Chapter 1
Online Software-Based Self-Testing in the Dark Silicon Era

Mohammad-Hashem Haghbayan, Amir M. Rahmani, Antonio Miele, Pasi Liljeberg, and Hannu Tenhunen

Abstract Aggressive technology scaling and intensive computations have caused acceleration in the aging and wear-out process of digital systems, hence leading to an increased occurrence of premature permanent faults. Online testing techniques are becoming a necessity in current and near future digital systems. However, state-of-the-art techniques are not aware of the other digital systems' power/performance requirements that exist in modern multi/many-core systems. This chapter presents an approach for power-aware non-intrusive online testing in many-core systems. The approach aims at scheduling at runtime Software-Based Self-Test (SBST) routines on the various cores to exploit their idle periods in order to benefit the potentially available power budget and minimize the performance degradation. Furthermore, a criticality metric is used to identify and rank cores that need testing at a time and power and reliability issues related to the testing at different voltage and frequency levels are taken into account. Experimental results show that the proposed approach can i) efficiently perform cores' testing, within less than 1% penalty on system throughput and by dedicating only 2% of the actual consumed power, ii) adapt to the current stress level of the cores by using the utilization metric, and iii) cover all the voltage and frequency levels during the various test.

Mohammad-Hashem Haghbayan
University of Turku, Turku, Finland, e-mail: mohhag@utu.fi

Amir M. Rahmani
University of Turku, Turku, Finland e-mail: amirah@utu.fi

Antonio Miele
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy, e-mail: antonio.miele@polimi.it

Pasi Lijeberg
University of Turku, Turku, Finland, e-mail: pasi.liljeberg@utu.fi

Hannu Tenhunen
KTH Royal Institute of Technology, Stockholm, Sweden, e-mail: hannu@kth.se

1 Introduction

The aggressive technology scaling in the fabricated chips has brought to the integration of several cores within the same chip. At the same time, the drawback of such a transistor shrinking has been an increase in the susceptibility of the digital circuits to internal defects, device variability and malfunction in execution units [1, 2]. As a matter of fact, aging and wear-out mechanisms, including time dependent dielectric breakdown (TDDB), negative bias temperature instability (NBTI), and electromigration (EM), are among the most increasingly adverse factors that can lead to timing errors and components' breakdowns causing system malfunctioning and, eventually, its overall failure. In addition, downscaling of CMOS technologies to the deep submicron levels has exacerbated the trend of high failure rate. This phenomenon has lead to an increased power densities and consequently operating temperatures in a device, being the main cause of the aging phenomenon. Thus, there is an increasing quest for reliability in modern computing systems.

In such a scenario, in order to handle such reliability quest, and in particular to detect and manage the occurrence of permanent failures in operational components, concurrent error detection and online testing may represent viable solutions. However, concurrent error detection is generally implemented by means of redundancy-based technique [3], such as duplication with comparison (DWC) or triple modular redundancy (TMR), which present a high cost due to area occupation. For this reason they are generally considered only in the design of systems specifically targeted for critical applications, such as the aerospace appliances where the cost is secondary concern. Similarly, within the online testing field, Built-in-Self-Testing (BIST, [4, 5]) circuitries are not commonly integrated in devices targeted to the consumer market, even if they present a more limited impact on the chip area with respect to the above discussed techniques. Another strategy for online testing is Software Based Self-Test (SBST, [6, 2]), which consists of periodic execution of specific testing routines devoted to the functional solicitation of the circuitry for the detection of permanent failures in the various execution units. Since such strategy does not require any additional circuitry (or, in some situations, a reduced one), it represents the most promising solution for consumer electronic devices. Indeed, an example of its large scale employment is in the automotive on-board computing systems [7].

Many-core systems fall under this umbrella of the digital devices requiring SBST [8]. In fact, they commonly do not feature any integrated hardware for online testing and, moreover, they are subject to a considerable stress caused by the intensive data-processing workload. However, this scenario presents two relevant issues:

1. The workload to be executed consists of applications requiring strict performance levels. This leads to the necessity of a *transparent test scheduling* because it is not possible to interrupt the nominal activities.
2. The system is characterized by a physical limit in the power budget which imposes that all the cores cannot be active at the same time at a full frequency.

At the opposite a relevant portion of them have to be put dark for power limits (such phenomenon is dubbed as *Dark Silicon* [9, 10]). This issue implies the necessity to consider also power consumption during the test scheduling, thus necessitating of a *power-aware testing approach*.

Hence, the employment of SBST in many-core systems presents at the same time new opportunities and challenges. We claim that in the scenario of the dark silicon era there is a quest for a *power-aware test scheduling approach* to detect at run-time permanent faults occurring in many-core architectures while not degrading the overall system performance.

An interesting and challenging aspect of modern many-core systems for test scheduling is the high dynamicity and heterogeneity of the executed workload. This makes the amount of dark area on the chip (i.e., total chip utilization) highly variable. Furthermore, due to the emergence of dim silicon [11] as a way to minimize dark areas and increase the number of active cores, the system might reach up to 100% utilization of its cores (if the majority of running application are not performance-demanding) by making use of power management features such as Dynamic Voltage and Frequency Scaling (DVFS) [12]. This makes the behavior of such systems to be highly related to the characteristics of the workload where at different moments of time it is possible to have considerable dark areas with small resource utilization, due to the fact that some other group of cores are set on a high voltage-frequency level thus reserving the majority of the overall power budget, or small dark areas with large resource utilization by globally setting a very low voltage-frequency level. Therefore, if suitable scenarios are intelligently identified (when there is enough room in power budget), such temporary dark areas can be favorable targets for online testing in order to improve the system reliability [13, 14].

Nevertheless, DVFS knobs introduce also other issues in the testing activity, as shown in the literature [15], systems should be tested at multiple voltage-frequency settings, since faults are manifested in different ways in different configurations. Therefore, the test scheduling needs to take into account the fact that SBST routines should be executed on the various cores in different voltage-frequency levels.

Given these motivations, this chapter presents an approach for a transparent power-aware online test scheduling in many-core systems in the dark silicon era. The proposed approach benefits from the large amount of cores and the available power budget to dynamically schedule SBST routines on the idle cores that have experienced a high stress in the recent past. In particular, the approach exploits a criticality metric, computed on the basis of a measure of the utilization of the cores, to select the units to be tested. Then, a test mapping and scheduling approach selects among the candidates the actual cores to be tested on the basis of two conditions: such cores must be idle (i.e., not currently involved in the execution of an application) and there must be some available power not currently used for the execution of the running applications. Further, the test scheduling approach selects also the optimal possible voltage-frequency settings to execute the SBST routine by considering the system's power budget.

The rest of the chapter is organized as follows. Section 2 reviews the related work discussing the limitations which motivate the work. In Section 3 the background on many-core systems is discussed, presenting also the considered architecture and application models. Then Section 4 describes suitable scenarios for online testing by means of a running example, showing power consumption and online testing issues. The proposed dark silicon aware online testing approach is presented in Section 5, while Section 6 proposes an enhancement to handle testing at different voltage-frequency levels. Section 7 discusses the experimental results presenting some statistics and a comparison against a state-of-the-art approach to demonstrate the effectiveness of the proposed approach. Finally, Section 8 draws the conclusions.

2 Related Work

Software-based Self Testing (SBST) has been known as a useful mechanism in recent studies on online testing as it can be applied easily without any need to extra hardware resource [16, 6, 2]. Furthermore, it has been used widely for online testing in multi-/many-core system testing recently [17, 8]. The main challenge in online testing in multi-/many-core systems is to minimize the overhead of testing mechanism on the overall system performance [18]. Several works have been presented in the literature studying the impact of online error detection on the performance of multi- and many-core systems [18, 19, 6, 20, 2, 8]. In [21], a SBST scheduling algorithm is proposed for testing cores at runtime while the system is working. In [22] the authors proposed online testing algorithm for many-core systems to achieve high fault coverage for both routers and PEs. In [23] and [24], a structural level process is presented to develop test softwares. In [4, 25, 26, 27] deterministic, random, and hybrid method were used for generating software tests. However, none of the state-of-the-art approach considers current available power budget while applying test process. In fact they are not power-aware.

Power-aware testing should not be confused with power-constrained testing. In power-constrained testing, the goal is to minimize the offline Test Application Time (TAT) by parallelization of testing the cores, e.g., using test access mechanisms, while avoiding peak power violation. Many studies have been done to achieve minimal TAT [28, 29, 5]. In fact, as the power consumption of the single core during test process is generally greater than that in normal operation mode [30], in power constrained testing the focus is on how to test the cores without damaging it due to thermal violation. However, in power-aware testing the target is to test the core(s) when the other cores are working in their normal mode. That is why a power feedback from the system is needed to know when we have enough power budget in runtime to test the cores.

As the fault model and testing strategy for multi-core and many-core systems with advanced dynamic power management features changes, recent studies is focused on proposing new techniques for testing such systems. In general these recent studies can be categorized into two groups 1) the techniques that considers the effect of such power management capabilities to new error manifestation and 2) strategies that get benefit such power management features to control the test power while TAT minimization process [31]. Most of these strategies have been proposed for offline testing purposes. Even though we can find a limited number of online testing methods in these two categories, they do not yet consider any power feedback from the system at runtime and instead use a pre-defined dedicated power budget for testing. An example of power-aware optimization of the SBST has been presented in [32], where the authors propose an optimal approach to test the L1 cache in microprocessors considering power profile. However, this work is not either fully power-aware as the authors use a pre-defined power model of the microprocessors for different applications, which lacks an online power feedback from the system.

Using SBST in online testing can be done in two different ways: intrusive and non-intrusive [2]. In intrusive online testing, test process is done during a fixed period in which the normal system operation is interrupted and the cores, or a subset of them, are reconfigured to *test mode* at runtime, and then run the test program. It can be concluded that, as in intrusive testing the normal operation of the system is interrupted, testing process might has negative effect on the performance of the system. On the other hand, in non-intrusive testing, each core executes the test program individually whenever the core is in idle state. As mentioned before, the power consumption needed for the test purpose is considerably higher than the power consumption of the system in the normal operation mode. As the power budget of the system is limited specially in the dark silicon era, it is not possible to perform a fully parallel intrusive testing as the power consumption can easily exceed the available budget and endanger the chip reliability. Furthermore, as the system is concurrently running multiple independent applications with different requirements, interrupting all or some of these might lead to deadline miss for some applications. Due to these facts, our focus will be on non-intrusive testing while honoring power budget.

Based on the above discussion, it can be concluded that *online testing is gradually reshaping to power-aware online testing in the dark silicon era, especially for many-core systems*. The main ground for this statement is that due to thermal and power constraints, the fraction of transistors that can operate at full frequency is decreasing with each technology generation. This highlights the fact that power budget is an extremely crucial resource in those technologies where the dark silicon phenomenon is more challenging to address (e.g., 22nm or 16nm). In such technologies, a many-core system demands an efficient power-aware online testing method capable of minimizing the usage of power for the online testing purpose. In other words, the online testing method should have the lowest negative impact on the system performance by efficiently using the power budget.

3 Adopted Many-core Architecture

Figure 1 shows an overview of the considered architectural platform and the above software stack, composed of a runtime management layer and a set of running applications.

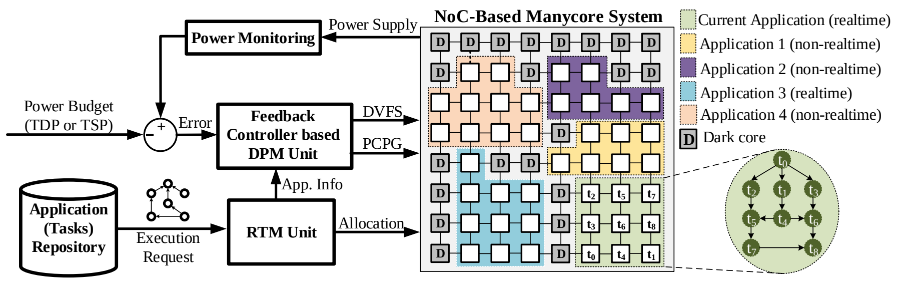

NoC-Based Manycore System
Power Monitoring
Power Supply
Power Budget (TDP or TSP)
Error
Feedback Controller based DPM Unit
DVFS
PCPG
App. Info
Application (Tasks) Repository
Execution Request
RTM Unit
Allocation
Current Application (realtime)
Application 1 (non-realtime)
Application 2 (non-realtime)
Application 3 (realtime)
Application 4 (non-realtime)
Dark core

Fig. 1: A NoC-based many-core system with mesh topology supporting DPM and RTM

The target platform is the classical many-core architecture, such as the Intel Single-chip Cloud Computer [33], or the Kalray MPPA manycore [34]. The architecture is composed of a set of homogeneous processing cores, each one provided with a private memory for instructions and data, and connected to the system's communication infrastructure. The communication infrastructure consists of an $M \times N$ 2D-mesh Network-on-Chip (NoC), using a message-passing protocol based on a X-Y deterministic wormhole routing schema.

The considered many-core architecture is generally adopted for the acceleration of intensive data-processing applications, such as image, video or audio processing. Each application is generally implemented with a pipelined dataflow paradigm [33]. Thus, the application can be modeled by means of a task graph, where nodes represent the various computation tasks, each one characterized by a specific execution time, and the direct edges represent data dependencies (in terms of data to be transmitted) between a source task to a target one.

In order to execute an application, each task is assigned, or *mapped*, to a specific core that will execute it. In other words, we may also say that a core is *allocated* for the execution of the task. Moreover, the execution model does not support multi-tasking, therefore at most one task can be mapped on a single core in a specific instant of time. As a motivation of this choice, Intel in 2011 [33] stated that, given the abundance of execution units in a many-core architecture, a one-to-one mapping may ease the execution management. Then, the mapped application is executed in a pipelined fashion: each core can perform a run of the hosted task each time it receives all required input messages and at the end of the execution it sends out the output messages. The NoC is in charge of routing and dispatching the messages from the senders to the receives. Actual transmission latencies will depend on the infrastructure operating frequency, the message size and the distance between the source and the target.

The right part of Figure 1 shows an example of the described system. The architecture is an 8×8 grid of cores, on which 5 different applications are mapped. In the detail in the right-bottom part of the figure, the task graph of a Gaussian Elimination application (retrieved from [35]) is shown.

The left part of Figure 1 shows the runtime management layer. This layer is a software module running on a controlling hardware unit, that may be a dedicated core in the NoC or an external host machine. The runtime management layer is composed of two main modules, called Runtime Mapping Unit and Dynamic Power Management Unit, that are discussed in the next.

The considered systems are generally employed in scenarios characterized by a highly variable workload. Indeed, applications arrive with an unknown trend depending on the requests of the various users. Moreover they may have different characteristics in terms of structure of the task graph and different requirements, for instance, on the minimum required throughput or on the amount of the processed data. As an intuitive example in Figure 1, applications are annotated with realtime/non-realtime requirements. In order to deal with this variable scenario, the runtime management layer contains a unit devoted to the runtime mapping (RTM). This unit receives the request of applications' execution arriving from the users, and maps them on the available cores by using a specific runtime strategy (e.g. [36]) to satisfy the specified performance requirements. It may also happens that in a certain instant of time there is no enough resources to run the newly-incoming application; in that case, the application will be delayed until it is not possible to satisfy its requirements.

On the other side, physical limits in circuit cooling, packaging, and power delivery in modern chips cause the many-core systems to have non negligible power issues, expressed in terms of a limited *power budget*. According to such power budget, only a part of the available cores can be used at the same time while the other ones have to be switched off, thus causing the *dark silicon* phenomenon. For instance, Figure 1 shows in gray the set of cores that are switched off (i.e., dark). Moreover, running applications may cause different power consumptions, depending on the number of allocated cores and the voltage/frequency levels at which cores work. This heterogeneity and the mentioned dynamicity in the workload will cause the amount of dark area to relevantly vary during the execution. In fact, in some situations, the allocated cores have to work at a high voltage-frequency level to provide the necessary performance in order to fulfill the required application throughput. Such cores will use a large part of the power budget, thus causing other units to be temporary set as dark. In other situations, it may happen that the set of applications to be executed do not demand high operating frequencies, and therefore it is possible to use the total chip utilization at a low voltage-frequency level, leaving no dark area on the chip. Such discussion motivates the necessity of a dynamic power management (DPM) within the runtime management layer.

Figure 1 shows the DPM Unit within the runtime management layer. Such a unit is connected to the RTM Unit to take coordinated decisions about the application mapping and power management. In particular, the aim is to achieve applications' performance requirements while respecting the power limit. The available budget is defined either at design time, by using the Thermal Design Power (TDP [9]), or dynamically managed at runtime with another feedback loop, by means of the Thermal Safe Power (TSP [37, 38]). Then, as in [39, 40, 41], the DPM Unit is implemented as a feedback-loop that monitors the consumed power by means of on-chip power sensors and acts on per-core DVFS and power gating knobs.

In conclusion, for each arrived application, RTM and DPM Units work together in order to decide if there is enough power budget available for the execution of such application, to define a mapping and a DVFS setting to achieve required performance while not violating the power budget. If such conditions are not satisfied, the application is delayed until some other application leaves the system and releases enough resources and power.

4 Suitable Scenarios for Online Testing

Nowadays, many-core system are generally employed for high performance computing in different fields spanning from data centers to high-end embedded and mobile appliances. All these scenarios are subject to highly varying workloads: different types of applications arrive with an unknown trend and are characterized by different performance requirements, variable amount of data to be elaborated, different request of processing resources and so on. Therefore, in each instant of time, the running workload will cause a different working configuration in the many-core system, in terms of the set of currently running applications, their actual mapping on the cores' grid, and related power consumption. Figure 2 depicts a taxonomy of the main working situations. For each situation, Figure 2 reports the allocated cores to different applications and idle cores, and, if any, the size of the new application requested to be mapped. Moreover, each subfigure reports also the related power consumption graph reporting the actual power consumption (with a solid line) and the given power budget (with a dashed line). In each of these situations, we have analyzed the possibility to perform a non-intrusive online testing on a selected candidate core. These scenarios are commented in the following paragraphs.

Scenario (a). At time t_1, three applications with strict performance requirements are running on the system. Due to the performance requirements, the active cores are set to a high frequency-voltage level, thus leading the overall power consumption to be too close to the available budget. Therefore the other cores are forced to be dark. In this case, although there are idle cores that can be tested without affecting the nominal activities of the system, there is not enough available power budget to be dedicated to online testing.

Scenario (b). At time t_2 eight applications with a performance requirements exactly fit on the available cores. In such a scenario, the low power requirement caused by each of the applications allows to use 100% of the available resources as the dim area. Consequently, even though there is available amount of power budget for the testing activity, in order to execute the SBST routine it would be necessary to intrusively interrupt one of the running applications. However, this violates our goal of transparency.

Scenario (c). At time t_3, the system is almost unloaded, since a few applications are running and the power consumption is quite low. This is the best scenario, since

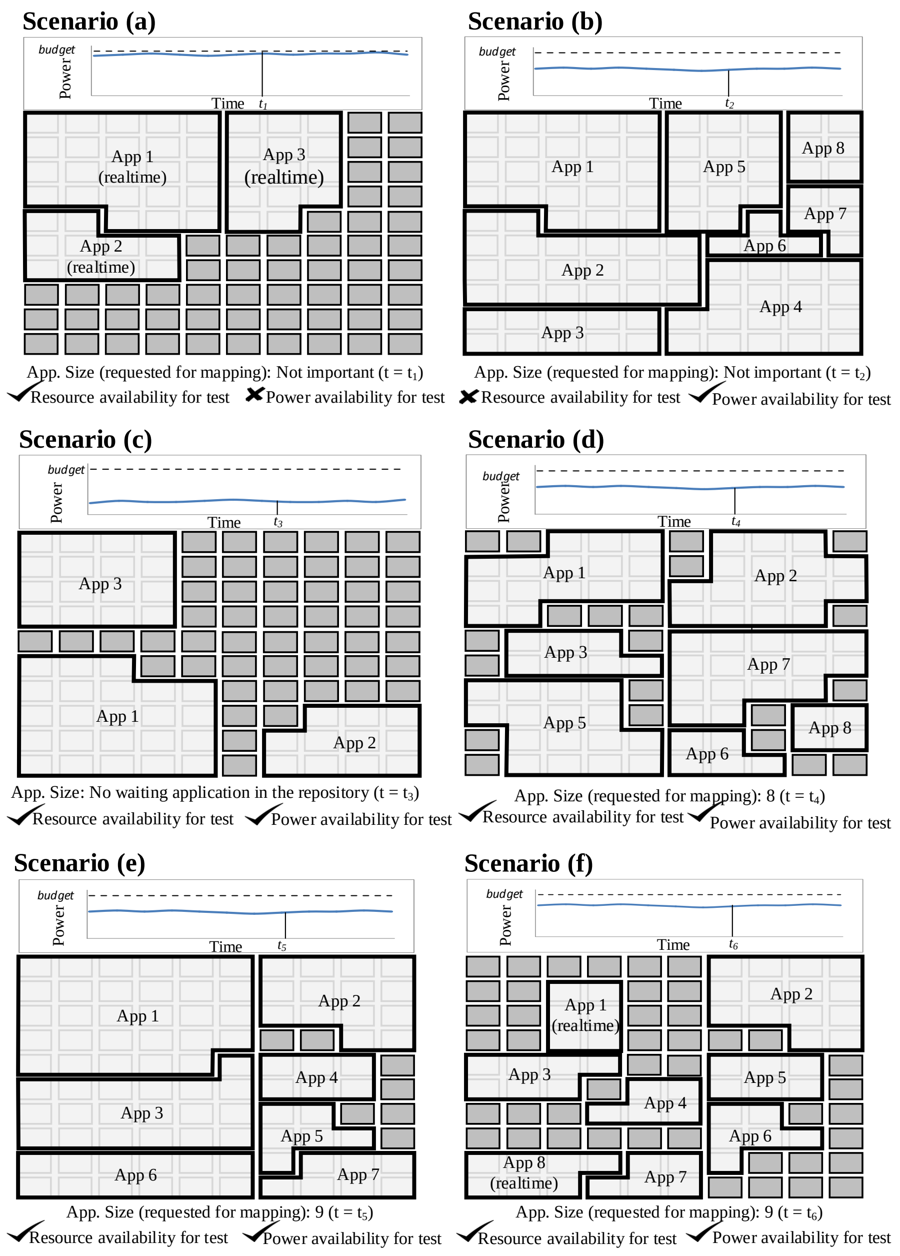

Scenario (a)
budget
Power
Time
t_1
App 1 (realtime)
App 3 (realtime)
App 2 (realtime)
App. Size (requested for mapping): Not important ($t = t_1$)
✓ Resource availability for test
✗ Power availability for test
Scenario (b)
budget
Power
Time
t_2
App 1
App 5
App 8
App 7
App 6
App 2
App 4
App 3
App. Size (requested for mapping): Not important ($t = t_2$)
✗ Resource availability for test
✓ Power availability for test
Scenario (c)
budget
Power
Time
t_3
App 3
App 1
App 2
App. Size: No waiting application in the repository ($t = t_3$)
✓ Resource availability for test
✓ Power availability for test
Scenario (d)
budget
Power
Time
t_4
App 1
App 2
App 3
App 7
App 5
App 8
App 6
App. Size (requested for mapping): 8 ($t = t_4$)
✓ Resource availability for test
✓ Power availability for test
Scenario (e)
budget
Power
Time
t_5
App 1
App 2
App 4
App 3
App 5
App 6
App 7
App. Size (requested for mapping): 9 ($t = t_5$)
✓ Resource availability for test
✓ Power availability for test
Scenario (f)
budget
Power
Time
t_6
App 1 (realtime)
App 2
App 3
App 5
App 4
App 6
App 8 (realtime)
App 7
App. Size (requested for mapping): 9 ($t = t_6$)
✓ Resource availability for test
✓ Power availability for test

Fig. 2: Frequent scenarios regarding resource and power availability for test

SBST routines can be executed by using the remaining power budget and on the idle cores, i.e. in a transparent way since the system performance is not degraded.

Scenario (d). At time t_4, the RTM Unit receives a request to execute a new application having eight tasks. However, the RTM strategy [36] may decide that it is not convenient to immediately execute the applications. The reason is that, even though there is room in the power budget, the current system status is characterized a high dispersion of the idle cores. This would imply an inefficient choice in terms of performance and energy consumption due to the communication costs. Therefore, since the RTM Unit delays the application execution until a contiguous region composed of at least eight cores will be available, the system can employ the available power budget to run the test process on the idle cores. Dispersed cores in such scenarios are the best candidates for being non-intrusively tested without any degradation of the system performance.

Scenario (e). At time t_5, the RTM Unit receives a request to execute an application with nine tasks. In this scenario, even if the available power budget is sufficient for the execution of the application, there are not enough cores available in the system to map the arrived application. Once again, the available power budget can be exploited for non-intrusive online testing of the available cores.

Scenario (f). At time t_6, the RTM Unit receives the request to execute a new application having nine tasks. When considering the current system status, the application can be potentially executed in that moment due to the availability of more than nine idle cores. Unfortunately, according to the pre-mapping power estimation performed by the DPM Unit (with specific techniques, such as [39]), the available power budget is not large enough to support the arrived application. At the same time, the DPM Unit is also not able to reduce the power consumption of the other applications currently running on the system due to their performance requirements. This scenario represents another situation in which the available power budget can be used to test a number of unallocated cores.

There are also other issues regarding online testing in the considered scenario: when conditions on the availability of resources and power are satisfied (as in scenarios from (c) to (f)), it is also necessary to choose the candidate cores to be tested. However, the concurrent test of all the idle cores generally overcomes the available power budget. Moreover, as discussed in Section 1, testing activities, and in particular SBST, have to be executed at several voltage-frequency levels to ensure the correct behavior of the system with the various settings [42, 15]; as a consequence, it is necessary to take into account that each of these configurations will have a different power consumption/execution time trade-off. As a result, in the scenarios (d), (e), (f), it is also necessary to consider such aspect to run the SBST routines with a low voltage-frequency setting on several cores at the same time, or, when it is required, to run a single test with a high voltage-frequency setting on a specific core.

The accurate analysis of the presented scenarios clearly shows the promising opportunity to perform non-intrusive online testing in many-core systems. Actually, the highly variable and evolving status of the many-core system due to the dynamic workload presents periods with a high resource and power utilizations and period with a low utilization. Therefore, an opportunistic online test scheduling method can take advantage of the second kind of situations in order to test the dark cores as long as there is enough room in the remaining power budget. This chapter will present a possible solution to this online test scheduling problem in the era of dark silicon.

5 Dark Silicon Aware Online Testing Framework

The proposed framework for dark silicon aware online testing is presented in Figure 3. It is an extension of the classical runtime management framework discussed in Section 3, with some additional components devoted to the execution of the test-related activities.

The goal of the proposed approach is to transparently run SBST routines during the system activities without affecting the execution of the nominal workload. Thus, the aim is to guarantee that processing cores are not affected by permanent failures and, at the same time, to maintain the required level of performance for the running workload. The basic idea is to test each core with a rate proportional to the stress it has been affected due to its utilization. If a core is frequently used for execution of applications, it is highly stressed and therefore needs frequent tests. On the other side, if the core has been rarely allocated, it does not require urgent testing in the near future. The benefit of this approach is to guarantee the necessary test frequency without performing cores' over-testing that would have a negative effect on the execution of the nominal workload in terms of larger power consumption and unnecessary resources occupation, or cores' under-testing that would reduce the reliability of the system.

The proposed testing approach introduces a new component to the system, Test Scheduling Unit (TSU), that is devoted to select the cores that need to be tested according to the experienced stress and the scheduling of the testing task on those cores. The experienced stress is estimated by means of a criticality metric. It is computed according to a specific hardware component integrated within each core counting the number of executed instructions. The TSU works in a tightly-coupled way with RTM and DPM units to define a proper test scheduling. In particular, the RTM unit has been slightly modified in order to take into account the fact that if a core is candidate for the test procedure, it should not be considered for mapping purposes. In the following subsections, the various activities of TSU are discussed in details together with the internal modifications to RTM Unit necessary to handle the test information received by TSU.

5.1 Monitoring Cores' Stress

The first activity of the Test Scheduling Unit is to select the cores to be tested. Such activity is performed by monitoring the stress experienced by each core.

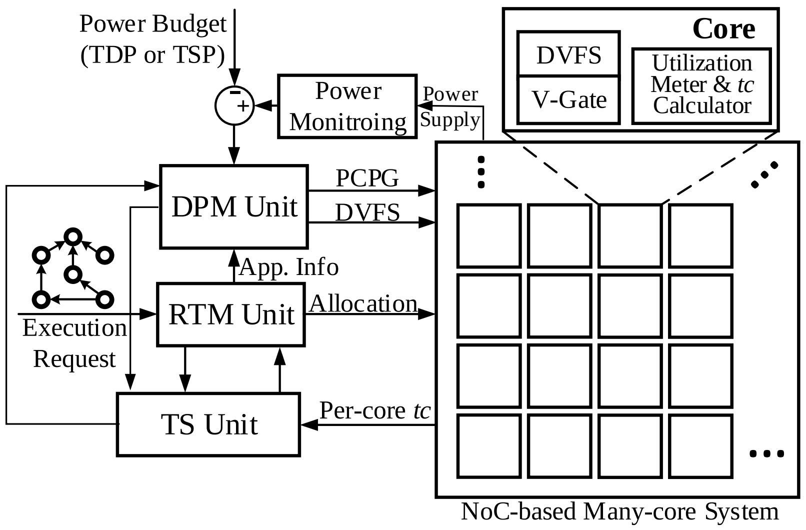


Fig. 3: The system architecture including the online testing framework

As many modern multi-core architectures are not provided with aging sensors, in order to measure the experienced stress, some past testing approaches [8, 17] have exploited the available per-core hardware counters of the executed instructions. An example of architecture provided with such counters is the Intel SCC platform [33]). For instance, the approach proposed in [17] schedules a test on a core every time the instruction count, also dubbed as utilization metric, reaches a specified threshold i.e., 10M, 100M, or 1B instructions. Moreover, in [8], a similar more fine-grained approach envision the availability of counters for each execution unit in order to reduce the execution time.

Therefore, we assume that each core with coordinates $(i, j))$ is equipped with an instruction counter, called Utilization Meter (UM), which value α_{ij} is incremented every time an instruction is executed and is reset when the core is tested. Based on the α_{ij}, the UM computes a *test criticality* parameter tc_{ij} indicating the urgency of a core to be tested due to the experienced stress. More precisely, the parameter is computed according to the following equation:

$$tc_{i,j} = \frac{\alpha_{ij}}{\delta} - 1 \tag{1}$$

where δ is a threshold stating the number of executed instructions that triggers the test procedure. According to this definition, tc_{ij} assumes a value in the range $[-1; +\infty)$. As long as tc_{ij} is lower than 0.0, α_{ij} value is still below the specified threshold δ and therefore the core does not need to be tested. Then, whenever tc_{ij} exceeds 0.0, it means the corresponding core needs to be tested at the earliest convenient moment. The UMs send tc_{ij} matrix to the TSU at fixed time intervals, by using an interrupt-based mechanism to minimize redundant communications. Then, TSU collects all candidate cores requiring to be tested so that they can be analyzed in the subsequent test-aware mapping and test scheduling phases.

Finally, when TSU starts the execution of the SBST routine on a core, it also resets the corresponding α_{ij} counter; consequently, the tc_{ij} value becomes -1.

5.2 Testing-aware Mapping

The mapping of the nominal workload and the testing execution are two conflicting activities since both require processing resources and consume power. A classical approach of interrupting nominal execution to execute test procedures as soon as the triggering condition is violated, as in [8], cause a considerable performance degradation, especially if there is a high requests' rate. In fact, execution of test procedures use a share of the power budget, and, moreover, the mapping-agnostic selection of cores to be tested would increase fragmentation in cores' occupation [43, 44]. On the other hand, prioritizing the mapping of the nominal workload would negatively affect the reliability of the system by delaying test procedures. Indeed, test execution should be dynamically adapted to transparently "intersect" with nominal applications' execution. In this way the goal of the approach is achieved: one should not cause any performance degradation in the workload execution while satisfying reliability issues.

In the considered system, the RTM unit uses a strategy which maps the tasks of the same application on a contiguous set of cores [43, 44]. In this way, power consumed by communication and latencies are considerably reduced. In the RTM unit, such a region of neighboring cores is identified through a metric called Squared Factor SF_{ij}, introduced in [43]. In particular, SF_{ij} metric relates to the number of almost-contiguous available nodes around a given node.

In this scenario, TSU needs to prevent the RTM unit from allocating cores with $tc_{ij} > 0.0$. Unallocated cores can be later scheduled for testing in an appropriate time, when there is enough available room in the power budget. However, if TSU directly disables the cores having $tc_{ij} > 0.0$, it may cause a dispersion of the planned contiguous allocations. For instance, let us assume that an application with 10 tasks has to be executed on the system depicted in Figure 4. As the SF_{ij} of the node (4,5) is 10, it will be selected to map the application onto its surrounding nodes. However, if two cores of this region have $tc_{ij} > 0.0$, the RTM unit has to allocate some available nodes from south-west region of the system which leads to a high dispersion.

To avoid such performance crippling dispersions, the RTM unit has been enhanced by means of a $newSF_{ij}$ value, which is the number of cores with $tc_{ij} > 0.0$ from the original SF_{ij} value. As a result, the $newSF_{ij}$ value shows the number of available cores around a given core that are not candidates for testing. For instance, the new SF value of the core in Figure 4 will be $newSF_{ij} = 10 - 2 = 8$. Thus, the core will not be selected as the first node for mapping an application with 10 tasks, but with 8 tasks instead. It is worth mentioning that apart from the disabling of the cores with $tc_{ij} > 0.0$, this approach for computing the $newSF_{ij}$ will not necessarily prevent such cores from executing a task. Instead, the metric will only discourage using that area, possibly privileging other areas with a lower number of cores to be tested.

5.3 Test Scheduling

Test Scheduling Unit (TSU) implements test scheduling algorithm that determines the cores to be tested among the candidate ones. The core selection strategy in the scheduling algorithm is based on the following considerations. Due to limits on the available power budget, it may not be possible to test all the candidate cores at the same time. Therefore, it is necessary to define a ranking strategy to assign a priority to the testing activities. In an intuitive way, a possible ranking strategy may be: the higher the tc_{ij} is, the more critical it is to schedule a SBST routine on that core. However, by means of a systematical analysis of several possible working scenarios, we noted that if there are several cores with similar tc values, the ones with vacant vicinity should be ranked higher for testing.

The latter consideration is based on the idea that cleaning up regions of idle cores facilitates future application mappings. In fact, isolated cores with or without $tc_{ij} > 0.0$ are, nevertheless, not suitable for being allocated. More concretely, if we test a core with busy neighbors instead of the one with idle cores around, this will lead to a high dispersion of applications and hence degrading the system performance. At the opposite, if we clean up regions with a large amount of contiguous cores, we

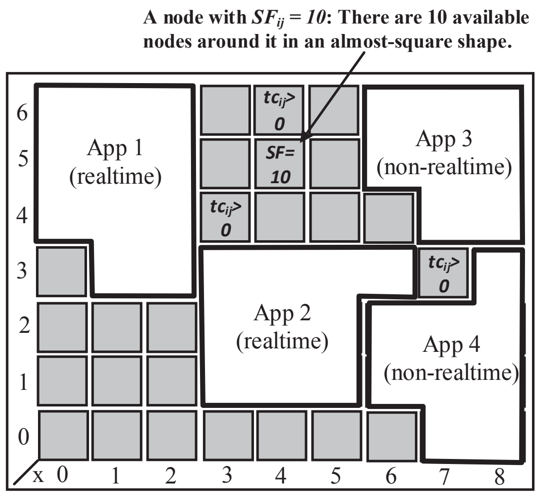


Fig. 4: Example of SF calculation of a node for a given system configuration

Algorithm 1 Selecting Cores for Test Scheduling

```
In predefined intervals:
1: if there is available resource and power for test then {// One of the suitable scenarios shown
   in Figure 2}
2:     Sort available cores based on their $tr_{ij}$ values;
3:     while there is enough power budget for test and # of (cores under test) < $\tau_{\#Test}$ do
4:         Schedule the first core in the ranked list for testing;
5:     end while
6: end if
```

will increase the possibility to map applications and achieving higher performance. Finally, testing applications may be power-hungry. We should avoid placing them in close proximity to each other or to other running applications. Otherwise, testing several adjacent cores together can cause high power densities, and consequently high local temperatures, as shown in the example in Section 7.

Based on the above considerations, a new metric has been defined to rank candidate cores by considering at the same time test criticality and the number of idle cores in the proximity. The metric is defined as:

$$tr_{ij} = tc_{ij} + \frac{\sqrt{SF_{ij}}}{total\ number\ of\ cores} \tag{2}$$

where SF_{ij} value is normalized to the total number of cores in the system. As a metric, SF_{ij} estimates the number of vacant cores around a given core; i.e., the larger the SF_{ij} value of a core is, the more idle cores are around it. Moreover, we use a square root of SF_{ij} value to limit its impact to the cases where tc_{ij} values are too close to each other. For instance, in case of equal tc_{ij} values in Figure 4, the cores $(3,4)$ and $(4,6)$ will be ranked higher than the core $(7,3)$.

Algorithm 1 shows the pseudo-code for the selection of cores to be tested. A peculiarity of the algorithm is an additional control of negative impact of testing on system performance. This is implemented by limiting the maximum number of cores that can be simultaneously tested by means of a given threshold, $\tau_{\#Test}$. The motivation is that we have to cope with a highly evolving scenario; while there might be enough power at the moment to test even more cores, this can change in the near future. Other applications might enter the system, or the power demand and behavior of running applications might change.

In general, execution time of the SBST routine is short compared to applications' execution time. However, regardless of the application types, the overhead of the SBST routine is almost independent of the applications' execution time. The test criticality value of a core (tc_{ij}) depends on the number of instructions executed over time. In case of short applications, tc_{ij} becomes greater than 0 only *after* execution of several applications. While, in case of long applications, the allocated core might need to go under test after the application execution. In this case, the overhead would be again negligible compared to execution time of the application.

Finally, it is worth of noting that the tc_{ij} value increases significantly in case of executing very long applications. Such situation would be managed by means of task migration. However, we leave such an aspect as a future work.

6 Test Scheduling for Different Voltage-Frequency Settings

Based on the recent studies, some specific faults manifest themselves in a particular voltage-frequency (VF) settings [45]. These studies have concluded that multi-/many-core systems equipped with DVFS feature should be tested at multiple voltage levels to ensure that cores can operate reliably at different conditions. Testing at multiple voltage levels is more challenging compared to single voltage level testing as in each voltage level a separate SBST routine execution is needed and the maximum possible operating frequency is limited [46]. Applying the trivial and straightforward test scheduling and repetitively run a test process for every voltage level, drastically increases the overall Test Application Time (TAT) that have a direct impact on the overall system performance. At low voltage levels, test process becomes slower as the frequency is lower that resulting in a longer TAT. In this section, an efficient technique is proposed to test cores at different voltage levels with the aim of providing a uniform testing probability for all the levels while minimizing the performance overhead.

To apply online testing on cores running at different voltage levels, it is essential to use a test scheduling policy with the minimum negative impact on system performance. To this end, allocated cores(s) need to be detected and enough power budget need to be available for the test purpose so that the upper power consumption bound will not be violated. However, as test power consumption at different voltage levels considerably varies, the suitable frequency level in each voltage level should be properly determined at runtime.

In multi-/many-core systems equipped with DVFS feature, usually for each voltage level, an upper bound for the maximum frequency that can operate at that voltage level is defined. For example, in Intel SCC platform, 7 voltage levels for each island are defined where each voltage level has a maximum possible frequency, thus forming more than 15 VF levels per island which can be changed at runtime. In each particular voltage level, different operating frequencies used for testing result in different test power/energy. As the system is tested at runtime with functional methods, and a test at a certain voltage level can be performed at different frequencies (i.e., equal or lower than the maximum frequency at the respective voltage [46]), we define a VF set as the set of different available frequencies (i.e., VF levels) that can be selected for testing at a given voltage level. At low VF levels, power consumption is lower at the cost of longer TAT, compared to high VF levels where higher power consumption is needed to achieve a shorter TAT. This raises a question whether it is more efficient to use a low VF level and save the power to have parallel testing of multiple cores or to use a higher VF level and reduce the TAT for individual cores. Our solution to address this issue is inspired by the traditional 2D rectangular pack-

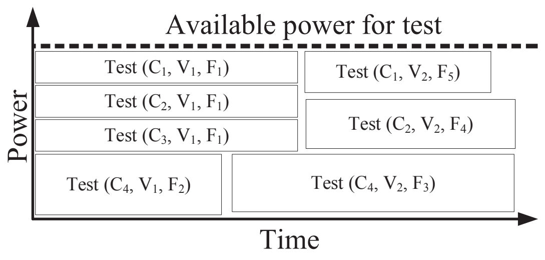

Available power for test
Test (C_1, V_1, F_1)
Test (C_1, V_2, F_5)
Test (C_2, V_1, F_1)
Test (C_2, V_2, F_4)
Test (C_3, V_1, F_1)
Test (C_4, V_1, F_2)
Test (C_4, V_2, F_3)
Power
Time

Fig. 5: An example of rectangular packing model for power aware testing

ing model used in power-constrained testing. Figure 5 shows an example of using 2D rectangular model when three cores are tested over time at different VF levels. Each rectangle depicts a test process as a triplet (C_i, V_j, F_k) where C_i is the core to be tested, V_j is the voltage of test, and F_k is the frequency of test. The width and length of the rectangle correspond to the test power consumption and TAT, respectively, where the total summation of test power at each moment of time should not exceed the maximum available power budget for test. It can be observed that when power budget is limited and an optimal test scheduling algorithm is used, the total areas of the all test rectangles determine the overall test time. This area is the TAT-test power product which can be called as energy consumption for test. We use the energy consumption for test as a metric to choose the proper VF level for test when there is an option to select one VF level among the available VF levels in a particular VF set.

In Figure 6, we have compared the normalized energy with different frequency levels when the voltage is fixed. As can be seen, by increasing the frequency up to the maximum possible frequency, the energy consumption exponentially decreases. That is because of the fact that for a constant voltage, the static power remains constant, and by decreasing the frequency, the penalty of unchanged high static power superimposes the overall power and energy accordingly. From these two observations, we propose a general rule for our test scheduling algorithm that for a given voltage level, the test frequency should be increased as much as possible while monitoring and honoring the total power budget.

Algorithm 2 shows in more details the proposed test scheduling strategy for testing the cores at different VF levels. Algorithm 2 is the extension of Algorithm 1 to consider VF levels in test scheduling, thus offering the system manager the option to choose two different test scheduling policies with contrasting reliability-complexity trade-offs. The input of the test scheduler is the instantaneous power consumption of the system (i.e., P_c) which is provided by the chip power sensor and the output of the test scheduler is the core(s) targeted for being tested at specified VF level(s) (i.e., set of (C_i, V_j, F_k) where V_k and F_k are the voltage and frequency of the core C_i during the test process).

First the amount of available power budget (i.e., *availablePower*) is calculated which is the available portion of power budget that can be used for test purpose (Line 1 in Algorithm 2). If it is less than or equal to zero, it means there is no available

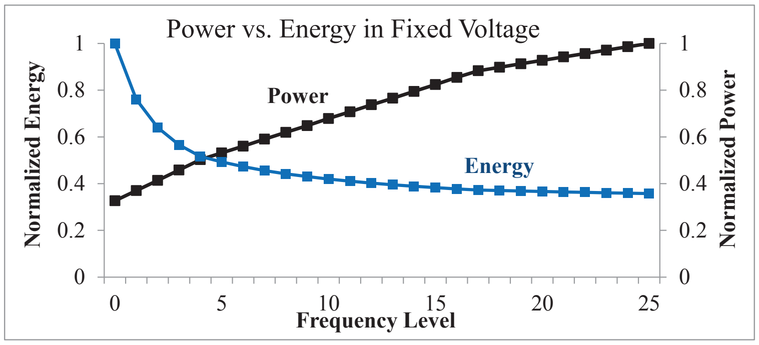


Fig. 6: Power versus energy in a fixed voltage level

power for the test purpose. If *availablePower* is greater than zero (i.e., there exists available power for the test purpose) and the number of cores under test is smaller than the maximum threshold (i.e., $\tau_{\#Test}$), the algorithm will find the first core in the list of cores sorted based on their tr_{ij} values as the target for test (Line 3-4). If such a core exists in the system, for each VF set (i.e., $VFset$) at which the core has not been tested yet, the algorithm will check if the available power can be used to test the core at that VF set or not (Line 5-12). Based on a pessimistically pre-calculated test power for each VF level, the function *minPower* returns the minimum required test power (i.e., CUT_{power}) and the corresponding voltage and frequency (i.e., (V_j, F_k)) to test the core at one of the VF levels at that specific VF set (i.e., $VFset_j$). If the test power is less than the available power, then the core and voltage-frequency for test will be added to the set of target cores for test (i.e.,CUT_{set}) and $availablepower$ will be updated accordingly (Line 7-11). Whenever a core is selected for test, the t_r value for other cores will be updated based on the consideration of the selected core as an occupied node. This causes the next core for test to be selected in other vacant areas.

Searching for faster test process continues as long as $\tau_{\#Test}$ threshold is reached or *availablePower* is less than zero. *availablePower* is the amount of power budget that can be used for test purpose. As the power for testing the cores in different VF levels can be measured in design time and it is determined in runtime, the maximum VF level at which cores can be tested without violating *availablePower* is calculated in the algorithm through a trial-and-error process (Line 14-19). Highest possible VF level is calculated by function $maxVF$. If such a level exists, then the new voltage-frequency for testing will be added to the updated set of target cores for test (i.e., CUT'_{set}) and $availablepower$ will be updated. This process continues until either *availablePower* is larger than zero or all the cores in CUT_{set} are selected. To determine the appropriate VF levels for test purpose we make use of the ideas applied for the traditional power constrained testing in multi-clock domain SoCs [47]. In such works, the problem is to achieve the best TAT while for testing the cores in an SoC, while each core can be run on different VF level. The only difference

Algorithm 2 Selecting Cores for Test Scheduling with VF selection algorithm

Inputs: P_c: Instantaneous power measurement from the sensor;
P_{max}: The maximum power budget (i.e., TDP or TSP);
Output: CUT'_{set}: The target core(s) to be tested at specified VF level(s) (i.e., set of (C_i, V_j, F_k));

Variables: $availablePower$: Available power for test;
CUT_{set}: Temporary variable for the target core(s) and their VF level(s) for test;
CUT_{power}: Core under test power consumption at a given VF level;
Constant: $\tau_{\#Test}$: Maximum number of core(s) under test;

Body:

```
1:  $availablePower \leftarrow P_c$ - $P_{max}$;
2:  while $availablePower > 0$ and # of (cores under test) $< \tau_{\#Test}$ do
3:      Sort available cores based on their $tr_{ij}$ values;
4:      $C_i \leftarrow$ The first core in the ranked list for testing;
5:      if $C_i$ is not tested in $VFset_j$ then
6:          $(CUT_{power}, V_j, F_k) \leftarrow minPower(VFset_j)$;
7:          if $CUT_{power} < availablePower$ then
8:              $CUT_{set} \leftarrow (C_i, V_j, F_k)$;
9:              Update $tr$ for all cores;
10:             $availablePower \leftarrow availablePower$ - $CUT_{power}$;
11:         end if
12:     end if
13: end while
14: while $availablePower > 0$ and there is unselected core(s) in $CUT_{set}$ do
15:     select core $(C_i, V_j, F_k)$ from $CUT_{set}$;
16:     $(CUT_{power}, V_j, F'_k) \leftarrow maxVF((C_i, V_j, F_k), availablePower)$;
17:     $CUT'_{set} \leftarrow (C_i, V_j, F'_k)$;
18:     update ($availablePower$);
19: end while
```

from such works with our problem solving attempt is that the maximum power for test for those power constrained testing is fixed since the test process is done offline, while in our online test scheduling *availablePower* changes during the time. However, if the test time is short enough (which is reasonable assumption as discussed in Section 5.3), we can assume that the power budget for test does not change and two problems are the same. More details regarding the efficiency of this method can be found in [47]. It is worth noting that the proposed algorithm for test scheduling is targeted for platforms featuring per-core DVFS. The extension to also consider per-cluster DVFS is left as a future work.

Table 1: The system settings for different experiment setups

	Technology Node	**System Type**	**Area (mm^2)**	**NoC Size**
First Experimental Setup	16nm	medium	138	12×12
Second Experimental Setup	22nm	large	232	11×11
Third Experimental Setup	32nm	large	254	8×8

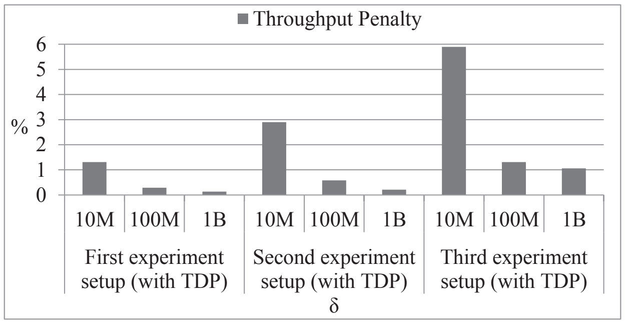


Fig. 7: Throughput penalty for different experiment setups and δ values while using TDP

7 Experimental Evaluation of the Approach

To experimentally evaluate the proposed approach, we implemented a system-level simulation platform for the described many-core architecture together with accompanying runtime management layer and testing procedures in SystemC on the basis of Noxim NoC simulator [48]. The basic core has been characterized by using the Niagara2-like in-order core specifications obtained from McPAT [49]. Physical scaling parameters, power model, voltage-frequency scaling model, and TDP calculation were extracted from the Lumos [11], a framework to analytically quantify power/performance characteristics of devices in near-threshold operation. The physical scaling parameters have been calibrated via circuit simulations with a modified Predictive Technology Model [50]. Then, we integrated HotSpot 5.0 [51] for modeling the thermal behavior of the device. To demonstrate the efficiency of our dark silicon aware online testing approach on many-core systems, we defined three instances of the architecture by considering different technology nodes and different grid sizes as described in Table 1. Finally we defined a variable workload consisting of both synthetic task graphs with 4 to 35 tasks using TGG [35], and real applications, such as MPEG-4, UAV and VOPD, from [52].

The proposed runtime management layer has been defined by using the runtime mapping algorithm presented in [36] and the dark silicon aware power management (DSAPM) technique presented in [39]. In this power management strategy, a PID (Proportional Integral Derivative) controller is used for dynamic power management

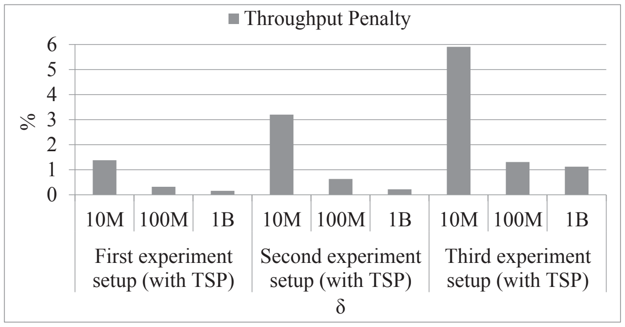
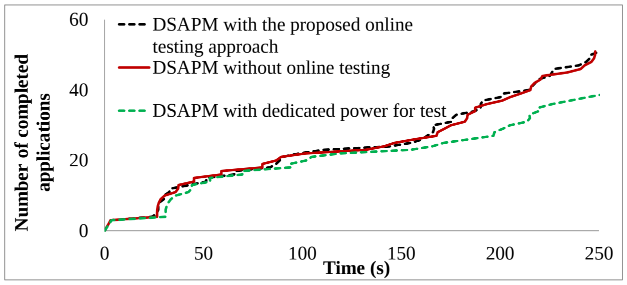


Fig. 8: Throughput penalty for different experiment setups and δ values while using TSP

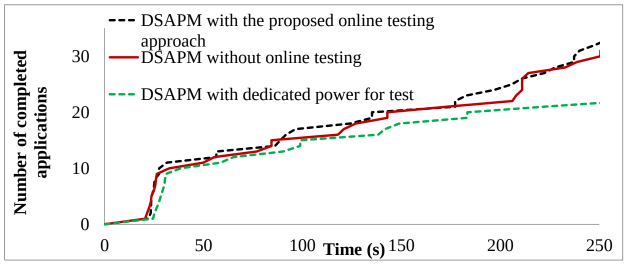
that considers a fixed power budget (i.e., TDP). As an alternative, we have also integrated the TSP calculation tool from [37] to evaluate the dynamic power budget based on the number of active cores at each moment of time.

To prepare the SBST program, we first generate deterministic test patterns from the netlist of HDL implementation of Niagara2- like cores using the technique proposed in [53]. In particular, *NetlistGen.exe* is used for generating netlists of the synthesized cores and fault simulation has been performed with PLI library in HDL environment [54]. Then, we develop test macros based on the generated deterministic test patterns. The overall coverage for the cores' datapath and controller are 79% and 63%, respectively. The duration of the SBST routine is 9000 cycles for each core. Dynamic and static power consumption of the test process has been measured by using the adopted models [11]. Finally, we set $\tau_{\#Test}$ to 4.

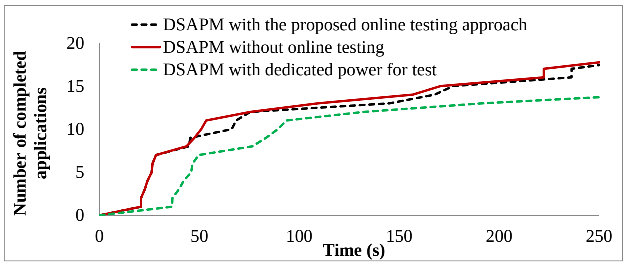
In a first experiment we analyzed the throughput penalty in terms of executed instruction per unit of time of the proposed test scheduling approach when the δ is set to 10M, 100M, and 1B instructions. Moreover, we defined power budgets by using both TDP and TSP methods. The results for the two different power limits are shown in Figures 7 and 8, respectively. As can be seen from the bar charts, the proposed online testing method has a negligible throughput penalty for both TSP and TDP based approaches. In all the cases except for 10M, the overhead is less than 1.5%; when δ is set to 10M, the execution frequency of the test procedure introduces an overhead up to 6% for the architecture designed with the 32nm technology. An interesting aspect is that, for both TDP and TSP experiments, the minimum throughput penalty is observed for the architecture designed with the 16nm technology (first experimental setup), that is the newest node technology, where power limitation is more challenging and the system size, i.e. the total number of cores, is larger than in the other experiments. This shows that the proposed approach will have even more opportunities in the future technologies to find suitable scenarios for online testing. It can be noticed that the penalty while using TSP as the maximum power limit is very similar to the penalty of using TDP. Finally, we can also note that the throughput penalty obtained by the proposed method is considerably lower than in the existing online testing methods reported in [17, 18]. The reason of such improvement is that



(a) For 16nm Technology (first experiment setup)



(b) For 22nm Technology (second experiment setup)

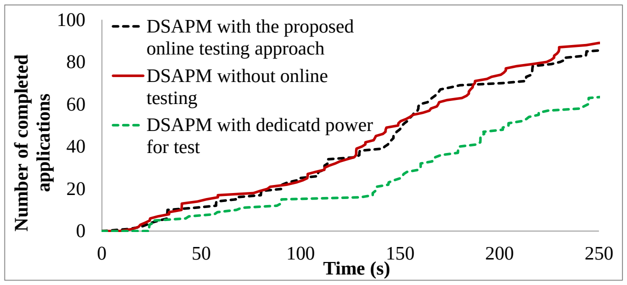


(c) For 32nm Technology (third experiment setup)

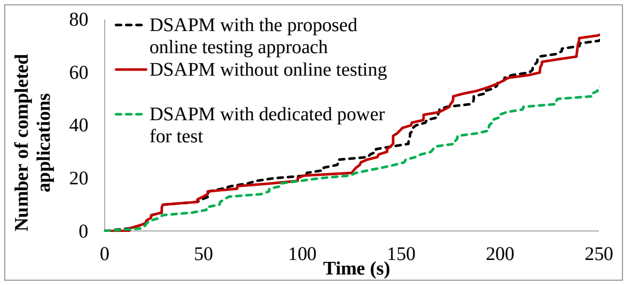
Fig. 9: The number of completed applications vs. time (TDP-based approach)

our method adapts on the working status of the system. In particular, it takes advantage of non-intrusive testing of the cores that are temporarily located in the dark area by exploiting available power budget.

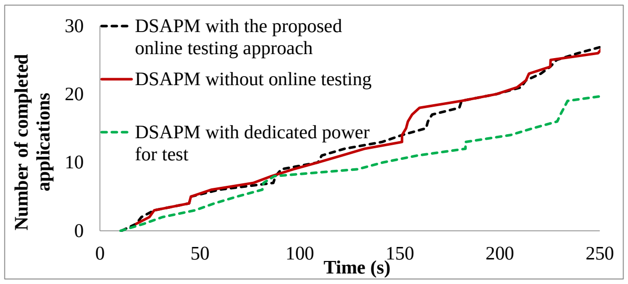
In the subsequent experimental sessions, we delved more into details of the performance overhead analysis of the proposed approach by comparing it against most relevant state-of-the-art methods [55]. Notice that, this earlier method dedicates a fixed amount of power budget to the test process. Thus, we re-run the same experiment with δ set to 10M (that is the worst case scenario) for 250 seconds and we plotted the system throughput over time as shown in Figure 9 and 10 when using TDP and TSP, respectively. Each of these figures compares the throughput of the proposed approach against classical dark silicon aware power management



(a) For 16nm Technology (first experiment setup)



(b) For 22nm Technology (second experiment setup)



(c) For 32nm Technology (third experiment setup)

Fig. 10: The number of completed applications vs. time (TSP-based approach)

(DSAPM) strategy without testing option and the DSAPM strategy coupled with with the technique presented in [55]. It can be seen that the proposed online testing approach achieves a better performance over time compared to the DSAPM approach with dedicated power for test procedures. In particular, the throughput penalties for DSAPM coupled with the technique presented in [55] for 16nm, 22nm, and 32nm technologies are 23%, 20%, and 16%, respectively for the TDP-based approach, and 25%, 23%, and 22% for the TSP-based approach which is larger compared to the corresponding results obtained by the proposed approach, reported in the previously discussed Figures 7 and 8. Furthermore, it can be noted that applications complete and leave the system with almost the same trend for the both

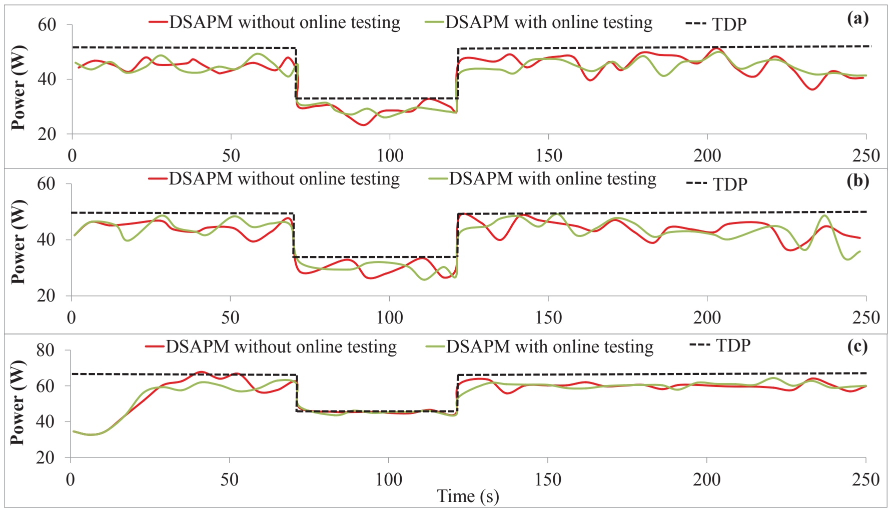
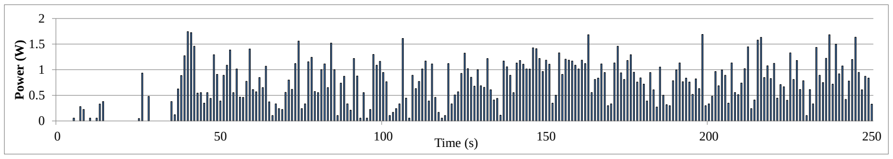


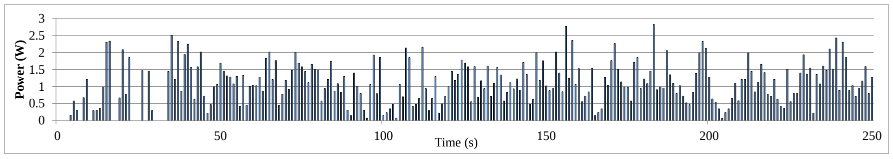
Fig. 11: The power consumption of the system with and without online testing approach for different experiment setups, (a) 16nm, (b) 22nm, and (c) 32nm

DSAPM with online testing and DSAPM without online testing. This confirms capability of the proposed approach to perform transparent scheduling by using the available power budget and resources at runtime for testing the cores with a negligible penalty on the system performance.

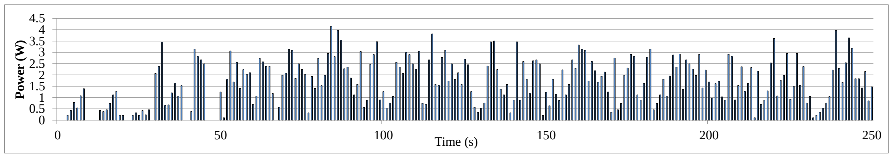
Within the same experimental setup we also evaluated the power consumption of the system over time. Figure 11 shows the power consumption of the system when running a group of random applications while using DSAPM with and without the proposed online testing approach. As it can be observed from the power curves, the total power consumption does not violate the available power (defined with TDP) for both approaches. At the same time, when the power budget is changed, the approach is able to adapt to a new condition. This shows that even though a dedicated power budget is *not* allocated to the test purpose, the DPM unit efficiently honor the TDP bound even when the TDP is changed at runtime. The power curves show that small throughput penalties are experienced in scenarios when the system is frequently busy and the total chip power consumption is most of the time close to the upper bound. In a last series of graphs (Figure 12), it is shown we the actual power consumption dedicated for testing over time. It is worth noting that a bar chart is used since test power is not continuous but it is dedicated in specific periods. The maximum value never exceeds 3W on the available 50W for 16nm and 22nm technologies, and 4.5W on the available 70W for 32nm technology. Moreover, in average such test power is around 2% of the overall power consumption. This demonstrates that the approach is able to limit the instantaneous test power by distributing SBST routine execution over time.



(a) For 16nm Technology (first experiment setup)



(b) For 22nm Technology (second experiment setup)

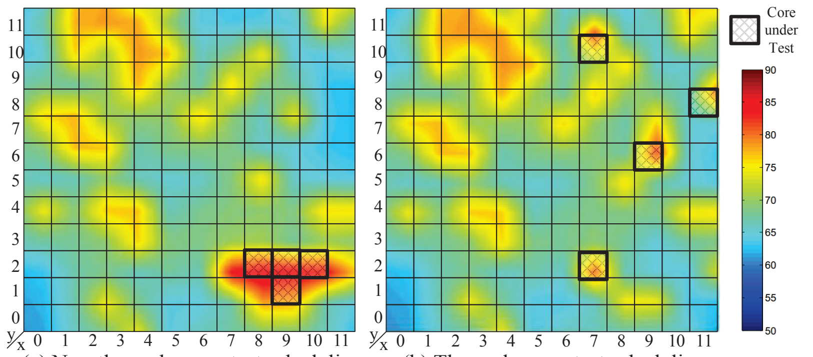


(c) For 32nm Technology (third experiment setup)

Fig. 12: Test power consumption of the system in 16nm, 22nm, and 32nm technology, respectively

We also analyzed efficiency of the test scheduling approach in avoiding temperature hotspots. In particular we analyzed the effect when considering square factor of (SF) in Equation 2. For this, several thermal snapshots are monitored during system runtime and compared against a modified version in which such parameter is not considered in Equation 2, dubbed as non-thermal-aware scheduling. Figure 13 shows the temperature profile of the system while running non-thermal-aware and thermal-aware test scheduling at a given instant of time (with $\tau_{\#Test} = 4$). As can be seen, the non-thermal-aware scheduling selects four neighboring cores which causes high temperatures in a restricted area of the chip. At the opposite the thermal-aware strategy selects cores which are far from each other to avoid thermal hotspots.

Finally, we analyzed the effectiveness of the testing procedure at different voltages/frequency (VF) settings. We characterized the simulation platform with 6 VF sets, i.e. voltage levels, for a total of 29 VF levels. Table 2 reports these different VF sets, by specifying for each of them the related voltage and available frequencies in each set. The target VF level to be assigned to the core under test is chosen among all the options in each VF set. The results of the experiments, performed with the same setup discussed above, are reported in Figure 14, for the three considered technologies, respectively. In particular, each pie chart reports a share of each VF set used for testing activities from the total number of tested cores at the end of the simulations. As can be noticed, VF sets are selected in almost similar way hence demonstrating the fairness of the proposed DVFS-aware test scheduling algorithm. As the sets with higher VF levels consume more power, their shares are a little bit



(a) Non-thermal-aware test scheduling (b) Thermal-aware test scheduling

Fig. 13: Heat maps while running non-thermal-aware and thermal-aware test scheduling

Table 2: Voltage-Frequency sets for test

Technology	**16nm**					
VF set for test	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**
VF Level	1-5	6-10	11-15	16-20	21-25	26-29
Voltage(V)	0.47	0.51	0.56	0.59	0.63	0.68
Frequency(GHz)	0.4-0.64	0.4-1	0.4-1.54	0.4-2	0.4-2.6	0.4-3.1
Technology	**22nm**					
VF set for test	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**
VF Level	1-5	6-10	11-15	16-20	21-25	26-29
Voltage(V)	0.49	0.54	0.6	0.65	0.7	0.74
Frequency(GHz)	0.4-0.67	0.4-1.1	0.4-1.6	0.4-2.1	0.4-2.8	0.4-3.2
Technology	**32nm**					
VF set for test	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**
VF Level	1-5	6-10	11-15	16-20	21-25	26-29
Voltage(V)	0.52	0.58	0.63	0.69	0.75	0.8
Frequency(GHz)	0.4-0.68	0.4-1.13	0.4-1.6	0.4-2.2	0.4-2.8	0.4-3.2

lower than the sets with lower VF levels. As a conclusion the sets with lower VF levels have a better chance to use the available power than the other sets.

8 Conclusions

This chapter presented a power-aware online testing strategy for many-core systems in the dark silicon era. The strategy consists of a non-intrusive online test scheduling algorithm using software-based self test techniques to test idle cores in the system while respecting the system's power budget. Moreover, a criticality metric is used

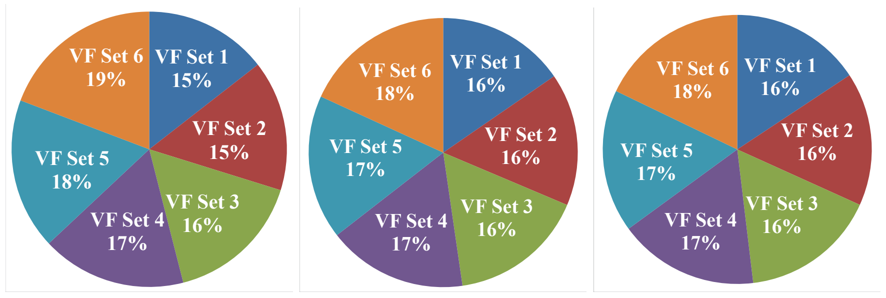


Fig. 14: The share of every particular VF set from the total number of tested cores (%) for 16nm, 22nm, and 32nm technologies

to identify and rank cores that need testing. The goal of the approach is to guarantee prompt detection of the occurred permanent faults, while minimizing the overhead and satisfying the limited available power budget. The presented experimental results show that the proposed power-aware online testing approach can 1) efficiently utilize temporarily unused cores and available power budget for the testing purposes, within less than 1% penalty on system throughput and by dedicating only 2% of the actual consumed power, 2) adapt to the current stress of the cores by using the utilization metric, and 3) cover and balance all voltage-frequency levels during various test procedures.

References

1. JEDEC Solid State Tech. Association, “Failure mechanisms and models for semiconductor devices,” 2010, JEP122G.
2. M. Kaliorakis, M. Psarakis, N. Foutris, and D. Gizopoulos, “Accelerated online error detection in many-core microprocessor architectures,” in *Proc. VLSI Test Symp. (VTS)*, 2014, pp. 1–6.
3. D. P. Sieworek and R. S. Swarz, *The Theory and Practice of Reliable System Design*. Digital Press, 1982.
4. P. Parvathala, K. Maneparambil, and W. Lindsay, “FRITS - a microprocessor functional BIST method,” in *Proc. Int. Test Conf. (ITC)*, 2002, pp. 590–598.
5. M. H. Haghbayan, S. Safari, and Z. Navabi, “Power constraint testing for multi-clock domain SoCs using concurrent hybrid BIST,” in *Proc. Int. Symp. on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2012, pp. 42–45.
6. N. Foutris, M. Psarakis, D. Gizopoulos, A. Apostolakis, X. Vera, and A. Gonzalez, “MT-SBST: Self-test optimization in multithreaded multicore architectures,” in *Proc. Int. Test Conf. (ITC)*, 2010, pp. 1–10.
7. P. Bernardi, M. Grosso, E. Sanchez, and O. Ballan, “Fault grading of software-based self-test procedures for dependable automotive applications,” in *Proc. Design, Automation & Test in Europe (DATE)*, 2011, pp. 1–2.
8. M. Skitsas, C. Nicopoulos, and M. Michael, “DaemonGuard: O/S-assisted selective software-based Self-Testing for multi-core systems,” in *Proc Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2013, pp. 45–51.
9. H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark Silicon and the End of Multicore Scaling,” *IEEE Micro*, vol. 32, no. 3, pp. 122–134, May 2012.
10. M. Taylor, “Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse,” in *Proc. Design Automation Conf. (DAC)*, 2012, pp. 1131–1136.
11. L. Wang *et al.*, “Dark vs. Dim Silicon and Near-Threshold Computing Extended Results,” in *University of Virginia Department of Computer Science Technical Report TR-2013-01*, 2012.
12. A.-M. Rahmani, M.-H. Haghbayan, A. Kanduri, A. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, “Dynamic Power Management for Many-Core Platforms in the Dark Silicon Era: A Multi-Objective Control Approach,” in *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2015, pp. 1–6.
13. M. Shafique, S. Garg, J. Henkel, and D. Marculescu, “The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives,” in *Proc. Design Automation Conf. (DAC)*, 2014, pp. 185:1–185:6.
14. M. Haghbayan, A. Rahmani, P. Liljeberg, J. Plosila, and H. Tenhunen, “Online Testing of Many-Core Systems in the Dark Silicon Era,” in *Proc. Int. Symp. on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2014, pp. 141–146.
15. X. Kavousianos and K. Chakrabarty, “Testing for SoCs with advanced static and dynamic power-management capabilities,” in *Proc. Conf. on Design, Automation & Test in Europe (DATE)*, 2013, pp. 737–742.
16. D. Gizopoulos, A. Paschalis, and Y. Zorian, *Embedded processor-based self-test*. Kluwer Academic Pub, 2004.
17. K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco, “Software-Based Online Detection of Hardware Defects Mechanisms, Architectural Support, and Evaluation,” in *Proc. Int. Symp. on Microarchitecture (MICRO)*, Dec 2007, pp. 97–108.
18. S. Nomura, M. Sinclair, C.-H. Ho, V. Govindaraju, M. de Kruijf, and K. Sankaralingam, “Sampling + DMR: Practical and low-overhead permanent fault detection,” in *Proc. Int. Symp. on Computer Architecture (ISCA)*, 2011, pp. 201–212.
19. A. Apostolakis, D. Gizopoulos, M. Psarakis, and A. Paschalis, “Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors,” *IEEE Trans. on Computers*, vol. 58, no. 12, pp. 1682–1694, Dec 2009.
20. M. Kaliorakis, N. Foutris, D. Gizopoulos, M. Psarakis, and A. Paschalis, “Online error detection in multiprocessor chips: A test scheduling study,” in *Proc. Int. On-Line Testing Symp. (IOLTS)*, 2013, pp. 169–172.

21. B. Khodabandeloo, S. Hoseini, S. Taheri, M. H. Haghbayan, M. R. Babaei, and Z. Navabi, "Online Test Macro Scheduling and Assignment in MPSoC Design," in *Proc. Asian Test Symposium (ATS)*, 2011, pp. 148–153.
22. M. Kakoee, V. Bertacco, and L. Benini, "A distributed and topology-agnostic approach for on-line NoC testing," in *Proc. Int. Symp. on Networks on Chip (NOCS)*, 2011, pp. 113–120.
23. N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-based self-testing of embedded processors," *IEEE Trans. on Computers*, vol. 54, no. 4, pp. 461–475, 2005.
24. C. L. and S. Dey, "Software-based self-testing methodology for processor cores," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, 2001.
25. P. Kabiri and Z. Navabi, "Effective RT-level software-based self-testing of embedded processor cores," in *Proc. Int. Symp. on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2012, pp. 209–212.
26. M. Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Raghunathan, and S. Ravi, "Systematic software-based self-test for pipelined processors," in *Proc. Design Automation Conf. (DAC)*, 2006, pp. 393–398.
27. T. Lu, C. Chen, and K. Lee, "Effective Hybrid Test Program Development for Software-Based Self-Testing of Pipeline Processor Cores," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 516–520, 2011.
28. Y. Xia, M. Chrzanowska-Jeske, B. Wang, and M. Jeske, "Using a distributed rectangle bin-packing approach for core-based soc test scheduling with power constraints," in *Proc. Int. Conf on Computer Aided Design (ICCAD)*, 2003, pp. 100–105.
29. H. Yu, S. Reddy, C. Wu-Tung, P. Reuter, N. Mukherjee, T. Chien-Chung, O. Samman, and Y. Zaidan, "Optimal core wrapper width selection and SOC test scheduling based on 3-D bin packing algorithm," in *Proc. Int. Test Conf. (ITC)*, 2002, pp. 74–82.
30. R. Chou, K. Saluja, and V. Agrawal, "Scheduling tests for VLSI systems under power constraints," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 2, pp. 175–185, June 1997.
31. P. Venkataramani and V. Agrawal, "ATE test time reduction using asynchronous clock period," in *Proc. Int. Test Conf.*, 2013, pp. 1–10.
32. G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Power-aware optimization of software-based self-test for L1 caches in microprocessors," in *Proc. Int. On-Line Testing Symp. (IOLTS)*, 2014, pp. 154–159.
33. J. Howard *et al.*, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Proc. Int. Solid-State Circuits Conference (ISSCC)*, 2010, pp. 108–109.
34. Kalray, "Kalray MPPA Manycore," http://www.kalrayinc.com/.
35. "TGG: Task Graph Generator," http://sourceforge.net/projects/taskgraphgen/, last Update: 2013-04-11.
36. M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen, "Adjustable contiguity of run-time task allocation in networked many-core systems," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 349–354.
37. S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power: Efficient Power Budgeting for Many-core Systems in Dark Silicon," in *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES)*, 2014, pp. 10:1–10:10.
38. M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark Silicon As a Challenge for Hardware/Software Co-design." in *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES)*, 2014, pp. 13:1–13:10.
39. M.-H. Haghbayan, A.-M. Rahmani, A. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dark silicon aware power management for manycore systems under dynamic workloads," in *Proc. Int. Conf. on Computer Design (ICCD)*, 2014, pp. 509–512.
40. K. Ma and X. Wang, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 13–22.

41. Z. Chen and D. Marculescu, “Distributed Reinforcement Learning for Power Limited Many-core System Performance Optimization,” in *Proc. Design, Automation & Test in Europe (DATE)*, 2015, pp. 1521–1526.
42. F. Vartziotis, X. Kavousianos, K. Chakrabarty, R. Parekhji, and A. Jain, “Multi-site test optimization for multi-Vdd SoCs using space- and time- division multiplexing,” in *Proc. Conf. Design, Automation & Test in Europe (DATE)*, 2014, pp. 1–6.
43. M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, “Smart hill climbing for agile dynamic mapping in many-core systems,” in *Proc. Design Automation Conf. (DAC)*, 2013, pp. 1–6.
44. C.-L. Chou, U. Ogras, and R. Marculescu, “Energy- and Performance-Aware Incremental Mapping for Networks on Chip With Multiple Voltage Levels,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1866–1879, Oct 2008.
45. N. Ali, M. Zwolinski, B. Al-Hashimi, and P. Harrod, “Dynamic voltage scaling aware delay fault testing,” in *Proc. European Test Symp. (ETS)*, 2006, pp. 15–20.
46. X. Kavousianos, K. Chakrabarty, A. Jain, and R. Parekhji, “Test Schedule Optimization for Multicore SoCs: Handling Dynamic Voltage Scaling and Multiple Voltage Islands,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 11, pp. 1754–1766, Nov 2012.
47. T. Yoneda, K. Masuda, and H. Fujiwara, “Power-constrained test scheduling for multi-clock domain socs,” in *Proc. Design, Automation & Test in Europe (DATE)*, 2006, pp. 1–6.
48. F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” *URL: http://sourceforge.net/projects/noxim*, 2008.
49. S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Proc. Int. Symp. on Microarchitecture (MICRO)*, 2009, pp. 469–480.
50. B. Calhoun, S. Khanna, R. Mann, and J. Wang, “Sub-threshold circuit design with shrinking CMOS devices,” in *Proc. Int. Symp. on Circuits and Systems (ISCAS)*, 2009, pp. 2541–2544.
51. K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, “Temperature-aware microarchitecture: Modeling and implementation,” *ACM Trans. Archit. Code Optim.*, pp. 94–125, Mar 2004.
52. M. Fattah, A.-M. Rahmani, T. Xu, A. Kanduri, P. Liljeberg, J. Plosila, and H. Tenhunen, “Mixed-Criticality Run-Time Task Mapping for NoC-Based Many-Core Systems,” in *Proc. Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, 2014, pp. 458–465.
53. M. Haghbayan, S. Karamati, F. Javaheri, and Z. Navabi, “Test Pattern Selection and Compaction for Sequential Circuits in an HDL Environment,” in *Asian Test Symp. (ATS)*, 2010, pp. 53–56.
54. Z. Navabi, *Digital System Test and Testable Design: Using HDL Models and Architectures*. Springer, 2010.
55. M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, J. Plosila, and H. Tenhunen, “Energy-efficient concurrent testing approach for many-core systems in the dark silicon age,” in *Proc. Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014, pp. 270–275.