Grecu. "Design of low power & reliable networks on chip through joint crosstalk avoidance and multiple error correction coding". Journal of Electronic Testing, Theory and Applications (JETTA), online, January 2008.

[10] T. Dumitras, R. Marculescu. "On-chip stochastic communication". Design Automation & Test in Europe (DATE), March, 2003, pp: 790- 795.

[11] G. Cao, M. Singhal. "On Coordinated Checkpointing in Distributed Systems". IEEE Trans. on Parallel and Distributed Systems, Vol. 9 n.12, pp: 1213-1225, 1998.

[12] L. Alvisi, E.N. Elnozahy, S. Rao, S.A. Husain, and A. Del Mel. "An analysis of communication-induced checkpointing". 29th International Symposium of Fault-Tolerant Computing, pp: 242-249, June 1999.

[13] S. Monnet, C. Morin, and R. Badrinath. "A hierarchical checkpointing protocol for parallel applications in cluster federations". 18th International Parallel and Distributed Processing Symposium (IPDPS), 2004.

[14] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel. "The performance of consistent checkpointing". Proc. 11th Symposium on Reliable Distributed Systems, pp: 86–95, Oct. 1992.

[15] K.M. Chandy, L. Lamport. "Distributed snapshots: Determining global states of distributed systems". ACM Trans. on Computing Systems 31, 1, pp: 63-75, 1985.

[16] D. Buntinas, C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, F. Cappello. "Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI". Elsevier, Future Generation Computer Systems, Vol. 24, Issue 1, January 2008, pp: 73-84.

[17] F. Quaglia, A. Santoro. "Modeling and optimization of non-blocking checkpointing for optimistic simulation on myrinet clusters". International Conference on Supercomputing 2003, pp: 130-139, 2003.