Chip Self-Organization and Fault-Tolerance in Massively Defective Multicore Arrays

Jacques Henri Collet, Piotr Zajac, Mihalis Psarakis, Member IEEE, and Dimitris Gizopoulos, Senior Member IEEE

Abstract—We study chip self-organization and fault tolerance at the architectural level to improve dependable continuous operation of multicore arrays in massively defective nanotechnologies. Architectural self-organization results from the conjunction of self-diagnosis and self-disconnection mechanisms (to identify and isolate most permanently faulty or inaccessible cores and routers), plus self-discovery of routes to maintain the communication in the array. In the methodology presented in this work, chip self-diagnosis is performed in 3 steps, following an ascending order of complexity: interconnects are tested first, then routers through mutual test, and cores in the last step. The mutual testing of routers is especially important as faulty routers are disconnected by good ones with no assumption on the behavior of defective elements. Moreover, the disconnection of faulty routers is not physical (“hard”) but logical (“soft”) in that a good router simply stops communicating with any adjacent router diagnosed as defective. There is no physical reconfiguration in the chip and no need for spare elements. Ultimately, the multicore array may be viewed as a black box, which incorporates protection mechanisms and self-organizes while the external control reduces to a simple chip validation test which, in the simplest cases, reduces to counting the number of valid and accessible cores.

Index Terms—multicore architectures, multiprocessors, fault diagnosis, fault tolerance, massively defective nanotechnologies.

1 INTRODUCTION

Massively defective technologies are the direct consequence of the feature size shrinkage in today’s CMOS technology, in emerging nanoelectronics beyond CMOS, as well as in the technologies envisioned in the molecular-electronics approach, which in all cases increases the impact of structural fluctuations in the construction of the molecular edifices [1, 2] and increases the occurrence of permanent and transient physical faults in circuit operation [3, 4]. Due to these circuit reliability problems, dependability becomes one of the major challenges of all future nano-(or subnano-) technologies and fault tolerance becomes an essential property that must be integrated from the very beginning in every chip design. In this paper, we tackle the challenge of dependability and fault tolerance for multicore chips.

The main issue that arises in such massively defective multicore chips is the way they continue to deliver processing services in spite of defective cores. The current test practice in multiprocessor architectures is the thorough manufacturing testing of the chip, the localization and finally the isolation of the defective cores. Tolerating defective cores in multiprocessor chips is already implemented on the production lines of major manufacturers: the testing process of Intel’s Core Duo [5], AMD’s Quad core [6], IBM’s Cell processor [7] and Sun’s UltraSparc [8] supports diagnosis and isolation of defective cores providing partially good chips and improving the effective yield. However, the detection and deactivation of defective cores in future multicore arrays built in any massively defective technology pose new and complex challenges.

- First, the one-time factory testing practice is not suitable in the case of future multicore arrays which are highly susceptible to early defects but also to latent defects caused by time-dependent device degradation and material wearout which may produce system failure at any time in the field [9].

- Second, the dependability of future chips should be achieved with very limited external control since the increasing chip complexity reduces the controllability and observability of the chip from the external world making external testing of the array components (processors, memories, routers, etc) not scalable for massively parallel architectures.

Consequently, future chips should integrate some kinds of self-organization mechanisms to continue delivering their processing services in a quasi-autonomous way in spite of defective cores due to the presence of permanent and/or transient faults throughout their lifetime. Furthermore, chip organization should exhibit some physical redundancy to build a dependable sub-system from a defective and intrinsically unreliable system. In a redundant system, one may use...
self-diagnosis methods to identify and shut down the faulty components (in that case the diagnosis method depends on the component) or (and) spatial redundancy methods, for example duplication-with-compare or N-modular redundancy which consume significant portions of the chip area.

The choice of the physical organization of a multicore chip is crucial in this context. Although dozens of architectures have been presented in the literature in the two last decades [10, 11, 12, 13], two architectures prevail. Practically, the bus-based (or crossbar-switch-based) topology is commonly preferred when the number of cores is small, i.e. typically lower than 32, while a scalable array topology (2D-mesh or torus) is used for larger multi-node systems. In this paper, we consider the 2D array topology since we study chips including hundreds of cores typically. The array topology has the two following advantages:

- First, it is scalable and solves the complexity problem at the physical level because it is based on the regular replication of a few basic elements of definite complexity, like processor cores, routers and memory cores.
- Second, it is intrinsically highly fault-tolerant due to the symmetry of the array. There is no critical, indispensable element, while in centralized architectures, like for example in crossbar-switch based architectures, a critical element is the crossbar switch itself which is not replicated and thus should be protected or specially designed to tolerate internal physical faults.

In what follows, we consider a coarse-granularity vision of the chip, and the basic blocks are cores, routers, and caches, typically. There are two critical issues that any fault-tolerant approach of a multicore array must address:

1. **How to diagnose faulty cores in an array?** One could adopt a conventional test approach used for uncore processors, for example based on functional or structural scan-based testing. In this work, to preserve the self-diagnosis nature of the mechanism, we choose a self-testing approach based on software (or instruction-based) tests running on processor cores and requiring minimum external control. **Software-based self-testing (SBST)** provides significant advantages over other testing approaches because (a) it uses the core apparatus to virtually test itself eliminating the need for additional test hardware; (b) it is executed locally decoupling the core testing from the testing of interconnection infrastructure as involved by our modular diagnosis approach described in the following; (c) it can be easily applied in the field enabling the self-organization capability through the entire product lifetime; (d) it exploits the inherent execution parallelism of the array avoiding the scaling of the test application time with the number of cores. Recent approaches have proposed efficient SBST solutions for the testing of various processor models, from simple, low-complexity processors [14, 15] to complex pipelined processors with advanced performance mechanisms [16, 17, 18, 19] and real industrial designs [20, 21].

2. **How to tolerate faulty cores and how to preserve processing dependability?** The complexity of this task will explode in the future massively defective technologies where a significant portion of processing cores will be defective. The processing dependability in such systems should rest on the implementation and especially the coordination of fault tolerance mechanisms at several abstraction levels, i.e., at circuit level (fine granularity), at architectural level (coarse granularity) and in the scheduling, allocation and execution of processing threads. At the architectural level, many authors have advocated the idea of replacing the faulty cores by spare ones that implies complicated, physical reconfiguration of connections [22, 23, 24, 25, 26]. In this paper, we propose a **self-organizing fault-tolerant (SOFT) approach** at the architecture level to improve the processing dependability. It rests on the execution of different mechanisms, namely: (a) self-test and mutual-diagnosis of defective routers, processors and memories at the architectural level; (b) autonomous isolation of the faulty elements by good elements; (c) autonomous discovery of valid routes, and (d) deactivation of defective modules. We have already proposed a first SOFT approach in two previous publications [27, 28] which enables the chip to autonomously identify its defective parts and to self-reconfigure for building from the very beginning of operation a dependable array out of the blocks deemed as fault-free. Unfortunately, this approach is limited because it isolates the entire node including the router when the test mechanism detects a faulty core, thus generally sacrificing good communication resources. Consequently, the method becomes increasingly inefficient when the fraction of defective cores exceeds 25–30%; above this threshold many good cores of the array are not reachable and the processing power of the array drops dramatically.

To overcome these problems, we introduce a new SOFT approach, whose key original idea is to separate the diagnosis of cores from that of the communication sub-system (links and routers). Moreover, contrarily to what was presented in our previous works, testing of interconnects and routers is performed using hardware-based methods with high fault coverage. We believe that this approach is a logical tradeoff between test complexity and test effectiveness and is based on the following reasoning. Testing of interconnects and routers may well be performed using classical hardware-based methods with high fault coverage, as the necessary circuit overhead will be low compared to the size of cores. Note that self-diagnosis is executed in a modular manner and in the ascending order of complexity: it begins with the diagnosis of interconnects, proceeds with the diagnosis of routers and finally with the diagnosis of cores. We have also ensured that our methodology is consistent in that the “smaller” element always performs the test of the “larger” element which reduces the number of erroneously disconnected fault-free elements and significantly improves on our previous approach. The crucial benefit of the new approach is that all cores (whatever their states are, good or defective) are now accessible up to typically 25-30% of defective routers.

In the last part of this work, we address the problem of tolerating the erroneously diagnosed cores or routers. It is clear that the larger the multicore array, the more errone-
ously diagnosed cores (EDC) persist in the chip due to the incomplete fault coverage of the diagnosis mechanisms. These EDCs may lead to unpredictable and undependable operations, when they are faulty but erroneously diagnosed as good. We calculate the persistence statistics of erroneously diagnosed cores (or routers) in the chip at the end of the preceding SOFT phase, and the production yield as a function of the fault coverage and the accepted chips including a fraction of EDC. However, because EDCs cannot be identified and are intrinsic to the partial fault coverage of tests, multicore chips in massively defective technologies will have to dedicate a fraction of their processing power to detect and tolerate both permanent faults (which have escaped from the diagnosis process) and transient faults in operational conditions.

The paper is organized as follows. In Section 2, we discuss self-testing methods currently used in unprocessors, their application to multicore arrays and the mutual diagnosis approach used in previous works. In Section 3, we describe the proposed self-organizing fault-tolerant approach, and we introduce the new version to overcome the limitation of previous studies. In Section 4, we describe the last step of self-organization, i.e. route discovery and chip sorting/validation mechanism which reduces (in the simplest case) to counting the number of valid cores which are reachable in the defective array. Finally, in Section 5, we study the tolerance of erroneously diagnosed cores, in particular the case when faulty cores are deemed as fault-free. Section 6 concludes the paper.

2 Self-Diagnosis Methods in Multicore Arrays

In this section, we first present various testing methods currently used in unprocessors and next we discuss their applicability to the diagnosis of cores in multicore arrays. Finally, we describe the mutual diagnosis approach used by previous approaches [27, 28], explain how it can surpass the benefits of self-testing methods but also analyze its limitations to the diagnosis of defective nodes in multicore arrays.

2.1 Testing Methods for Unprocessors

Decoupling the core internal testing from the testing of the entire array, we can adopt conventional practices currently used for the testing of unprocessors for the modular testing of multicore arrays. These test practices can be generally categorized to functional test approaches, where the processor functionality is verified at normal operating mode through its external interface and structural test approaches, where internal blocks or structures of the chip are tested separately at special test modes. The external functional testing is usually performed by big-iron testers connected at the processor pins, while the structural testing solutions, like scan-based approaches improve the accessibility of the internal processor nodes involving Design-for-Testability (DfT) techniques and relax the requirements for high-cost testers. However, the hardware overheads of DfT modifications are sometimes not acceptable for carefully optimized designs like processors. Moreover, the scan-based techniques can not catch defects which are manifested when the chip operates at its actual speed in the field of operation.

For the above reasons, self-test approaches have been adopted which move the test functions from external testers to on-chip resources enabling at-speed testing and reducing the overall test cost. Hardware-based self-test approaches, such as Built-In Self-Test (BIST) techniques, have been efficiently applied to regular structures, like memory arrays [29]. However, the application of hardware-based BIST techniques to processor cores requires extensive modifications to make circuits BIST-ready, i.e. to eliminate bus conflicts or propagation of unknown values to observable points during BIST.

Another self-test approach that has gained increasing acceptance in the last decade is software-based self-test (SBST) [14]-[21]. The key idea of SBST is to exploit the on-chip programmable resources to run self-test programs: test patterns are generated and executed by the processor itself using its native instructions. In a typical low-cost SBST flow, the test programs and data are downloaded into on-chip memory, using a low-speed, low-cost tester. Subsequently, these test programs are executed by the processor at actual/full speed and test responses are stored back in the on-chip data memory from which they can be read out and compared with the correct responses. The main advantages of SBST are:

- **Non-intrusive approach**: No extra hardware is required and no extra power is consumed compared to the normal operation mode.
- **At-speed testing**: Test application and response collection are performed at the processor’s actual speed. Thus all physical failure mechanisms that lead to performance problems and not logic only ones can be detected.
- **No over-testing**: Since SBST is performed in normal mode, it avoids detecting errors that will never occur during the normal processor operation increasing yield. This is a usual problem when structural, scan-based self-test methods are used.
- **Reusability**: Similar self-test programs can be used at manufacturing testing or in-field testing throughout the product lifetime.

2.2 Applicability of Unprocessor Testing Methods to Multicore Arrays

The current industrial test practice for unprocessor chips or chips with a few processor cores is to use a combination of the above test approaches to achieve the best trade-off between lowest Defective Parts per Million (DPM), test application time and overall test cost. However, the one-time factory testing practice targeting high defect coverage is not adequate in the case of multicore arrays where the massively defective technology increases the circuit susceptibility to defects caused by time-dependent device degradation and material wearout and may occur at any time during product lifetime [9]. Which are the requirements that a core diagnosis method for massively defective multicore arrays must supply?
1. **In-field reusability**: It must have the potential to be used both at (a) factory testing setup to detect infant mortality defects and enable sorting (or binning) of multicore arrays according to the number of their operational cores, and (b) in-field testing setup to detect latent defects and enable the graceful performance degradation of the multicore array shutting down the faulty cores.

2. **Exploitation of execution parallelism**: A challenging problem in the testing of multicore arrays is the minimization of the test application time [30]. The test application time will scale with the number of cores exploding the total test time when the processor core test patterns can not be applied in parallel to many processor cores taking advantage of the execution parallelism of the multicore array.

3. **Small hardware overhead**: Another critical feature of the processor core test approach incident to the modular methodology proposed in this paper is the amount of imposed test hardware. The modular methodology is based on the assumption that the fault probability of a module is proportional to its size, and thus the fault probability of the module under test is much larger than that of the module that performs the test (test component, i.e. the chip infrastructure used for testing). This assumption will be violated when the processor core test approach imposes significant amount of test hardware reducing the gap between the two probabilities.

The test approaches based on external patterns (either functional patterns or scan/ATPG generated patterns) typically requires a complex test access mechanism to apply test stimuli to the cores, to collect the responses and setup the required control to put the core in test mode. They are not scalable to large arrays and may lead to routing difficulties because the communication network has already imposed significant routing overhead. Recent approaches for the testing of NoC-based SoCs [31] propose the modification of the existing functional interconnect infrastructure of the SoC to be reused for test access purposes. Also [32] proposes test scheduling methods to increase test parallelization and reduce test application time.

Self-test approaches fulfill the modular testing requirements of a multicore array. Hardware BIST techniques are ideal for components with a regular topology like memory arrays; actually, they are the only practical solution for embedded memories with limited accessibility. Furthermore, efficient SBST solutions have been recently proposed for a variety of processor models, from simple, low-complexity processors [14, 15] to complex pipelined processors with advanced performance mechanisms [16, 17, 18, 19] and real industrial designs [20, 21]. The efficiency of SBST methodology has also been demonstrated for periodic, in-field, self-testing of embedded processor cores targeting permanent and intermittent operational faults [15]. Recently, the applicability of SBST to bus-based multiprocessor architectures and the efficient exploitation of execution parallelism have been demonstrated [30].

### 2.3 Mutual Diagnosis Approach

A diagnosis approach that adopts the SBST methods described in the preceding section for the testing of the cores in a multicore array is based on the assumption that, following the diagnosis step, each faulty core is able to correctly execute the protection actions. In other words, it is implicitly assumed that the core includes some fault-free subsystem able of executing the protection mechanisms. The mutual diagnosis approach [33, 34, 35, 36] in multicore arrays overcomes the above limitation since the isolation of each faulty core is performed by its good neighbors. Mutual diagnosis in its principal is built on top of a self-test method, because each core first executes its self-tests (running for example test programs locally stored in node memory or activating a built-in test pattern generator) and generates a test signature. Then, it sends this signature to its four neighbors. As all cores execute separately this procedure, each one may compare its own signature with those received from its four neighbors. When a good core discovers that the signature from an adjacent core is different from its own one (or when it receives no signature within some reasonable timeout interval), it considers the adjacent core to be faulty and tells its router that all the communication with this node should be stopped. We call this the communication suspension mechanism (CS) or node isolation mechanism.

To be concrete, let us describe the mutual test mechanism between the two adjacent nodes shown Fig. 1 [27, 28].

---

![Fig. 1: Node structure. Ci is the core, Ri the router. The figure also shows the test bits $B_{N}$, $B_{S}$, $B_{E}$, and $B_{W}$, which are updated by the diagnosis process and control the inhibition of communications (see text).](image-url)
neighbour. Following the comparison mechanism, the router will read the direction bits in the local memory and cancel all incoming and outgoing communication with the neighboring nodes when respective direction bits equal zero.

The mutual diagnosis approach exhibits the following benefits:

- Decisions made by faulty nodes are of no importance because if a core is faulty, it will certainly be disconnected by its good neighbors and if it belongs to a faulty cluster, the cluster is disconnected by good nodes. This is an intrinsic advantage of the mutual diagnosis: the isolation mechanism of faulty nodes relies exclusively on the action undertaken by good nodes with no hypothesis on the operation of faulty cores, faulty routers or faulty interconnects, contrarily to the self-test approaches which assume that a faulty element can always diagnose itself correctly.
- There is no physical reconfiguration to establish valid routes in the chip and no need for spare elements contrarily to what was suggested in many previous works [23, 24, 37, 38, 39].
- The disconnection mechanism automatically splits the communication network in several zones of faulty and valid (operational) nodes, regardless of the actions of defective nodes.

### 2.4 Limitation of mutual diagnosis

Unfortunately, the mutual diagnosis approach has one pernicious limitation, which arises from the different complexity levels of the core and the router. Let us reasonably assume that the core is much more complex than the router, i.e., $N_c > N_r$, where $N_c$ and $N_r$ are the core and router gate counts. This assumption is reasonable, as all modern powerful super-scalar execution cores include several millions transistors. Comparatively, the related literature shows that it is possible to build a one-clock-latency 5-input 32-bit-parallel crossbar with typically 100 to 200 kilo-gates, including the input FIFO memories [40, 41, 42]. Therefore, if the fault probability of a combinatorial block is approximately proportional to its gate count (which is a legitimate assumption), the probability $p_{FR}$ of a router to be faulty should be typically a few percents of the probability $p_{FC}$ of a core. Consequently, when a good core discovers that the signature of an adjacent node is different from its own signature, this is most probably due to faults in the adjacent core and not due to faults in the corresponding router. Nevertheless, the CS mechanism isolates the entire node and sacrifices the router and the communication capability of the network altogether. Thus, because $p_{FC} > p_{FR}$, the number of sacrificed good routers equals approximately the number of defective cores and it becomes progressively impossible to maintain full reachability between any two nodes in a faulty 2D mesh network. We showed in [27, 28] that practically, all valid cores can be contacted as long as the fraction of defective and disconnected nodes is typically less than 25%. However, 10-12% of valid cores can not more be contacted and are lost for the processing when the fraction of defective cores reaches 40%. The fraction of lost cores still increases at higher density of faulty nodes. So, in the next section, we introduce a new approach which overcomes this limitation.

### 3 Modular Diagnosis Approach

The proposed diagnosis approach is based on the two following key ideas:

1. Separating the diagnosis and the configuration of the communication architecture (i.e., routers and links) from the validation of processing cores.
2. Diagnosing the routers using mutual testing and a tester much simpler that the router so that $p_{FR} >> p_{FT}$ (where $p_{FT}$ is the fault probability of the tester) to ensure that few routers will be erroneously sacrificed when applying the CS mechanism.

For the implementation of the proposed approach, we need to modify the structure of the node shown in Fig. 1. It becomes slightly more complex (see Fig. 2) and integrates some additional testing blocks, namely:

- A tester T which is a small block to generate test patterns for the router and to perform the logical isolation of the faulty core;
- A test data generator (TDG) and a test error detector (TED) which are two very small blocks to test the interconnect.

The diagnosis process is performed in three steps:

- **Step 1**: diagnosis of interconnects;
- **Step 2**: diagnosis of routers; and
- **Step 3**: diagnosis of cores.

Let us review in details the different steps.

**Step 1: Diagnosis of interconnects**

Many efficient BIST-based solutions have been proposed for interconnect testing achieving very high fault coverage. Here, we adopt the BIST solution proposed in [43] where each link is equipped with two test blocks TDG and TED, as shown in Fig. 2. Each TDG generates testing sequences, based on the Maximal Aggressor Fault (MAF) model, which are analyzed by the corresponding TED. According to MAF model, one line is considered to be the victim, while all oth-
ers are aggressors. MAF model can abstract crosstalk defects in interconnects, while only a small number of tests per line are required to cover all different crosstalk defects. We slightly modified the method by adding to the TED the ability to inhibit the communication across a neighboring link which does not pass the test. For instance, the direction bit BE of node 1 will be updated to 0 in Fig. 2 if the test of interconnect between nodes 1 and 2 fails.

Note that test circuits are very simple. For example, the TDG circuit proposed in [43] for a 32-bit-wide link requires only 471 two-input NAND gates. Given their low complexity, the test circuits can be easily protected using N-modular redundancy without excessive area overhead so that we may assume that they cannot fail and generate no diagnosis error, i.e., $p_{F,TDG} \ll p_{F,TEG}$, where $p_{F,TDG}$ and $p_{F,TEG}$ are the failure probabilities of the test circuits and link, respectively.

Step 2: Mutual diagnosis of routers

In step 2, the tester T (see Fig. 2) performs the diagnosis of the router. We assume that the tester, as in a classical BIST approach, loads test vectors (which are identical for all routers and can be either stored in local memory or generated by a hardware TPG) and gathers all test responses to calculate a diagnosis signature. Then, we consider mutual test, instead of using classical self-test. Consequently, each tester sends its diagnosis signature to its four neighbors via the active links and compares its own signature with that of each neighbor. If signatures differ, a tester assumed to be fault-free concludes that the corresponding neighbor is defective. Concretely, the good tester updates the node direction bits $\{B_i | i=N,E,S,W\}$ (see Fig. 2), and subsequently, the good router suspends the communication in direction $i$ when $B_i=0$.

At first sight, this mutual diagnosis seems very similar to that described in section 2.3. However, there is a crucial difference in that the tester is simpler than the router, while in the preceding case the core which performed the tests was more complex. Previous studies have proved that a router can be efficiently tested by a relatively small tester. For instance in [44], the authors studied a BISTed router for a 2D-mesh multiprocessor chip. They estimated that the BIST controller occupies about 13% of the total router area. In [45], the authors described a BIST methodology for FIFOs (which occupy a large portion of router) and estimated that the BIST logic requires only about 4000 transistors for a SRAM-type FIFO of size 512×32-bit words to achieve very high fault coverage: 97.8% coverage comprising stuck-at faults, transition faults, inversion coupling faults, data retention faults, and stuck-open faults. Note that the combinatorial part of the router is very simple, (typically implemented with a few hundred gates [46, 47]) and can be protected using N-modular redundancy. Consequently, only a few good routers will be sacrificed by the CS which follows the mutual diagnosis given that the tester is much smaller than the router, thus $p_{F,T} \ll p_{F,R}$. In other words, in the majority of the cases when a router is diagnosed as faulty, it will be due to the actual occurrence of a fault in the router.

Step 3: Diagnosis and validation of cores

Each core in the array executes its self-test program independently of the router mutual diagnosis mechanism described in the previous step. The self-test program can be stored in: (a) a local cache, (b) shared on-chip DRAM or (c) a local ROM. In the case of distributed storage of the self-test program in local memories, the self-test program is executed faster, without time consuming shared memory accesses compared to the case of centralized storage. However, the disadvantage is that multiple identical copies of the test program must be downloaded locally to each node. The choice of preprogrammed local ROMs does not require time consuming downloads but it wastes on-chip programmable resources.

After the self-test program execution is completed, the core calculates one or more test signatures and stores them in the local memory. Then, tester T performs a simple comparison between the calculated signatures and predefined “golden” signatures also stored locally in the node. In case of a mismatch, the core is disabled by the tester.

Efficient SBST solutions have been recently proposed [14]-[21] for the testing of various processor models, such as publicly available models shown in Table 1:

- **Plasma**: low-complexity processor implementing a simple 3-stage pipeline mechanism
- **miniMIPS**: medium-complexity processor with a more sophisticated 5-stage pipeline mechanism and coprocessor functionality
- **OpenRISC 1200**: more complex RISC processor with basic DSP functionality and virtual memory support used in many industrial applications.
- **Enhanced miniMIPS**: miniMIPS processor core enhanced with branch prediction mechanism (based on branch target buffers).

<table>
<thead>
<tr>
<th>TABLE 1</th>
<th>PROCESSOR MODELS WITH VARIOUS COMPLEXITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor model</td>
<td>Features</td>
</tr>
<tr>
<td>Plasma</td>
<td>MIPS-I ISA</td>
</tr>
<tr>
<td>miniMIPS</td>
<td>3-stage pipeline MIPS-I ISA</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>5-stage pipeline Basic DSP capabilities Virtual memory support Performance counters</td>
</tr>
<tr>
<td>Enhanced miniMIPS</td>
<td>Branch prediction unit (Branch Target Buffer with 2-bit predictors)</td>
</tr>
</tbody>
</table>

Table 2 presents the statistics of these SBST approaches in terms of fault coverage, test size and test application time. All the approaches achieve a good compromise among fault coverage (more than 90% stuck-at fault coverage in all cases), test size (including both test program and test patterns) and test application time.
As the processor complexity increases, the SBST approaches require larger test programs and execute more instructions to achieve high fault coverage level. The increased complexity of the enhanced miniMIPS model is not followed by a relative increase in self-test size since regular structures like the branch target buffer which dominates the branch prediction unit area can be tested more easily by software tests.

<table>
<thead>
<tr>
<th>Processor model</th>
<th>Fault coverage</th>
<th>Test size</th>
<th>Test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma</td>
<td>95.3</td>
<td>3.4</td>
<td>5.7 K</td>
</tr>
<tr>
<td>miniMIPS</td>
<td>95.0</td>
<td>6.3</td>
<td>7.2 K</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>90.0</td>
<td>12.5</td>
<td>56.7 K</td>
</tr>
<tr>
<td>Enhanced</td>
<td>92.6</td>
<td>10.3</td>
<td>33.7 K</td>
</tr>
</tbody>
</table>

### 4 SELF-ORGANIZATION

#### 4.1 Route discovery

In this subsection we describe the autonomous discovery of routes in the communication network which, with the mutual diagnosis and local disconnection mechanisms described in Section 3, is one of the foundations of the self-organization method in the defective multicore array. Route discovery consists of executing a handshake mechanism, which we proposed previously in [27, 28]. In this approach, a node emits a request message (RM) in the zone(s) composed of good nodes. The RM is propagated by means of flooding diffusion, where each node forwards each incoming RM to all links except the incoming link [48]. Flooding protocols enable moving around the disconnected zones, i.e., around zones including routers diagnosed previously as faulty. The idea here is that each router forwarding a RM adds in the message header the routing which is locally executed (for instance, the index of the output link), so that during propagation, a RM stores the route which it follows. Then, each valid core receiving the RM sends one route acknowledgment message (AM) back to the emitter. The AM simply follows the RM route in the reverse direction. Finally, the emitter core builds an array of routes to contact other nodes by analyzing received acknowledgment messages.

#### 4.2 Chip validation

Chip validation rests on the execution of the route discovery mechanism from an input/output port (IOP) of the chip. Note that the crucial question is not to decide whether the validation test must be executed by the chip itself but that the chip validation test must be simple, and especially scalable to large multicore arrays. In this context, using a low-cost external tester is likely the simplest solution, as it may be assumed that the external tester is reliable and built from a fault-free technology. The test that we consider reduces to discovering the routes from the external tester, which emits a RM through an IOP. In the simplest approach, chips are simply sorted as a function of the number of AMs returned to the tester (which equals the number of valid accessible cores). Note that some more sophisticated tests could be also considered, based on the emission of several messages and a more sophisticated analysis of the acknowledgment messages [49].

### 4.3 Reachability limits in a defective array

To appreciate the reachability limits, let us consider for instance a technology such that as much as 30% of processor cores are defective, which means typically that $p_{RC}$ is of the order of 0.3. Of course one may consider more defective technologies with 50, or 60% defective cores, but this poses the question about their practical interest when the number of defective cores exceeds the number of good ones. In such technologies, the mutual diagnosis approach described in [28] would sacrifice a large number of good routers (typically as many as defective cores when $p_{RC} >> p_{FR}$) and we know that the communication capability in a 2D-mesh network quickly collapses when the fraction of defective routers exceeds 30%.

On the contrary, the proposed modular-diagnosis approach separating the diagnosis of the communication architecture from the validation of the cores completely removes the above limitation. The direct benefit is that all cores (whatever their states, good or defective) are reachable up to 25-30% of defective routers. Two reasons explain this remarkable property: first, few routers will be erroneously disconnected (because $p_{RC} >> p_{FR}$), and second because of the ascending complexity order, the fault probability of routers even in very defective technologies, should be much smaller than that of cores, thus typically of the order of a few percents. The exact value of $p_{FR}$ has no importance, as we know that almost all cores will be accessible so long as the fraction of defective and disconnected routers in the array is below 20%.

### 5 TOLERANCE OF ERRONEOUS DIAGNOSES

Unfortunately, several fault configurations directly threaten the dependability of the multicore array and cannot be avoided by the modular diagnosis and the self-organization mechanisms described in the previous sections. This occurs when some faulty elements are erroneously diagnosed as fault-free and may lead to unpredictable and undependable operation of the array. In what follows, we analyze these cases for cores and routers separately.

#### 5.1 Statistics of erroneously diagnosed cores (EDC)

Core diagnosis is based on a SBST approach while the local tester is responsible for the comparison of the test signatures and the disconnection of the cores diagnosed as faulty. An erroneous diagnosis occurs when the core is faulty but it is not disconnected by the tester. This situation happens due to one of the following reasons: (a) some faults in the core are not detected by the SBST method due to limited fault coverage of the test; (b) the tester module is defective and does not disconnect a faulty core; (c) the test compactor produces a final correct signature although intermediate test responses are faulty. This can occur due to aliasing problems of the test compactor; for instance, if test responses are compacted using a 16-bit CCITT CRC algorithm, it is known that the number of cases leading to fault masking in the final test
signature is negligible, less than 0.01%. The opposite case (i.e., the disconnection of a good core by a faulty tester) reduces the processing power of the chip, but does not threaten the chip’s dependability.

In this section, we study the occurrence of such cases that we characterize by the probability to have $n_E$ defective elements erroneously diagnosed as fault-free in a population of $N$ elements. Note that we do not specify the very nature of the elements which plays no role in the calculation. First, the probability $P(n, N)$ to have $n$ faulty elements out of $N$ simply reads:

$$P(n, N) = \binom{N}{n} p_F^n (1 - p_F)^{N-n} \quad \text{(Eq. 1)}$$

where $p_F$ is the fault probability for the elements under consideration. Then, in a first approach, the probability to have $n_E$ erroneously diagnosed elements (EDE) among these $n$ faulty elements is $P(n_E, n) = \binom{n}{n_E} Q^{n_E} (1 - Q)^{n-n_E}$ where $Q$ is the fault coverage of the test program. Thus, the probability $P(n_E, n, N)$ to have $n_E$ EDE among $n$ faulty elements in $N$ nodes simply reads:

$$P(n_E, n, N) = \binom{n}{n_E} Q^{n_E} (1 - Q)^{n-n_E} \quad \text{(Eq. 2)}$$

The quantity $P(0, n, N)$ is of special interest, as it is the probability that all defective elements are correctly diagnosed. If we specialize Eq. 2 to cores in an array (so that $N$ is the number of nodes, $n$ the number of faulty cores, and $Q$ the fault coverage of the SBST method), $P(0, n, N)$ is nothing but the production yield of arrays containing $n$ faulty cores out of $N$, with all faulty cores correctly diagnosed as defective.

Let us consider Fig. 3, which calculates several probabilities for an array with 50 cores when the core fault probability is $p_F=0.1$. The upper line calculates $f(n_{\text{MAX}}) = \sum_{n=0}^{n_{\text{MAX}}} P(n, 50)$. Point C of coordinates (6, 0.78) which belongs to this line means that all chips having between 0 and $n_{\text{MAX}}=6$ defective elements out of 50 represent 78% of all chips when the fault probability of the element is 0.1. We chose at most $n_{\text{MAX}}=6$ defective elements out of 50 to estimate the fraction of faulty chips which preserve at least 88% of the processing power of a fault-free chip. In other words, point C means that 78% of chips statistically preserve at least 88% of the processing power when $p_F=0.1$. Now, let us consider the lowest curve which calculates $f(n_{\text{MAX}}) = \sum_{n=0}^{n_{\text{MAX}}} P(0, n, 50)$. Point A with coordinates (6, 0.5) means that the chips preserving at least 88% of the processing power and which include no erroneously diagnosed cores (EDC) represent 50% of all chips when the fault coverage is $Q=0.9$. Point B means these chips represent 62% of all chips when the fault coverage is $Q=0.95$. Point D means that about 78% of chips will be correctly diagnosed when $Q=0.95$. The quasi-perfect superposition between the dashed red line which calculates $f(n_{\text{MAX}}) = \sum_{n=0}^{n_{\text{MAX}}} [P(0, n, 50) + P(1, n, 50)]$ and the blue line means that all chips have 0 or 1 incorrectly diagnosed element. So, Fig. 3 shows the impact of the fault coverage on the probability to correctly diagnose a multicore chip.

The next figure (Fig. 4) shows the same statistics of EDCs for a large array with 250 nodes.

![Figure 3](image-url)  
**Fig. 3:** Upper curve: probability that a chip includes at least a predefined fraction of defective cores. Lower curves: Average production yield when the array contains no erroneously diagnosed cores (EDC) versus the top number of accepted faulty cores (see details in the text).

![Figure 4](image-url)  
**Fig. 4:** Upper curve: probability that a chip includes at least a predefined fraction of defective cores. Lower curves: nEDC=0 means that the curve shows the production yield of arrays containing no erroneously diagnosed core, nEDC=1 the production yield of arrays containing at most one erroneously diagnosed core, and so on for the curves nEDC=2, 3, 4. Point C of coordinates (30, 0.90) means that all chips having between 0 and $n_{\text{MAX}}=30$ defective elements out of 250 represent 88% of all chips when the fault probability of the element is 0.1. In other words, point C means that 90% of chips statistically preserve at least 88% of the processing power. Point A with coordinates (30, 0.1) means that the chips preserving at least 88% of the processing power...
and which include no erroneous diagnoses cores (EDC) represent approximately 10% of all chips when the fault coverage is \( Q = 0.9 \). Point B means that chips including from 0 to 2 EDC represent 52% of all chips when the fault coverage is \( Q = 0.90 \). Thus, it is clear that the more nodes, the more residual EDC persist in the chip after the diagnosis process.

5.2 Statistics of erroneously diagnosed routers

The occurrence of erroneously diagnosed routers (EDR) is very rare. For instance, let us consider that the core fault probability is \( p_{FC} = 0.1 \), and that due to the complexity order between the core and the router the router fault probability is \( p_{FR} = 0.01 \). The fault coverage of the router using a BIST solution implemented by the tester is extremely high, i.e. \( Q = 0.978 \) [45]. So, we conclude that the probability that all routers are correctly diagnosed is 99% when \( N = 50 \), and 95% when \( N = 250 \).

5.3 Tolerance of EDCs

The two previous sections showed that there is always a non-zero probability that some faulty cores are erroneously diagnosed as good, possibly leading to non-dependable operation. This issue is inevitable and comes from the fact that all diagnosis methods of modern complex cores have incomplete (although very high in some cases) fault coverage. Although an external testing approach may achieve higher fault coverage, the functional self-test approach is more suitable for our proposed modular diagnosis methodology as we analyzed in Section 3. Consequently, multicore chips in massively defective technologies will have to dedicate a fraction of their processing power to detect and tolerate both permanent faults (which have escaped from the diagnosis process) and transient faults in operational conditions.

This dedicated processing power should be adaptable depending on the criticality level of the applications under execution. The best approach to enhance the operation dependency at this level is likely to consider that each node of the array should be a symmetric multicore (typically a dual, or a quad core) which is configured depending on the criticality of the application that is allocated to it [50, 51, 52]. Considering that a node is a dual or quad core does not significantly modify the diagnosis analysis described in Section 3. The sole difference is that tester T will have to check the generated signatures of each core.

When application criticality is low, no execution redundancy may be even considered and each core in a node could operate separately, possibly increasing the execution parallelism. Contrarily, for highly critical applications, one should consider local node redundancy, in particular using the so-called lock step configuration (LSC) where two (or more) identical cores run cycle by cycle the same instructions of the same program. In this symmetric multithreading (SMT) processor, a checker circuit compares the outputs, flags an error and initiates a software recovery sequence on an output mismatch between the cores [53, 54, 55]. This mechanism is already implemented in multicore processors [56]. This approach is especially well adapted to the detection and tolerance of transient faults. Its application to the detection of permanent faults is more complicated. One solution is that the tester should store statistics on the error cases and send a “permanent error” message to the scheduler when some user-defined criteria of error occurrence are exceeded [49]. One important feature is that fault tolerance at thread level is distributed in the array and achieved locally in each node. To complete this section, let us underline that the input ports of the chip belong to the critical nodes, which should operate in the LSC, quite simply because no reconfiguration alternative is possible in the event of permanent or transient failure on them.

6 Conclusion

We presented an architectural approach for building SOFT multicore chips in massively defective technologies. SOFT properties result from the conjunction of a diagnosis phase (to identify the defective parts) followed by the execution of simple protection actions to isolate the parts diagnosed as faulty and to maintain the communication in the array. Three points are important in the proposed approach:

1. All steps are executed autonomously by the array with no external control enabling fault-tolerant self-organization.
2. Fault probabilities of the various elements of the array can be ordered in an increasing sequence \( p_{F1} < p_{F2} < p_{F3} \) (where \( p_{F1} \) if the fault probability for interconnects, \( p_{F2} \) for tester modules, \( p_{F3} \) for routers, and \( p_{F4} \) for cores) which is the expression in terms of fault occurrence of the complexity order between these different elements. The benefit of the proposed modular approach is that the fraction of routers useless disconnected is small given the ratio \( p_{F2}/p_{F1} \) is small. Consequently, one may claim that the vast majority of valid cores can be reached in the array, independently of their state.
3. The diagnosis of cores is independent of that of the communication architecture. The cores are self-tested running test programs and a local tester is used to compare the generated test signature with the “golden” signature and validates or disables the core.

Consequently, a SOFT multicore array may be viewed as a black-box which autonomously executes diagnosis and protection mechanisms. The sole mandatory external control reduces to a simple chip validation test, which in the simplest case consists in counting the number of valid cores which can communicate with the external world. The proposed SOFT approach is scalable and significantly contributes to enforce dependable operation of large multicore arrays in massively defective technologies. Unfortunately, the global dependability of a massively defective multicore array cannot be guaranteed. This issue is inevitable and comes from the fact that all diagnosis methods of modern complex cores have incomplete (although very high) fault coverage.

In this context, the global dependability must be ultimately guaranteed by the software executed by processing elements of the array. Thus, any multicore array will have to dedicate a fraction of its processing power to detect and tolerate both permanent faults (which have escaped from the diagnosis process during manufacturing) and transient faults in opera-
tional conditions. This dedicated processing power should be adaptable depending on the criticality level of the applications under execution.


Jacques Collet is currently a Directeur de Recherche at CNRS within the “Dependable Computing and Fault Tolerance” research group at LAAS-CNRS. He obtained a PhD in solid-state physics from the University of Toulouse, France in 1977. For a detailed description of JH Collet career and publications, see his home page at the URL: http://www.laas.fr/~collet/. JH Collet first worked in the field of the Optical spectroscopy of Semiconductors in the steady state, in the picosecond and femtosecond regimes. From 1988, he progressively switched his investigations to the field of the Optical Processing (optical automata) and Photonic Architectures for Multiprocessor Systems. He worked on communications in symmetric multiprocessors and on the integration of Optical Interconnects in this context. From 2002, J.H. Collet has begun to work in the field of dependable computer architectures at LAAS in Toulouse. JH Collet also supervised the writing of a generic multiagent simulator, to analyze self-organization and emergence in the populations of faulty repulsive agents. J.H. Collet worked in St Petersburg with the Ioffe Institute in 1980, with IBM in 1981 (in Yorktown Heights), then periodically with the Max Planck Institute for Solid-State Physics in Stuttgart in the nineties. He was the Scientific Manager of several contracts with the General Armament Delegation (DGA) of the French Ministry of Defence, with the European Communities and with the Administrative District of Midi-Pyrénées. JH Collet is the author (or co-author) of 77 publications in international peer-reviewed journals, and 42 communications in international conferences, addressing the fields of theoretical solid state physics, ultra-fast spectroscopy, optical communications, non-linear optics, cache coherence in multiprocessors, multigrids systems, dependability in multiprocessors. His publications have been cited about 1000 times by different research groups as reported in the Science Citation Index database.

Mihalis Gizopoulos received his Engineering Diploma from the Computer Engineering and Informatics Department, University of Patras, Greece (1992), and his PhD from the Department of Informatics and Telecommunications, University of Athens, Greece (1997). He is Associate Professor at the Department of Informatics, University of Piraeus, Greece where he leads the Computer Systems Laboratory. His research interests include microprocessor-based systems dependable design, testing, on-line testing and fault tolerance. Dimitris Gizopoulos has published more than 100 papers in peer reviewed transactions, journals and conference proceedings, is inventor of a US patent, author of a book and editor of a second in test technology. He is Associate Editor of IEEE Transactions on Computers, IEEE Transactions on VLSI Systems, IEEE Design & Test of Computers Magazine, and Springer’s Journal of Electronic Testing: Theory and Applications, as well as Guest Editor for several Special Issues. Dimitris Gizopoulos is involved in the organization and technical program of many international conferences. The period 2004-2008 he was member of the Steering Committee of the IEEE International Test Conference (ITC) and the period 2001-2006 member of the Steering Committee of the IEEE European Test Symposium (ETS). He was General Chair of the IEEE European Test Workshop 2002, the IEEE International On-Line Testing Symposium 2003, and is the General Chair of the IEEE International Symposium on Defect and Fault Tolerance 2009. He was Program Chair of the IEEE International On-Line Testing Symposium 2007, 2008 and 2009 and the IEEE International Symposium on Defect and Fault Tolerance 2008 and is the Program Chair of the IEEE International On-Line Testing Symposium 2010. He is also member of the Organizing and Program Committees of several IEEE conferences. He is member of the Executive Committee of the IEEE Computer Society Test Technology Technical Council (TTTC) with contributions to the Technical Meetings and the Tutorials and Education Groups. Dimitris Gizopoulos is a Senior Member of IEEE, a Golden Core Member of IEEE Computer Society.