Functional Testing with STLs: A Step Towards Reliable RISC-V-based HPC Commodity Clusters

Functional Testing with STLs: A Step Towards Reliable RISC-V-based HPC Commodity Clusters*

Josie E. Rodriguez Condia[1][0000−0001−5957−5624], Nikolaos I. Deligiannis[1][0000−0002−7948−3361], Jacopo Sini[1][0000−0002−2163−9925], Riccardo Cantoro[1][0000−0002−1745−5293], and Matteo Sonza Reorda[1][0000−0003−2899−7669]

Politecnico di Torino
Department of Control and Computer Engineering (DAUIN), Turin, Italy,
{josie.rodriguez, nikolaos.deligiannis, jacopo.sini, riccardo.cantoro, matteo.sonzareorda}@polito.it

Abstract. The reliability of High-Performance Computing (HPC) systems is an essential concern due to their massive size and the complexity of their operation. Thus, functional tests have been extensively used to monitor HPC systems and use software routines to verify the software stack's operation, mainly focusing on high-level abstraction features. However, the miniaturization of transistor technologies and the increment of computational resources (to face the performance and computation capabilities of HPC systems for the exascale generation) impose new reliability challenges that involve the development of clever testing strategies considering the underlying hardware characteristics. Interestingly, resorting to open-hardware architectures (such as RISC-V-based platforms) in the HPC domain offers a unique opportunity to effectively combine traditional HPC functional testing techniques with the adoption of effective fine-grain hardware testing solutions, such as those based on the Software-Based Self-Test (SBST) strategy.

This work proposes the SBST strategy as an enhanced and complementary technique for functional testing of RISC-V platforms for HPC systems. The method provides fine-grain evaluations of the CPU cores, including quantitative information on the state of the CPU cores and the presence of faults. For the experiments, we resort to two RISC-V cores (`RI5CY` and `ibex`) to develop and verify the effectiveness of the SBST strategy. In total, we developed 11 STLs (SBST routines) showing that a considerable percentage of hardware faults (from about 82% and up to 90%) can be detected with minimal overhead, thus, allowing their use during empty time intervals or in combination with other in-field functional testing approaches for HPC clusters.

Keywords: High-Performance Computing · Open-hardware · Reliability · RISC-V architecture · Software-Based Self-Test (SBST).

* This work has been supported by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

1 Introduction

Current High-Performance Computing (HPC) systems exploit parallelism to handle massive data and application complexity by distributing tasks among the available commodity clusters. This distribution provides very high levels of performance and throughput in the system. Moreover, the current demand for exascale computing capabilities involves scaling the size of HPC machines and exploiting the latest technology node approaches to increase performance and computational power, meanwhile reducing power and energy budgets. Unfortunately, new integration technologies are highly prone to system failures due to hardware faults in the components (e.g., permanent faults produced by premature aging), so imposing new reliability challenges[21][23]. Thus, the development of effective testing mechanisms for the in-field operation of HPC machines and their internal components are of major importance for an HPC system to achieve required reliability thresholds [18], as well as to reduce possible economic impacts by ineffective uses of HPCs due to system anomalies.

Traditional testing strategies to verify the correct operation of HPC's commodity clusters usually consists of several (hundreds) software procedures intended to identify possible software errors and hardware faults affecting the system. These routines are applied to determine the current operative state of commodity clusters and their interconnections, as well as to support the subsequent correction stages of the system. Most testing procedures focus on the high-level abstraction of the HPC and mainly target the software stack (e.g., *scheduler*, *compilers*, and *libraries*). However, the current reliability challenges (e.g., the premature rising of permanent hardware faults) exacerbate the need for clever, focused, extensive, and exhaustive test routines considering the underlying hardware architecture in the system (e.g., in CPUs and hardware accelerators) to guarantee sufficiently low error rates [11], [22], [14].

Current techniques include profiling and *regression testing* [24] as essential steps for the integration of software/application into HPCs. Other test approaches involve several stages and some of these consider the underlying hardware as part of their test objectives. In [27], the authors describe an HPC *acceptance* software test comprising two stages: *i) Hardware Acceptance Testing* (HAT) and *ii) Final Integration Testing* (FI). HAT consists of hardware diagnostics (usually performed by manufacturers or integrator companies) to ensure that each component (e.g., processors, memory, interconnect) meets the required operational specifications for the HPC system. FI includes *Functionality Testing* to ensure the correct operation of the software stack, *Performance Testing* that checks the achievement of the nominal performance and the execution and scale of applications, and *Stability Testing* focused on verifying the status of the software layers and the execution of diverse workloads continuously and for an extended period on top of the system. Other typical testing approaches (e.g., sanity checks, such as Node Health Check (NHC) [38], Nagios [4], Ganglia [30], and benchmarks [29]) target functional and operational features in the HPC machine, such as the performance or the variation in precision in the compo-

nents (e.g., Variety [26]). Unfortunately, all the previous approaches neglect the exhaustive evaluation and test of the hardware in HPC commodity clusters.

In contrast, HAT tests focus on the functionality of the system's components, and are mainly used during HPC setup, configuration steps, or after major changes in their compositions (e.g., hardware updates). However, these tests require structure information of the hardware, which is usually protected and only available to the HPC manufacturer or system integrator (e.g., *Intel*, *Nvidia*, or *AMD*). Moreover, the quantitative HAT test's fault coverage is **barely** specified, possibly limiting their effectiveness for all components. It is worth noting that hardware-focused tests are not usually used for the production stage of HPCs.

Modern exascale HPC design strategies are based on co-design schemes to optimize performance and power consumption by the direct collaboration between hardware designer and system integrator companies, which can also support the development of more effective reliability mechanisms. Given the importance of the underlying hardware in HPC commodity clusters, effective alternatives of hardware-oriented and in-field tests, such as the Software-Based Self-Test (SBST) strategy [32], might contribute to increasing the detection of failures caused by faulty hardware. The SBST is a non-intrusive and effective near-structure strategy for the test of processor-based systems since it exploits their on-chip resources to run (at speed) and self-test the correct operation of their components (e.g., peripherals, hardware accelerators, and memories) [25],[2],[32],[34],[10]. Moreover, the SBST strategy can focus on the unit's structure and identify low-level hardware faults. The strategy focuses on the development of compacted software Test Programs (TPs) to compose libraries (*Self-Test Libraries* or STLs) using the native instruction sets of a target processor [17], [16]. Furthermore, the SBST strategy has been successfully used as a complementary in-field testing mechanism for safety-critical applications [5].

This work proposes the use of the SBST strategy to develop efficient STLs for the test of permanent (stuck-at) faults in RISC-V-based cores used in the HPC domain. The proposed method focuses on the main CPU core of an HPC cluster. For this purpose, we considered two RISC-V processors as a case study (`RI5CY` and `ibex`), and we developed 11 STLs (one per targeted sub-unit). All test routines for the cores were implemented using RISC-V assembly instructions according to their base-ISA implementation and extensions. The current adoption of open hardware into HPC systems, such as the RISC-V architecture, offers an excellent opportunity for adopting the SBST strategy and developing efficient and effective STLs considering the available access to the underlying architecture of the components in RISC-V-based HPC clusters, which can then be combined with other tests for the in-field (production) test of the system. These additional detection capabilities can support fine-grain anomaly detection on commodity clusters and reduce impacts in terms of job failures and their associated economical costs for the HPC system operators.

The paper is organized as follows. Section II overviews the main functional testing strategies for HPCs and introduces the SBST strategy. Section III describes the proposed methods and algorithms to adapt the SBST strategy to

the functional test of RISC-V CPU cores. Section IV presents the experimental results and the effectiveness of the developed algorithms for functional testing. Finally, Section V provides the main conclusions and outlines some future works.

2 Related Works of Software-based Testing Strategies

2.1 Related works

In the HPC domain, several functional testing techniques have been developed to monitor the component's operation in commodity clusters. Most of them are based on the smart deployment of software programs to identify failures and check the system's operation, collect information, and determine system features, such as power consumption, performance, occupancy, and usage at any layer of an HPC. Most of them focus on the software stack using conventional testing approaches (e.g., regression testing) [15],[24] that are customized (ad-hoc) to support the identification of functional and non-functional failures, especially when performing system's enhancements or configuration changes. Other tests resort to machine learning approaches and autoencoders [6]. Similarly, software approaches (e.g., *Functionality, Performance* and *Stability Testing*) check the performance achievement of applications in terms of scale [36][39], and the stable execution of several applications into the HPC resources [31]. Similarly, sanity checks [38],[4],[30], and benchmarks [29] are complementary tests and mainly target operational and functional features in the system [26].

Unfortunately, most works focused on software infrastructures to test the system and neglected the underlying hardware features of the clusters. Interestingly, analyses on real HPCs revealed that a major percentage (from 53% to 64%) of anomalies are directly associated with malfunctions in the hardware of HPCs [35], so exacerbating the need for testing mechanisms using the underlying hardware architecture to achieve low error rates [11].

Some functional tests, such as the *Hardware Acceptance Test* [27][37], already consider the underlying hardware features as part of the tests. This test targets the hardware diagnostics to ensure that each cluster component (e.g., processors, accelerators, memory, and interconnect) meets the operational specifications. Unfortunately, these procedures are usually performed by manufacturers of system integrator companies, which possess or have access to detailed information from the hardware structures (usually protected or unavailable). Moreover, these tests are applied before the production state of the HPC system (i.e., during the configuration and setup steps) without clear information regarding their test coverage, possibly limiting their effectiveness in some machine components. In addition, such tests are barely deployed during the production stages of the HPC by restrictions on their time execution or the availability of effective and compacted hardware tests.

Hybrid mechanisms consider the smart combination of field sensors and hardware controllers (e.g., *Baseboard Management Controller* or BMC) with software infrastructures to monitor and test the in-field system's state by resorting to the

definition of *healthy* and *unhealthy* states from the collected in-field parameters. The authors in [3] resorted to compacted embedded systems devoted to monitoring and testing tasks. In addition, in [28], the authors proposed a structure to monitor HPC equipment based on the clever combination of BMCs and control software. In both cases, the need of additional hardware structures is required to collect the in-field parameters and identify any anomaly in the system.

To solve the previous issues, we propose the adaptation of functional testing solutions based on the SBST strategy to exploit its flexibility to extract and detect faults in the underlying hardware, while reducing the cost in terms of performance. The TPs and STLs developed using the SBST strategy have been proven in several domains that were effectively adapted and deployed. Moreover, the TPs using the SBST strategy can be ported and combined to other testing mechanisms, such as stability or performance testing that are normally employed during the in-field state of HPCs. It must be noted that TPs and STLs crucially target faults that cannot be detected nor identified by workloads or other high-level functional test strategies, but which can potentially affect or produce erroneous results.

2.2 The Software-Based Self-Test strategy

The SBST strategy is a non-intrusive and flexible approach for functional hardware testing in processor-based systems [9]. Indeed, the SBST strategy has been successfully adapted to the safety-critical domain as a vital and complementary strategy to monitor and evaluate fault effects (i.e., permanent faults) arising in hardware components [5]. This approach allows the functional test of the hardware components by developing compacted and efficient special TPs and building STLs. These STLs are deployed at speed on a target unit and are able to verify and identify a considerable amount of hardware faults. The STLs are composed of routines developed using purely machine instructions, high-level languages, or a clever combination of both. In particular, three approaches (*i)* automatic, *ii)* deterministic, and *iii)* custom) are mainly used for the development of TPs. The automatic approach resorts to special algorithms to analyze the structure of a target hardware unit and identifies the possible patterns activating most faults. Then, the patterns are converted into equivalent instructions to build routines and programs [13], [12]. A variation might include the use of random and pseudo-random patterns to increase hardware fault detection. Both techniques can be complemented by software compacting algorithms to reduce the size and improve execution performance while providing acceptable fault coverage [33], [19]. The deterministic approaches exploit well-known algorithms to generate TPs and address specific structures (e.g., controllers, schedulers, or the register file). In contrast, custom approaches use structural details of the underlying hardware to determine the most efficient combination of instructions for the functional test of a unit (e.g., ALU).

The SBST strategy resorts to software-signature mechanisms, that propagate through software, to one or more structural observable points (e.g., *memory results*). The accumulative use of TPs allows the identification and coverage of

most structures from a system. Interestingly, the adoption of the SBST strategy into the HPC domain requires access to the structural features of the underlying hardware that is not typically available for HPC machines. In the case of an open-source RISC-V-based cluster node, the internal structural details are available, so providing an opportunity to effectively adapt the SBST approach in HPC machines. As can be depicted in Figure 1, the SBST strategy is closer to hardware than other typical functional test strategies.

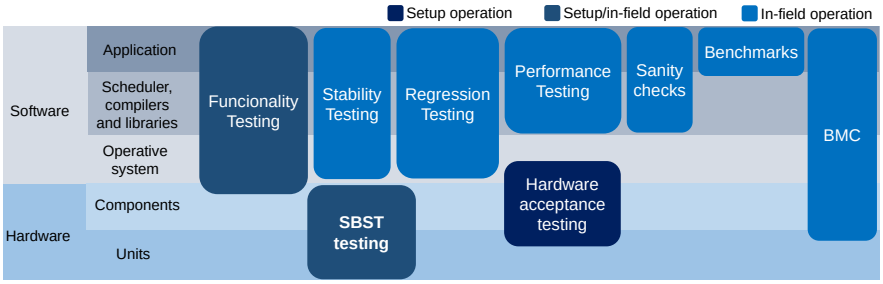

Fig. 1. A general scheme of the different functional testing strategies for HPC machines.

3 Testing RISC-V-based cores with STLs

The method uses a clever combination of several strategies to develop software test routines and allow the development of STLs to support the functional test of permanent faults (*Stuck-at*) affecting structures in RISC-V cores. The proposed approach is divided into five steps: *i)* Structural analysis of the core and definition of the main test targets, *ii)* Selection of test strategies, *iii)* Development and implementation of test programs, *iv)* Evaluation, and *v)* Validation and fitting of test programs, as depicted in Figure 2.

3.1 General idea

Our approach exploits a bottom-up scheme to analyze the available open structure/descriptions of RISC-V-based cores (e.g., core/cluster of cores) and then focus on each component for the development of test routines resorting to the SBST strategy. The method is applied to every unit to cover the complete system by accumulating TPs among the units of the system. For this purpose, the strategy divides the target units (e.g., functional units, logic units, and controllers), considering the main functional operation and the availability of machine instructions, routines, or functions to address them. Considering the type of units, we select one or more SBST strategies to develop feasible TPs. The TPs provide effective test patterns and propagate any effect in case of faults in the hardware. Finally, a development-flow framework supports the evaluation and validation of

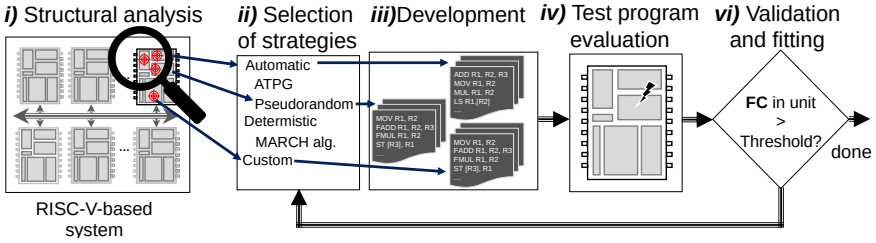

Fig. 2. An scheme of the method to adopt SBST strategies into RISC-V-based cores.

TPs on the gate-level implementations of the units, and quantifies the effectiveness of test coverage for each TP on the target structures. The next subsections describe the steps of the proposed approach of testing.

3.2 Structural analysis

First, we analyze the target unit's structure and its operational features (functionality and interaction with the system) to identify feasible strategies using machine instructions and possible routines to excite hardware faults inside the unit and propagate their effects (e.g., a fault in the ALU might be activated and propagated by one or a set of logic instructions). This step provides, as the main outcome, the sub-units identification.

3.3 Selection of test strategies

Table 1. SBST types and candidate units for the development of TPs.

SBST	Type	Unit							
		Functional units	Fetch unit	Decode unit	Controllers	Schedulers	Local memories	Register files	Interconnections
Automatic	*ATPG-based*	✓							
	Pseudorandom		✓	✓	✓	✓			
Deterministic	*March algorithms*						✓	✓	✓
	Others				✓	✓			
Custom		✓	✓	✓	✓	✓	✓	✓	✓

This step correlates the hardware sub-units with the feasible functional test strategies. We analyze the sub-units to determine which test strategy might offer better coverage under reasonable efforts. The test strategies can be divided into *i)* automatic, *ii)* deterministic, and *iii)* custom, as previously introduced in Section 2.2. Table 1 shows the relation between the strategies and different sub-units from a processor-based system.

In particular, automatic methods are effective on testing *functional*, *Fetch*, *Decode*, and *controller* units. The ATPG-based approaches usually resort to testing routines created using commercial Automatic-Test Pattern Generators

(ATPG) tools and post-processing wrappers that are mainly employed to analyze the structure of a sub-unit and determine feasible test patterns to map as equivalent instructions or values (i.e., to support the fault activation and propagation). This approach is effective when the complexity of the unit limits the use of deterministic approaches. Other approaches consist of pseudo-random testing algorithms (e.g., based on *Linear-feedback shift registers* or *LFSRs*) that produce pseudo-random patterns that can be used as values and operands on data-path sub-units, such as functional units. Similarly, the pseudo-random technique can generate random instructions for the later building of TPs. On the other hand, deterministic approaches employ well-defined algorithms to address regular hardware structures (e.g., register file and memories), and units following deterministic operations, such as embedded controllers. Among the deterministic approaches, MARCH algorithms [20] focus on the execution of well-defined operations (e.g., sequence of *writings* and *readings*) for the functional test of memories. Another approach focuses on the custom development of one or more specific routines targeting the functional testing of the sub-units of a system. This approach directly employs the functional operation and their constraints to select the best instructions able to provide effective fault detection. Moreover, this method can be applied to any sub-unit but it is used to require considerable timing efforts in their development. Moreover, custom approaches provide lower scalability (these can hardly be reused into similar structures on different cores). Interestingly, most custom approaches are hybrid combinations of manual, automatic, and deterministic procedures.

Each developed TP integrates a software signature (e.g., one or more characteristic values produced by the correct operation of the test program) to indicate when a given hardware unit has correctly executed a routine and no faults were detected. A mismatch in the signature after the execution of the test programs usually indicates the propagation of a hardware fault that allows its detection from the software. For each unit, one or more software strategies are defined as candidates for developing the test procedures. The main idea is to identify at least one strategy to apply to each target system sub-unit.

3.4 Development and implementation of test programs

We use a development-flow framework to support the TP generation and develop one or more routines for functional testing on each targeted sub-unit. Cross-compilation schemes support the translation from algorithms into valid machine instructions for a target RISC-V core processor. Then, a golden and fault-free micro-architectural logic simulation is performed with the purpose of collecting the signals and the TP's behavior when running on hardware. This information is employed in the next step to support the evaluation and validation of TPs.

3.5 Evaluation

The third step targets the evaluation of TPs resorting to an automatic framework that injects and evaluates the fault detection features of each TP through a

sequence of fault injection campaigns (i.e., a procedure to inject one hardware fault per sub-unit and then evaluate the TP's behavior). For this purpose, we resort to one commercial grade parallel logic simulator tool to inject permanent (stuck-at) faults and identify propagation and detection effects on every targeted sub-unit of a RISC-V core. In the simulation process, the framework evaluates the complete processor core and employs the interconnections with the main memory to determine the propagation and detection of faults. It must be noted that for the evaluation, we inject only one permanent fault per simulation to the gate-level description of the sub-unit under test.

3.6 Validation and fitting of test programs

In this case, the complete set of TPs is executed in the target hardware unit. the automatic framework determines the TP fault coverage and evaluates a quality threshold (e.g., a percentage of fault identified as detected). When the threshold is achieved, a new TP is targeted for development and evaluation. Otherwise, improvement and changes on TPs are performed for new evaluations. It must be noted that the fourth step is optional, but it is common that several iteration steps are required to describe an efficient and effective TP for a given sub-unit.

4 Experimental results

To evaluate and validate the proposed approach, we target the functional testing of permanent hardware stuck-at faults in the internal structures of the individual cores inside the commodity clusters. We consider that memories (e.g., cache or main RAM) and the interconnect infrastructures are protected by one or several levels of fault detection and mitigation mechanisms (e.g., Error-Correcting Codes or ECCs) and are not of direct interest to the functional testing targets.

For the evaluation of the SBST strategy, we employ two representative implementations of RISC-V architectures (*IBEX* and the *RI5CY*). *RI5CY* uses 4 in-order pipeline stages at 32-bit. The ISA of RI5CY was extended to support multiple additional instructions including hardware loops, post-increment load and store instructions and additional ALU instructions that are not part of the standard RV ISA. RI5CY has become a popular core for a huge variety of applications, especially for IoT designs. Similarly, the *Ibex* is a production-quality open source 32 bit RISC-V CPU core written in SystemVerilog. The CPU core is heavily parametrizable and well suited for embedded control applications. *Ibex* is being extensively verified and has seen multiple tape-outs. For the experiments, the processors' RTL `SystemVerilog` description was synthesized using the *Silvaco* 45nm Open Cell Library [1] using `Design Compiler` by Synopsys. All the experiments were performed on a server with 12 Intel Xeon CPUs running at 2.5 GHz and 256 GB of RAM.

4.1 Adapting the SBST strategy

For both RISC-V cores, we divide their internal organizations to use a bottom-up approach and develop independent and focused TPs for each sub-unit. It must be noted that similar sub-units might reuse a given SBST strategy, but it must be adapted to the specific underlying architecture of the targeted RISC-V core.

The *IBEX* core was divided into five main targets for testing (*Fetch*, *Decode*, *Execution*, *Load-store*, and local memories (register file and control status registers). Similarly, the *RI5CY* core is divided into six testing targets (*Fetch*, *Decode*, local memory hierarchy or register file, *Execution* that is internally divided as ALU and multiplier unit, *Load-store* unit, and *control-flow* controller).

4.2 Development flow

We use an incremental approach to tune the best trade-off of fault coverage for each sub-unit. For some units, we combine more than one test strategy and obtain acceptable fault coverage.

As depicted in Table 1, some test strategies can be used to test internal units in processors. In the proposed approach, we identify that pseudo-random strategies are simple and effective, hence we employ a software-based LFSR approach to target the *Fetch* and *Decode* units. Moreover, we use a MARCH algorithm (MATS+) to target the local memory units. MATS+ ($\uparrow\downarrow$ (w_0); ($\uparrow$ (r_0), $\downarrow$ (w_1)); $\downarrow$ (r_1,w_0)) provides effective test coverage when applied into all registers/cells of a memory. For other units, we include pseudo-random and custom strategies to allow the execution of the instructions and propagate fault effects by resorting to software signatures.

In the evaluation experiments, each developed TP is evaluated through a fault injection campaign aiming to evaluate and verify the effectiveness of the implemented test strategies on both RISC-V cores. A total of 11 fault injection campaigns evaluate 125,196 and 162,146 permanent faults in the IBEX and RI5CY cores, respectively.

For the experiments, we resort to one commercial fault simulator tool (in our case *Z01X* by *Synopsys*) to perform the incremental evaluation and verify the operation of each developed TP on each RISC-V core implemented at gate-level. Table 2 summarizes the main features of the targeted RISC-V cores and their sub-units. Regarding the STL testing effectiveness, Figure 3 depicts the coverage of permanent faults on each sub-unit and the overall for both cores.

Table 2. Number of hardware faults per sub-unit in the evaluated RISC-V cores.

	Units	**Fetch**	**Decode**	**Execute**	**LS**	**Mem.**	**CF**	**Overall**
Number of Faults	***IBEX***	12,278	12,616	45,832	4,420	50,050	-	125,196
	RI5CY	12,024	65,582	37,838	5,608	39,590	1,504	162,146

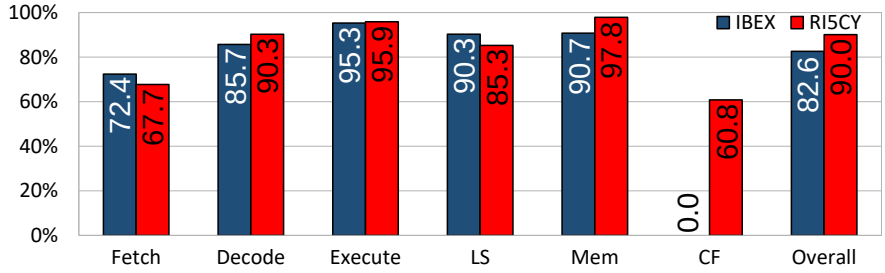

Fig. 3. Fault Coverage (FC) results for the sub-units (*Fetch*, *Decode*, *Execute*, *Load-Store* or 'LS', *Register File* or 'Mem', and *Control-Flow* 'CF') in both RISC-V cores.

Some of the developed TPs perform better for some sub-units (e.g., *Execute*, *Load-Store* 'LS', and the register file '*MEM*'). Interestingly, all previous units are mainly part of the data-path of the processor cores and directly interact with the machine instructions, so an extensive coverage is expected. In contrast, other units, such as the *Fetch* and the *Decode* obtained a slightly lower fault coverage. Unfortunately, since these units directly interact with the main memories (the *Fetch* case) and with any possible instruction format (*Decode* case), the requirements of the test program require of manual support for their development. i.e., in the case of the *Fetch* unit, several faults remain untested due to the limit to address high sectors of the memory resources. Similarly, for the *Decode* unit, the implemented strategies do not include all possible instructions, so some faults associated with those instructions are not identified.

The execution times of the set of TPs per RISC-V core (109,194 and 80,455 for the IBEX and RI5SCY, respectively) denote the different operative times required to test faults in the cores. In principle, complete in-field testing sequences might be deployed periodically in available time slots. In the results, we focused on the sub-units of one processor core. However, the same strategies can be scaled from one up to several core units in cluster configurations. Thus, an equivalent percentage of faults covered is expected in homogeneous clusters.

The observed results only consider the internal logic of the cores. Interestingly, the results show that independently of the organization of the RISC-V core, the selected and implemented SBST techniques and test programs can effectively detect (around 82% in *IBEX* and about 90% in *RI5CY*) of the possible permanent hardware faults arising inside the sub-units. Moreover, as observed for most sub-units in both RISC-V core processors, equivalent TPs provide acceptable and similar levels of fault coverage in the *Fetch*, *Decode*, *Execute*, *LS*, and *Mem* (from about 0.6% up to 6.9% of difference).

The reported fault coverage results focused on permanent stuck-at faults and do not consider possible functionally untestable (or *safe*) faults in the sub-units, which by definition cannot produce any failure in the considered operational scenario. Thus, the percentage of fault coverage per sub-unit can significantly increase after identifying the untestable set of faults in the core by resorting

to structural analysis and automatized methods [16], [7]. Interestingly, for some units, the proposed STLs are effective even when considering different organizations in the target processors. In the case of the *Decode* unit, the implemented test programs were effective to reduce the difference between the fault coverage percentages on both processor cores (lower than 5%), even considering that the *RI5CY* implementation includes ISA extensions.

The flexibility of the SBST technique supported and allowed the adaptation and development of Testing mechanisms for different structural organizations of RISC-V cores. In particular, in case of the RISC-V cores, the standard ISA for both cores contributes to identify and implement common software routines considering the underlying structures for each sub-unit. This means that similar approaches can be used to target more complex RISC-V-based core processors specially conceived for the HPC domain (e.g., including SIMD or Vector extensions). It must be noted that additional effort is expected to extend a test strategy for the additional features of a processor core.

Finally, the results support the claim that SBST strategies can be used as complementary mechanism for the in-field testing of the elements in a HPC system and deploy them in combination with other tests (e.g., functionality, or performance testing). Similarly, we are already working on extending the same strategy to support the testing of other fault phenomena arising in the hardware components of processor-based systems, such as path delay faults [8].

5 Conclusions and Future Work

In this work, we elaborate on the use of the SBST strategy to perform fine-grain functional testing of the internal units of RISC-V-based processors. The flexibility of the SBST strategy allows the development of compact and effective test programs to address the permanent hardware faults possibly affecting the units of processors used in HPCs. The reported results prove that these special testing programs can be effectively used as complementary mechanisms to those already used in HPC systems during the setup/configuration and during in-field operation. Moreover, the experimental results demonstrate that a combination of different testing strategies can provide acceptable levels of stuck-at-fault coverage (from around 82% to 90%) for the internal units of RISC-V-based processors. These figures are in line with those required for highly safety-critical applications in other domains (e.g., automotive), taking into account that a significant percentage of the remaining faults are likely to belong to the class of *safe faults* (i.e., cannot produce any failure).

In the future, we plan to adapt SBST strategies into shared resources in commodity clusters, such as hardware accelerators, memory resources, arbiter controllers, and on-chip and off-chip interconnect infrastructures. We are also working on STL extensions on fault models describing delay defects, which are known to be produced by semiconductor aging phenomena.

References

1. Silvaco 45nm Open Cell Library. https://si2.org/open-cell-library, [Online; accessed 17-Mar-2022]
2. Apostolakis, A., et al.: Software-based self-testing of symmetric shared-memory multiprocessors. IEEE Transactions on Computers **58**(12), 1682–1694 (2009)
3. Baghyalakshmi, D., et al.: Wsn based temperature monitoring for high performance computing cluster. In: 2011 International Conference on Recent Trends in Information Technology (ICRTIT). pp. 1105–1110 (2011)
4. Barth, W.: Nagios: System and network monitoring. No Starch Press (2008)
5. Bernardi, P., et al.: Development flow for on-line core self-test of automotive microcontrollers. IEEE Transactions on Computers **65**(3), 744–754 (2016)
6. Borghesi, A., et al.: Anomaly detection using autoencoders in high performance computing systems. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 9428–9433 (Jul 2019)
7. Cantoro, R., et al.: An analysis of test solutions for cots-based systems in space applications. In: 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC). pp. 59–64 (2018). https://doi.org/10.1109/VLSI-SoC.2018.8644846
8. Cantoro, R., et al.: New perspectives on core in-field path delay test. In: 2020 IEEE International Test Conference (ITC). pp. 1–5 (2020)
9. Chen, L., Dey, S.: Software-based self-testing methodology for processor cores. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **20**(3), 369–380 (2001)
10. Condia, J.E.R., et al.: Using stls for effective in-field test of gpus. IEEE Design & Test **40**(2), 109–117 (2023)
11. DeBardeleben, N., et al.: Gpu behavior on a large hpc cluster. In: Euro-Par 2013: Parallel Processing Workshops. pp. 680–689. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
12. Deligiannis, N.I., et al.: Automating the Generation of Programs Maximizing the Repeatable Constant Switching Activity in Microprocessor Units via MaxSAT. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2023)
13. Deligiannis, N.I., et al.: Automating the Generation of Programs Maximizing the Sustained Switching Activity in Microprocessor units via Evolutionary Techniques. Microprocessors and Microsystems **98** (2023)
14. Dixit, H.D., et al.: Silent data corruptions at scale. CoRR **abs/2102.11245** (2021), https://arxiv.org/abs/2102.11245
15. Evans, T., et al.: Comprehensive resource use monitoring for hpc systems with tacc stats. In: 2014 First International Workshop on HPC User Support Tools. pp. 13–21 (2014)
16. Faller, T., et al.: Constraint-based automatic sbst generation for risc-v processor families. In: 28th IEEE European Test Symposium (ETS2023), to be apear. pp. 1–6 (2023)
17. Faller, T., et al.: Towards SAT-Based SBST Generation for RISC-V Cores. In: 2021 IEEE 22nd Latin American Test Symposium (LATS) (2021)
18. Gomez, L.B., et al.: Gpgpus: How to combine high computational power with high reliability. In: 2014 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 1–9 (2014)

19. Guerrero-Balaguera, J.D., Rodriguez Condia, J.E., Reorda, M.S.: A novel compaction approach for sbst test programs. In: 2021 IEEE 30th Asian Test Symposium (ATS). pp. 67–72 (2021). https://doi.org/10.1109/ATS52891.2021.00024
20. Hamdioui, S., et al.: March ss: a test for all static simple ram faults. In: Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002). pp. 95–100 (2002)
21. Hamdioui, S., et al.: Reliability challenges of real-time systems in forthcoming technology nodes. In: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 129–134 (2013)
22. Hochschild, P.H., et al.: Cores that don't count. In: Proc. 18th Workshop on Hot Topics in Operating Systems (HotOS 2021) (2021)
23. IEEE: The international roadmap for devices and systems: 2022. In: Institute of Electrical and Electronics Engineers (IEEE) (2022)
24. Karakasis, V., et al.: Enabling continuous testing of hpc systems using reframe. In: Tools and Techniques for High Performance Computing. pp. 49–68. Springer International Publishing, Cham (2020)
25. Kranitis, N., et al.: Software-based self-testing of embedded processors. IEEE Transactions on Computers **54**(4), 461–475 (2005)
26. Laguna, I.: Varity: Quantifying floating-point variations in hpc systems through randomized testing. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 622–633 (2020)
27. Larrea, V.G.V., et al.: Towards acceptance testing at the exascale frontier. In: Proceedings of the Cray User Group 2020 conference (2020)
28. Li, J., et al.: Monster: An out-of-the-box monitoring tool for high performance computing systems. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER). pp. 119–129 (2020)
29. Luszczek, P., et al.: Introduction to the hpc challenge benchmark suite (4 2005)
30. Massie, M.L., et al.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing **30**(7), 817–840 (2004)
31. Pedicini, G., Green, J.: Spotlight on testing: Stability, performance and operational testing of lanl hpc clusters. In: State of the Practice Reports. SC '11 (2011)
32. Psarakis, M., et al.: Microprocessor software-based self-testing. IEEE Design & Test of Computers **27**(3), 4–19 (2010)
33. Riefert, A., et al.: A flexible framework for the automatic generation of sbst programs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **24**(10), 3055–3066 (2016)
34. Sabena, D., et al.: On the automatic generation of optimized software-based self-test programs for vliw processors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **22**(4), 813–823 (2014)
35. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. IEEE Transactions on Dependable and Secure Computing **7**(4), 337–350 (2010)
36. Sickinger, D., et al.: Energy performance testing of asetek's rackcdu system at nrel's high performance computing data center (11 2014)
37. Smara, M., Aliouat, M., Pathan, A.S.K., Aliouat, Z.: Acceptance test for fault detection in component-based cloud computing and systems. Future Generation Computer Systems **70**, 74–93 (2017)
38. Sollom, J.: Cray's node health checker: An overview. In: Proceedings of the Annual Meeting of the Cray Users Group-CUG-2011, Fairbanks, Alaska, USA (2011)

39. Tronge, J., et al.: Beeswarm: Enabling parallel scaling performance measurement in continuous integration for hpc applications. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1136–1140 (2021)