Fault-Tolerance Mechanisms for Permanent Failures in a Coherent Shared-Memory Many-Core Architecture

César Fuguet, Alain Greiner

Fault-Tolerance Mechanisms for Permanent Failures in a Coherent Shared-Memory Many-Core Architecture

César Fuguet, Alain Greiner
Sorbonne Universités, UPMC Univ Paris 06
CNRS UMR 7606, LIP6, F-75005, Paris, France
{cesar.fuguet-tortolero, alain.greiner}@lip6.fr

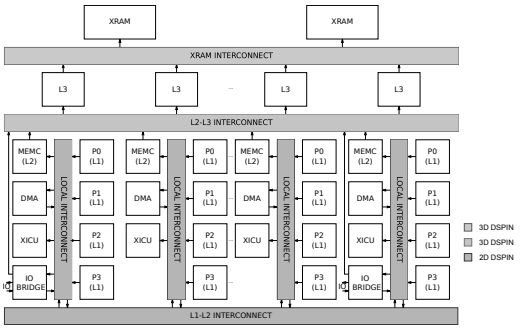

Figure 1. TSAR Networks-on-Chip

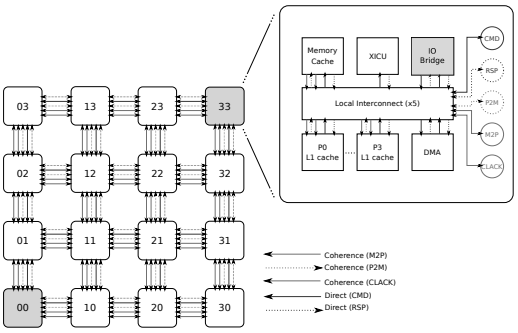

Figure 2. TSAR L1-L2 Network-on-Chip

I. Problem definition

The reliability in nowadays integrated circuits is rapidly decreasing as a consequence of the exponential augmentation on the transistor density which increases the probability of faults [1]. As a consequence, there is a diminution in both manufacturing yield and life-time (faster wear-out).

Therefore, the introduction of fault-tolerant mechanisms for allowing circuits to continue to work in the presence of permanents failures has become a primary necessity.

A. Many-core architectures

A many-core architecture exploits thread level parallelism through hundreds or thousands of cores. To increase memory bandwidth, these architectures are usually organized in several clusters, interconnected through one or several Network-on-Chip (NoC). Each cluster contains generally several processing cores and a local physical memory.

As many-core architectures are inherently redundant, we can imagine various degraded operation modes where, e.g., faulty cores or faulty memory banks are deactivated and only functional cores and physical memory banks are exploited by the operating system.

B. TSAR architecture

The TSAR architecture (Tera-Scale Architecture) [2] is a coherent shared memory many-core architecture jointly designed by BULL, LIP6 and CEA-LETI in the framework of the European CATRENE SHARP project. It implements a physical address space of 1 Terabyte (40 bits addresses) and the cores are MIPS32 processors supporting a 4 Gigabyte (32 bits) virtual address space.

This architecture supports up to 4096 cores and it is organized as a mesh of clusters containing 4 cores per cluster. TSAR implements 3 levels of cache memory using 3 levels of NoC interconnects (see figure 1) with a 3D stacking technology for the L3 cache. An external interconnect for IO peripherals is accessible through IO bridges placed in specific clusters (see figure 2).

In TSAR, the memory is physically distributed as every L2 cache manages an exclusive segment of the physical address space, defined by the most significant bits of the physical address. The cache coherence between L1 and L2 caches is guaranteed by hardware using a cache coherence protocol called Distributed Hybrid Cache Coherence Protocol (DHCCP).

II. Proposal

In this section, we describe our proposed "On the field" Detection, De-activation and Reconfiguration (ODDR) solution for permanent failure recovery. As the probability of faults increases with the silicon area, the footprint of the fault tolerance mechanisms should be small.

The ODDR mechanism exploits the intrinsic redundancy in many-core architectures, using the processor cores as test generators, test analyzers and reconfiguration drivers. Fault-recovery is achieved by means of software procedures executed in a distributed and cooperative manner by all processor cores during the bootstrapping process (boot), at each system's power-on.

The Operating System (OS) is only launched when the ODDR boot-loader completed the detection, deactivation and reconfiguration of the hardware platform. The OS is always executed on reliable hardware components because the faulty ones have been isolated during reconfiguration. In order to allow "on the field" dynamic reconfiguration, this firmware is stored in distributed on-chip Read-Only Memorys (ROMs) (one ROM per cluster).

A. General ODDR strategy

Any reconfiguration based fault-tolerance mechanism can be divided in four steps (as explained in [3]): fault-detection, fault-location, fault-containment and fault-recovery. The strategy proposed in this work provides solutions for the fault-location, fault-containment and fault-recovery parts of the reconfiguration process. The fault-detection uses existing Built-in Self Test (BIST) solutions for embedded memory banks, processor cores and NoC components.

Our strategy can be divided in two sequentially-executed stages: NoC Self-Test and Distributed Fault Location and Reconfiguration.

B. NoC Self-Test

During this stage, a fully distributed, hardware BIST is used to deactivate all faulty routers, and/or all inter-routers communication channels in the various NoCs of the TSAR architecture, as described in [4]. This BIST mechanism is implemented in the routers, and executed in parallel to detect faulty channels/routers in a 2D-mesh NoC. When a fault is detected, the faulty channel/router is deactivated and behaves as a "black-hole", i.e. it consumes every incoming packet and do not produce any outgoing packet.

C. Distributed Fault Location and Reconfiguration

The distributed firmware is executed in parallel by all processor cores during boot. The goal is to locate faulty cores, faulty memory banks and faulty routers, to build a global map of the operational platform, and to configure the various NoC routing functions, depending on the actual NoC topology, as it can now exist holes in the 2D Mesh or 3D Mesh. The different stages of this procedure are shown in figure 6.

The first step is to build a software configuration bus providing a reliable communication infrastructure in form of a spanning tree containing all functional (usable) clusters. The root of this tree is necessarily one of the two clusters containing an IO Bridge, and this communication infrastructure uses only software mailboxes supporting point-to-point, bi-directionnal communication between two neighbour clusters in the mesh. This software-based infrastructure is called Fault-Free Spanning Tree (FFST). An example of FFST is shown in figure 3 where the router associated to cluster (2,1) is faulty, and the root is cluster (0,0).

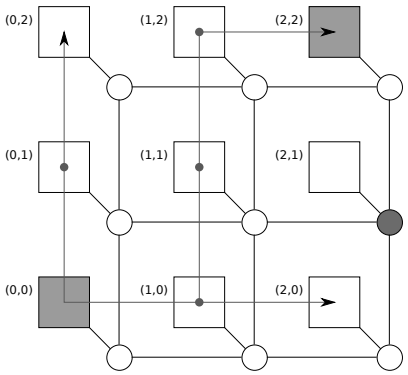

Figure 3. FFST example

The FFST is built in a bottom-up way in three sub-stages for improving fault-containment: Intra-Cluster, Inter-Cluster and FFST construction.

During Intra-Cluster stage, all processor cores in a given cluster execute in parallel a Software-based Self Test (SBST) as the one defined in [5] for fault-detection. After this, each core performs a mutual-test with all other cores in the same cluster (to avoid that a faulty core self-diagnoses as functional). To finish, a local leader is chosen among functional cores to represent the cluster in next stages. A cluster can participate in the FFST construction if it contains at least two functional cores (otherwise, mutual-test is not possible).

During Inter-Cluster stage, the local leader of each cluster performs a mutual-test with the four neighbor clusters (North, South, East, West in 2D mesh), as shown in figure 4 in order to check black-holes in the NoC, and to check which neighbor clusters have passed the intra-cluster test. Detection of black-holes or dead clusters needs a hardware watchdog timer in each processor core: if a software command/response transaction does not complete, the watchdog timer will trigger a software catchable exception, that will allow the core to diagnose the corresponding neighbour as non reachable.

During the FFST construction stage, the IO cluster playing the role of global leader starts the propagation of specific messages to functional neighbors (which recursively do the same), to finally build in a cooperative and distributed manner the FFST. From this point, it becomes possible to build a global map of all functional (non faulty) components in the platform.

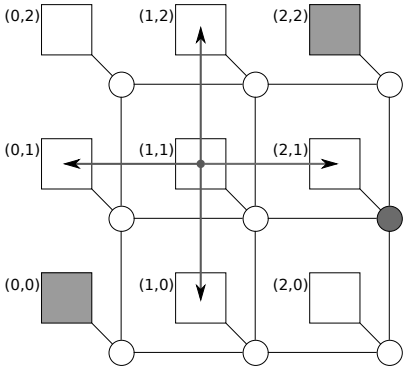

Figure 4. Inter-Cluster Stage

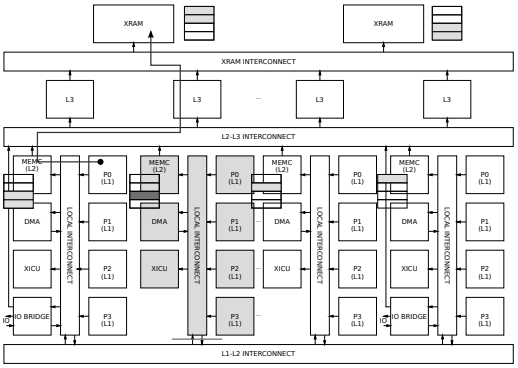

Figure 5. Physical Segment Recovery Example

D. NoC Reconfiguration

In the NoC reconfiguration stage, the global leader must first compute a modified dead-lock free and live-lock free deterministic global routing function for NoCs containing black-holes, because simple X-first routing function cannot be used. For this, we can use the algorithm proposed in [6], possibly extended to handle 3D mesh topologies. Finally, the global leader can use the FFST to write into the software addressable configuration registers contained in the routers to store the proper configuration and activate the various NoCs.

The configuration bus provided by the FFST infrastructure can be useful for other types of reconfiguration: As mentioned before, the TSAR architecture implements a physically distributed address space, where each cluster controls one segment (typically one Gigabyte per cluster), hence, if a cluster is unreachable, the corresponding physical memory becomes unusable for the OS. Fortunately, in TSAR, all clusters can communicate through several physically independent networks (L1-to-L2, L2-to-L3, XRAM). In case of a unreachable cluster, we can use the L2-to-L3 and XRAM NoCs to reach the memory segment of this cluster. This requires another reconfiguration of the global routing function to reallocate the corresponding segment to a neighbor functional cluster. An example of such segment migration is shown in figure 5 where the middle cluster is lost, but the corresponding physical memory segment is allocated to the first cluster.

E. Operating System (OS) loading

After the various NoCs have been configured, the OS can be loaded for normal operation. However, OS itself must be configured depending on the operational hardware architecture. As the number of processors or physical memory banks can change because of failures, the firmware must provide a description of the actual hardware organization before launching the OS.

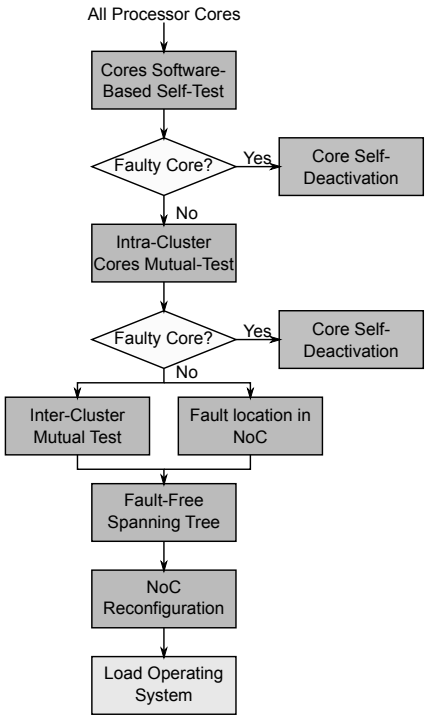

Figure 6. ODDR Strategy Stages

III. Experimentation environment

The experimentation environment used for validation of the proposed strategy relies on SystemC virtual prototyping. It will consist in a SystemC Cycle-Accurate Bit-Accurate (CABA) simulator of the TSAR architecture. This simulator will be developed using the library of SystemC components SoCLib [7] (which provides various components such as memories, interconnects, processors and others) and the TSAR specific SystemC library (containing especially the cache controllers).

The distributed firmware will be mapped in the ROMs and executed on the virtual prototype to evaluate the performance cost of the proposed ODDR strategy, depending on the number of faulty components.

IV. Conclusions

This work proposes an "On the field" fault-tolerance strategy for many-core processors allowing self-healing against permanent faults and therefore, increasing manufacturing yield and lifetime.

To minimize the fault-tolerance mechanisms footprint, we describe a robust distributed fault-tolerance strategy based on reutilization of processor cores as test generators, test analyzers and reconfiguration drivers through the execution of embedded software procedures at every processor power-on or reset to accomplish the "On the field" dynamic reconfiguration requirement. Execution is performed in a distributed and cooperative manner by all cores in order to reduce reconfiguration procedure time and also to increase processor robustness as the strategy avoids depending on a single core for achieving reconfiguration.

References

[1] ITRS (International Technology Roadmap for Semiconductors), http://www.itrs.net. I

[2] TSAR (Tera-Scale Architecture), https://www-soc.lip6.fr/trac/tsar. I-B

[3] B. Johnson, "Fault-Tolerant Microprocessor-Based Systems," *IEEE Micro*, vol. 4, no. 6, pp. 6–21, Dec. 1984. II-A

[4] Z. Zhang, A. Greiner, and M. Benabdenbi, "Fully distributed initialization procedure for a 2D-Mesh NoC, including off-line BIST and partial deactivation of faulty components," in *2010 IEEE 16th International On-Line Testing Symposium*. IEEE, Jul. 2010, pp. 194–196. II-B

[5] N. Kranitis, S. Member, and A. Paschalis, "Software-Based Self-Testing of Embedded Processors," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 54, no. 4, pp. 461–475, 2005. II-C

[6] Z. Zhang, A. Greiner, and S. Taktak, "A Reconfigurable Routing Algorithm for a Fault-Tolerant," pp. 441–446, 2008. II-D

[7] SoCLib, http://www.soclib.fr. III