J Electron Test (2013) 29:499–520
DOI 10.1007/s10836-013-5374-z

Process-Variation and Temperature Aware SoC Test Scheduling Technique

Nima Aghaee · Zebo Peng · Petru Eles

Received: 10 July 2012 / Accepted: 12 March 2013 / Published online: 1 May 2013

Abstract High temperature and process variation are undesirable phenomena affecting modern Systems-on-Chip (SoC). High temperature is a well-known issue, in particular during test, and should be taken care of in the test process. Modern SoCs are affected by large process variation and therefore experience large and time-variant temperature deviations. A traditional test schedule which ignores these deviations will be suboptimal in terms of speed or thermal-safety. This paper presents an adaptive test scheduling method which acts in response to the temperature deviations in order to improve the test speed and thermal safety. The method consists of an offline phase and an online phase. In the offline phase a schedule tree is constructed and in the online phase the appropriate path in the schedule tree is traversed based on temperature sensor readings. The proposed technique is designed to keep the online phase very simple by shifting the complexity into the offline phase. In order to efficiently produce high-quality schedules, an optimization heuristic which utilizes a dedicated thermal simulation is developed. Experiments are performed on a number of SoCs including the ITC'02 benchmarks and the experimental results demonstrate that the proposed technique significantly improves the cost of the test in comparison with the best existing test scheduling method.

Responsible Editor: Y. Zorian

N. Aghaee (✉) · Z. Peng · P. Eles
Embedded Systems Laboratory (ESLAB), Department of Computer and Information Science, Linkoping University, 58183 Linkoping, Sweden
e-mail: nima.aghaee@liu.se

Z. Peng
e-mail: zebo.peng@liu.se

P. Eles
e-mail: petru.eles@liu.se

Keywords SoC test scheduling · Adaptive test · Temperature awareness · Process variation · Thermal simulation

1 Introduction

Deep submicron integration causes high power densities and large process variation [7, 18]. The power density for a System-on-Chip (SoC) during test is considerably higher compared to the normal operation [4, 5, 27, 28] and so is the risk of overheating. Therefore, the temperature of the chip under test should be taken into account when designing the test process [5, 27]. Traditionally, thermal-aware test schedules are optimized to achieve minimum test application time while avoiding any overheating. Usually, this is done offline by introducing cooling cycles into the test schedule based on the thermal simulation data [10–12, 26].

A partitioning based approach is introduced in [26]. Selection of the start times for partitions is formulated as an optimization problem and solved with list scheduling [26]. The temperature is initially simulated using HotSpot [14] and is constrained to avoid overheating [26]. During the optimization, the list scheduler utilizes a superposition-based method to assess the temperatures [26].

A partitioning and interleaving approach is introduced in [12]. The suggested method in [12] formulates the number of test partitions and the length of cooling intervals into an optimization problem and then uses constrained logic programming to solve it. The temperature is simulated using HotSpot [14] and is constrained to avoid overheating. Since constrained logic programming is too slow to handle long test sets, a heuristic is proposed for test scheduling [12].

A partitioning and interleaving approach is introduced in [11] which determines the partition lengths, the cooling times, and the schedule, on-the-fly without iterating. The proposed heuristic schedules the partitions based on the

cores' temperatures and remaining test sizes [11]. The partition ends when its temperature approaches a high temperature threshold and then a cooling interval starts. The cooling interval may end when the core's temperature is lower than a low temperature threshold [11]. The optimal value for low temperature threshold is found experimentally so that the test application time is minimized. The temperature is simulated using ISAC [23] and is constrained to avoid overheating [11].

The focus of this paper is on thermal safety issues in the SoC test process, but it is worth mentioning that there are other thermal issues concerning the SoC test. For example, there are defects which appear only in a special temperature range and therefore the test to detect them should be carried out in that special temperature range. The details of this problem and solutions to it could be found in [9, 25]. Beside the thermal safety issues, modern SoCs are affected by large process variation; this issue is discussed next.

The temperature at a certain test cycle is traditionally assumed to be identical for identical chips. However identical chips manufactured with the state of the art technologies are likely to have different temperatures at a certain test cycle because of the process variation. Process variation could be described as geometrical variations in the chips' layouts and variations in materials properties. These phenomena induce variations in the electrical characteristics of the chip (e.g., threshold voltages and leakage currents).

The variations may also affect the thermal characteristics of the chip (e.g., thermal resistances and capacitances). The variation of the electrical characteristics causes power variations and thus temperature variations. The variation of the thermal characteristics causes variation in the device temperatures even if the power is identical for different chips. The temperature variation as an effect of process variation is studied in [8] for finFet devices.

In order to isolate the temperature uncertainty from other thermal phenomena, *temperature error* is defined as the difference between the expected temperature that is estimated by simulation and the actual temperature that is measured by sensors. Temperature error captures uncertainties due to different temperature related deviations, including ambient temperature fluctuations, voltage variations, and in particular process variation.

For traditional technologies, temperature error is small enough to be neglected or to allow worst case designs with negligible overhead [10–12, 26]. But, for new technologies the general trend of increase in power densities and process variation will eventually lead to a situation where the temperature error can no longer be ignored. Therefore, the negative effect of process variation (i.e., large temperature error) should be taken into account in developing efficient test solutions.

In [3], two process variation aware methods are proposed to maximize the test throughput. These methods consider thermal safety as a part of the optimization objective. However, the first method proposed in [3] does not react to temperature deviations, and the other method does not handle intra-chip process variation effects and time-variant temperature deviations. Based on [1, 2], in this paper an adaptive test scheduling method is introduced to address the intra-chip process variation. The proposed method adapts the tests' partitions, cooling intervals, and test schedule to the current temperature situation in order to lower the cost of test. The adaptive test schedule is achieved by selectively reading the on-chip temperature sensors.

Integration of temperature sensors in a chip and their use during both test and normal functionality are already practical. For example, the Power5 processor, manufactured in 2004, is reported to have 24 temperature sensors [6]. A variety of mechanisms to access the sensors during test have also been proposed [15, 24]. A thermal-aware test scheduling technique using on-chip temperature sensors is proposed in [24] which is based on a static schedule. Two heuristics are suggested for generation of the static schedule, one of which takes the temperature constraint into account. It assumes that each individual test runs to completion [24]. This means that a large cooling time is required before start of each test in order to guarantee the thermal safety. This approach leads to very long cooling times and does not handle long and power intensive tests which will lead to overheating if applied to completion [24].

The technique proposed in this paper is based on partitioning and interleaving, as proposed in [11], and efficiently utilizes the cooling times in order to decrease the overall test application time. It also handles the long and power intensive tests which are not thermally safe, per se. The proposed technique generates a near optimal schedule tree in an offline phase. During testing (online phase), a chip traverses the schedule tree, starting from the tree's root and ending at one of the tree's leaves, depending on the actual temperatures.

To our knowledge, we are the first to propose an approach which incorporates the on-chip temperature sensors data during test, in order to adapt to the temperature deviations caused by process variation and to achieve a superior test performance [1, 2]. The methods proposed in [1, 2] are enhanced to improve the test schedule quality, run faster, and handle larger number of cores. In order to achieve these improvements, in addition to the methods we proposed in [1, 2], this paper presents a modified optimization approach and a fast thermal simulation algorithm which is customized for the test scheduling technique.

The rest of the paper is organized as follows. Section 2 presents a motivational example. Section 3 provides the problem formulation and introduces the cost function. Section 4 describes the temperature error model. Section 5 presents the main body of the proposed method. Section 6 explains the thermal simulation approach. Section 7 presents

the experimental results and Section 8 presents the conclusion. A quick reference guide (Section 9) including abbreviations and notations is given at the end of the paper.

2 Motivational Example

Assume that there are two instances, C_w and C_n, from a set of chips manufactured for a given design. When the temperature error is negligible, the temperatures of C_w and C_n during a test process are equal and the same offline test schedule S_1 is used for both of them. As illustrated in Fig. 1a, both C_w and C_n are tested without overheating, since the test schedule includes cooling periods whenever the thermal simulator indicates that the chip temperature will exceed the limit.

Under a certain temperature error condition, due to process variation, the thermal responses of the different chips to the same test sequence will be different. Now, assume that chip C_w is warmer than expected, while chip C_n behaves normally. As illustrated in Fig. 1b, C_w will overheat. To prevent this, a more conservative offline schedule S_2 has to be designed based on the thermal profile of C_w, for both chips, as illustrated in Fig. 1c.

This new schedule will avoid overheating, but will lead to longer test application time TAT_2 compared with TAT_1. For chip C_n, this test application time is unnecessarily long, since the original schedule, S_1, in Fig. 1a is a safe schedule for this particular chip. For a set of manufactured chips with large temperature variations, in order to generate a globally conservative offline schedule, the hottest chip will be used to determine the test schedule. This test schedule will introduce too long cooling periods for most of the chips, leading to an inefficient test process.

In [3], we have proposed a technique to address the above problem with the help of a chip classification scheme. This scheme consists of several test schedules for different temperature error ranges. After applying a short test sequence, the actual temperature of the chip under test is measured using a sensor and depending on its value, the proper test schedule is selected. Therefore, the hotter chips will use a test schedule with more cooling, while the colder chips will have less cooling. The overheating issue is solved and the test application time will not be made unnecessarily long. This approach works fine under the assumption that the thermal behavior of the chips is time invariant.

However, in the case of large process variation, the thermal behavior is time variant and the technique presented in [3] will not be able to achieve high quality schedules. The variation of thermal response with time is illustrated in Fig. 1d. In this case, the temperature of chip C_w gradually lifts up compared to chip C_n, and C_w eventually overheats. A scheduling method capable of capturing temporal deviations is therefore required to deal with this new situation.

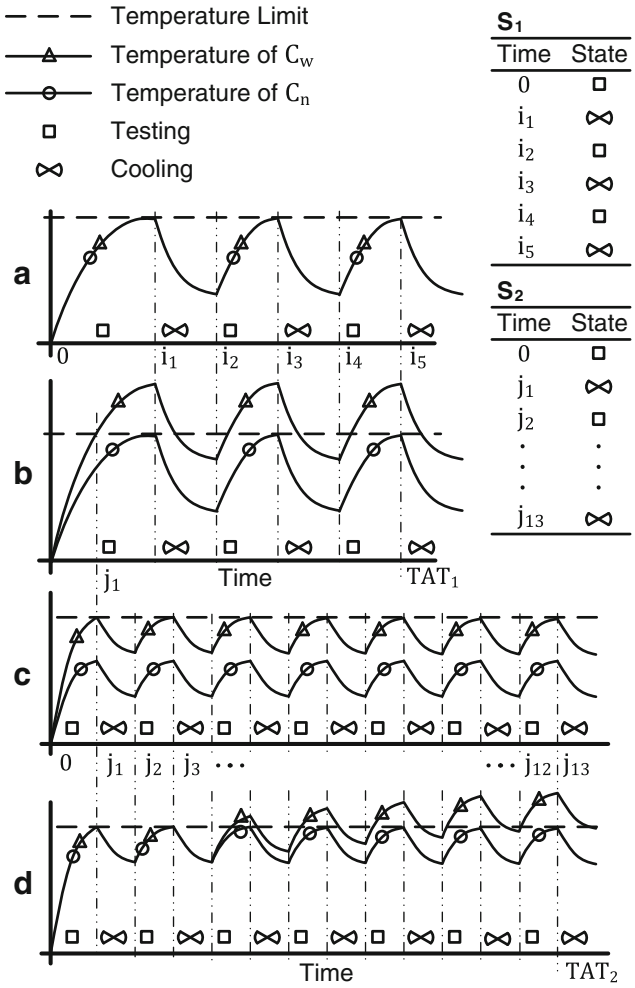
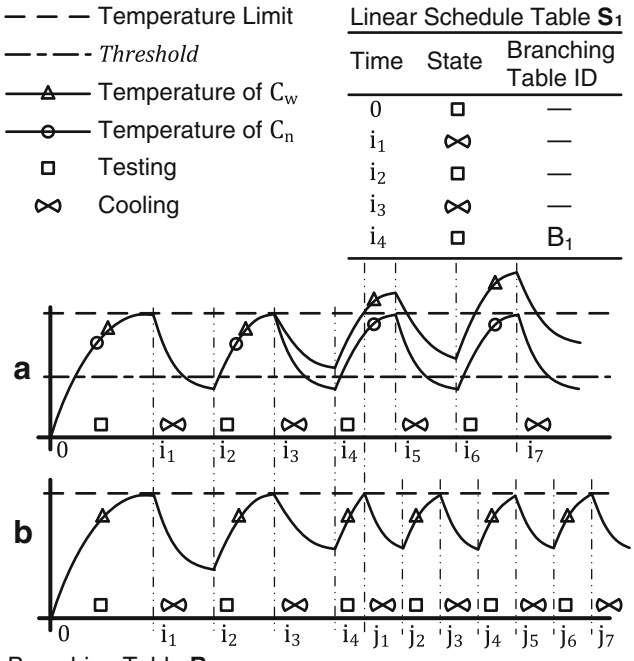


Fig. 1 Test schedule examples (curves are only illustrative). Temperature curves **a** when there is no temperature error; **b** when there is time-invariant temperature error and schedule S_1 is used; **c** when there is time-invariant temperature error and schedule S_2 is used; **d** when there is time-variant temperature error

The temperature behavior given in Fig. 1d is captured in Fig. 2a with more details. The lift up of the temperatures of chip C_w starts at time i_3, as shown in Fig. 2a. Since C_w will only overheat after i_4, both chips can be safely tested with schedule S_1 up to i_4. At i_4, the actual temperature of the chip under test, θ^4, can be obtained via sensors. The actual temperature can then be compared to a *Threshold* and two different situations can be identified:

$$\begin{cases} C_w & if \quad \theta^4 > Threshold \\ C_n & if \quad \theta^4 \leq Threshold \end{cases}$$

For the rest of the test, after i_4, two dedicated schedules, S_2 and S_3, are generated in the offline phase for C_n and C_w, respectively. Therefore, in the online phase the test of C_n continues with schedule S_2, as in Fig. 2a, and the test of C_w continues with schedule S_3, as in Fig. 2b.



Linear Schedule Table S_1

Time	State	Branching Table ID
0	□	—
i_1	⋈	—
i_2	□	—
i_3	⋈	—
i_4	□	B_1

Branching Table B_1

Condition	Linear Schedule Table ID
Temperature ≤ *Threshold*	S_2
Temperature > *Threshold*	S_3

Linear Schedule Table S_2

Time	State	Branching Table ID
i_5	⋈	—
i_6	□	—
i_7	⋈	—

Linear Schedule Table S_3

Time	State	Branching Table ID
j_1	⋈	—
j_2	□	—
j_3	⋈	—
j_4	□	—
j_5	⋈	—
j_6	□	—
j_7	⋈	—

Fig. 2 Schedule and branching tables (curves are only illustrative). Temperature curves when there is time-variant temperature error **a** when both chips are tested with linear schedules S_1 and S_2; **b** when by referring to the branching table, B_1, test of chip C_w continues with linear schedule S_3 after time i_4

In this illustrative example, at the end of S_1, the schedule does a branching to either S_2 or S_3 based on the actual temperature. This information and the branching condition can be captured in a branching table, B_1 in Fig. 2. As shown in Fig. 2a, C_n is tested initially with S_1 and then with S_2, while, as shown in Fig. 2b, C_w is initially tested with S_1 and then with a more conservative schedule, S_3. The segments of the schedule which are executed sequentially without branching are called linear schedules. An adaptive test schedule consists therefore of a number of branching tables in addition to multiple linear schedule tables.

The focus of this paper is multi-core SoC, although the above illustrative example was about a single-core design. It

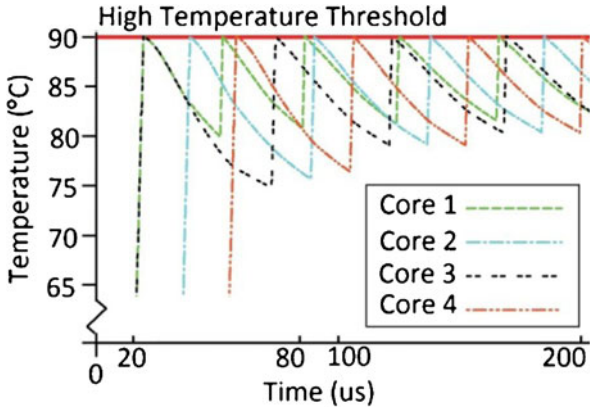


Fig. 3 Temperature curves for a chip under test with four cores [11]

is assumed that, due to the intra-chip process variation, each core has its own thermal behavior similar to what is described above for a chip. Moreover, multi-core designs usually are affected by lateral heat dissipation among the cores and also by the limited test bus width which is shared by different cores.

Temperature curves for a SoC with four cores, as an example, are given in Fig. 3 [11], which shows how the temperatures of the different cores change over time. For a given core, when it is tested, its temperature increases (usually rapidly as shown by the curves). When a core is not tested, there are no switching activities, and it starts to cool down, as shown by the temperature curve going down. To guarantee thermal safety, testing is interrupted when a core reaches the high temperature threshold. As shown in Fig. 3, more than one core may be tested at the same time (e.g., core 1 and core 3 at 20 *us*—the temperature curves for core 1 and 3 are sharply going up). Cores will utilize the available test bus which is freed during the cooling intervals of other cores (e.g., at time 80 *us* core 1 and core 2 are utilizing the cooling time of core 3 and core 4) [11].

3 Problem Formulation

Our goal, as discussed in the previous section, is to generate an efficient adaptive test schedule offline. This is formulated as an optimization problem. The input consists of a SoC design with its set of cores and their corresponding test sequences. The floor plan, the thermal parameters, the static power parameters, and the dynamic power parameters for the chip are given as inputs. The statistical data that models the temperature deviations are also given as input. The adaptive test schedule should be generated to minimize the test application time and the probability of overheating. These objectives are captured by a cost function which expresses the overall efficiency of the generated test schedule, as discussed in the following.

The test schedule should be generated under two constraints. The first constraint is the available test bus width.

The test bus width limits the number of cores that can be tested in parallel. The second constraint is the available Automatic Test Equipment (ATE) memory which limits the size and the number of the linear schedule tables and branching tables. It is assumed that the available memory after loading the test patterns will be utilized for storing the schedule and therefore the amount of memory dedicated to the schedule will not introduce new costs.

In this paper a comprehensive cost function is defined by combining the cost of the overheated chips and the cost of the test facility operation, as follows:

$$cost = \frac{CTF}{TT} + POC \times \frac{TOP}{(1-TOP)} \tag{1}$$

The first term in the cost function is related to the test facility operation cost, which is defined as the operational Cost of the Test Facility per time unit (*CTF*) divided by the Test Throughput (*TT*). The cost of the test facility operation per time unit depends on the cost of the ATE machines, their maintenance costs, and other operation costs. The test throughput is explained later.

The second term of the cost function is related to the cost of the overheated chips, which is the product of the Price of One Chip (*POC*) and the expected number of overheated chips. The expected number of overheated chips is calculated based on the Test Overheating Probability (*TOP*) which represents the number of overheated chips per number of chips entering the test facility.

In (1) the test overheating probability, *TOP*, is divided by $(1-TOP)$ in order to give the expected number of overheated chips per number of non-overheated chips. The cost of the test facility per time unit, *CTF*, and the price of one chip, *POC*, depend on the particular manufacturing and test facility and on the particular SoC. To have a simple model for the test throughput, *TT*, assume that the given test facility is characterized by its overall Effective Test Time per Second (*ETTpS*) and Test Handling Time (*THT*).

The effective test time per second is the total test time that the test facility provides. For example if there are two ATE machines working in parallel, the *ETTpS* could be as high as two. Therefore, the *ETTpS* depends on the number and specification of the ATE machines and possibly other test facility specifications. The test handling time represents the wasted times that chips are not actually under test (e.g., placing, connecting, and detaching the chips) and therefore, it depends on the test facility specifications. The test throughput, *TT*, which depends on the Applied Test Size (*ATS*) and Test Application Time (*TAT*), is calculated as:

$$TT = \frac{ETTpS \times ATS}{TAT + THT} \times (1-TOP) \tag{2}$$

In order to gain a better understanding of the test throughput, the Normalized Test Throughput (*NTT*) is defined by normalizing the test throughput, *TT*, to the effective test time per second, *ETTpS*, and assuming that the test handling time, *THT*, is negligible, as follows:

$$NTT = \frac{ATS}{TAT} \times (1-TOP) \tag{3}$$

The normalized test throughput, *NTT*, is proportional to the applied test size divided by the test application time. It is also proportional to the percentage of the chips that have completed the test without overheating. Therefore large test application time and large test overheating probability will result in small test throughput and consequently the cost component related to the test facility operation will be higher.

In this paper, *CTF*, *POC*, and *ETTpS* do not depend on the test schedule and therefore they are considered to be constants. The cost function is then normalized so that all constants are lumped into one new constant, the Balancing Coefficient (*BC*). The result is the Normalized Cost Function (*NCF*) which is expressed as:

$$\begin{aligned} NCF &= \frac{1}{NTT} + BC \times \frac{TOP}{(1-TOP)} = \\ &= \frac{TAT}{ATS \times (1-TOP)} + BC \times \frac{TOP}{(1-TOP)} \end{aligned} \tag{4}$$

The balancing coefficient, *BC*, is in direct proportion to the price of one chip, *POC*, and in inverse proportion to the cost of the test facility per time unit, *CTF*. The first term in the above equation (normalized cost function) represents the test facility operation cost and shows what volume of test could be applied by the test facility per time unit. The second term represents the balanced cost of the overheated chips and is proportional to the test overheating probability, *TOP*, and the balancing coefficient. The balancing coefficient balances the cost of the overheated chips against the cost of the test facility operation. Expensive chips will results in a larger balancing coefficient and expensive test facility will result in a smaller balancing coefficient.

4 Temperature Error Model

In order to distinguish between the effects of the process variation and other undesirable thermal effects, four different temperatures are defined. The first one is e*xpected temperature*, that is the temperature of a normal chip which is not affected by undesirable thermal effects (including process variation). The expected temperature is an abstract concept and its exact value could not be acquired. The second one is s*imulated temperature*, that is the temperature computed by

simulation. The aim of simulation is to compute the expected temperature and therefore, ideally, the simulated temperature is equal to the expected temperature. The third one is a*ctual temperature*, that is the actual real-world temperature. The actual temperature is physical, but its exact value is usually impossible to acquire due to measurement errors. The forth and last one is *measured temperature*, that is the measured temperature using temperature sensors.

Based on the above definitions, three different temperature errors can be defined. The first one is *simulator error*, that is the difference between the expected temperature and the simulated temperature. The inaccuracies in the simulation model and algorithms contribute to this error. The second one is *measurement error*, that is the difference between the actual temperature and the measured temperature. The inaccuracies in the sensor technologies contribute to it. The third and last one is *variation error*, that is defined as the difference between the actual temperature and the expected temperature. This error has various sources including process variation, ambient temperature fluctuations, and voltage variations.

The focus of this paper is process variation which mainly contributes to the variation error and therefore in this paper we focus on variation error. In the rest of this paper, the temperature error is considered to be the difference between the expected temperature which is estimated by simulation and the actual temperature which is measured by sensors.

Temperature error is discussed above based on its origins but there still is another important point of view, the way it affects the cores. Therefore, temperature error is further categorized into spatial temperature error and temporal temperature error. Spatial temperature error shows that different cores have different temperature errors while the temporal temperature error shows that the same core has different errors at different times. A temperature error model gives the probabilities of the temperature errors for every core in every test cycle. The spatial error model gives the initial error distribution and then the temporal error model is used to recursively estimate the error distribution for the next cycle.

For example, a spatial temperature error model which consists of a discrete distribution shows that at the very beginning of the test the probability of an error equal to −2.3 °C in core 1 is 0.001 while the probability for the same error in core 2 is 0.02. The spatial error model, in this paper, is specified using a look up table which is assumed to be given as one of the inputs. Assuming that the error for a SoC design may range from −20 °C to +20 °C by a resolution of 0.5 °C, the number of the look up table entries (M) would be 80 for a core and $M \times C$ for a SoC with C cores.

Our temporal temperature error model is assumed to be a discrete-time model which means that the temperature error is fixed during a period and then it changes discretely from one period to the next. Therefore, the temporal temperature error model specification has two parts, the period which is called temporal error period and a table of error change probabilities. The temporal temperature error table gives the probability of a particular change in error.

For example, a temporal temperature error model shows that the probability that the error increases by +0.6 °C is 0.015. Assume that the temporal error period is $1\,ms$ and the error is measured to be −5.3 °C at time 0, as shown in Fig. 4. The error will remain −5.3 °C up to $1\,ms$ ($0 + $ *temporal error period*). Then after $1\,ms$ the exact error is not known any more. However the probability of a certain error can be estimated using the temporal temperature error model. In this example, the probability of a temperature error equal to (−5.3 °C + 0.6 °C) = −4.7 °C, between $1\,ms$ and $2\,ms$ is 0.015. Without a measurement at $2\,ms$, the only available information is that the probability of a temperature error equal to (−4.7 °C + 0.6 °C) = −3.1 °C is 0.015×0.015, between $2\,ms$ and $3\,ms$. In Fig. 4, a new measurement is done at time $3\,ms$ and the actual error is −4.7 °C.

The size of the temperature error data set, given as input, might be quite large. In such a case it is necessary to extract a smaller set of data which is representative of the original data in accordance with the accuracy and speed requirements. This is done by clustering the errors into error clusters. The error clusters are characterized by temperature Error-clusters Borders (*EB*). The temperature error range, resolution, and error clusters are assumed to be identical for all cores, in this paper.

The Temperature Error Values ($TEV_m (1 \le m \le M)$) and the Spatial Temperature Error Probabilities ($STEP_{c,m} (1 \le c \le C; 1 \le m \le M)$) are original inputs which are given for a SoC with C cores for M temperature error samples. The Temporal Temperature Error Probability (*TTEP*) is the other input and it gives the probability for a certain change in the error value. The probability that the temperature error value changes from TEV_i to TEV_j is

$$P\left(\text{error value change from } TEV_i \text{ to } TEV_j\right) = TTEP\left(TEV_j - TEV_i\right). \quad (5)$$

The error clustering is assumed to be uniform and the error-clusters borders, $EB_l (0 \le l \le L)$, are identical for all cores. Assuming L error clusters, the size of the original data set

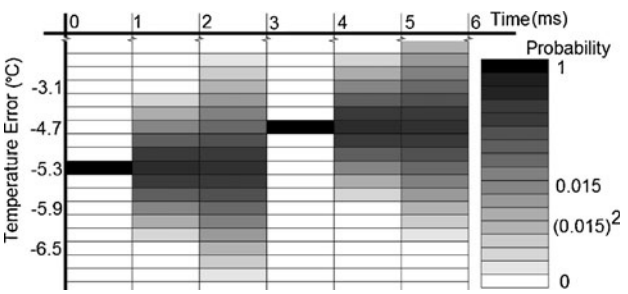


Fig. 4 An example for temporal temperature error probabilities

reduces to $L \times C$. Error clustering will divide the C-dimensional error space into error cells indexed using Cartesian system (i.e., $[l_0, l_1, \ldots, l_{C-1}]$). For example assume that for a SoC with two cores, each core has two error clusters. The 2-dimensional error space is divided into four error cells, indexed with [0,0], [0,1], [1,0], and [1,1]. While the original size of error space is M^C, the number of error cells is L^C. Assuming $M=80$ and $C=2$, the original size is $80^2=6{,}400$ while the size of the clustered error space, with $L=3$, is $3^2=9$.

5 Adaptive Test Scheduling

The proposed adaptive method is based on the on-chip temperature sensors implemented on each core. During test, the actual temperatures of selected cores are read at certain selected moments and the gap between sensor readouts is filled with thermal simulation. A group of chips with similar thermal behavior which are tested with the same schedule is called a chip cluster. During the test, chips are dynamically classified into one of the chip clusters and are tested using the corresponding schedule. The chip clusters vary during the test, and at every adaptation moment (time moment corresponding to a certain branching table) the chip clusters change into a new scheme which is suitable for the new situation.

The parameters that affect the efficiency of the adaptive method are the moments when branching/adaptation happens, the number of edges (i.e., linear schedule tables) and the branching conditions (i.e., chip clustering). For the example in Fig. 2, the adaptation is happening at i_4, the number of edges is two (two linear schedule tables, S_2 and S_3), and the branching condition is a comparison with the *Threshold*.

Since the possible branching moments are multiples of the temporal error period, the first design decision is whether to branch or not at a possible node in a schedule tree. This design decision will be merged with the second design decision which is the number of edges (i.e., the number of chip clusters). The third design decision is the chip clustering for nodes. These problems are summarized into the following two sub-problems.

1. How many chip clusters, at each possible node in the schedule tree, is suitable? The special case of one edge implies no branching, no sensor reading, and no extra effort.
2. What is the proper chip clustering into the given number of chip clusters? The number of chip clusters is known from the answer to the previous question. Depending on the chip clustering some cores may not need sensor readout.

The second question is only relevant when the answer to the first question is larger than one. The above questions are then formulated in two different forms, the first question is described as a tree topology and the second question is the chip clustering for the nodes of that tree topology.

A candidate schedule tree is generated by combining a possible tree topology with a possible chip clustering. The number of candidate tree topologies and the number of alternative chip clusterings grow very fast with parameters like temporal error resolution and the number of cores. Since the number of candidate trees is the product of the tree topology alternatives and the chip clustering alternatives, the search space is so huge that ordinary search approaches would not work fast enough. Therefore a constructive method is suggested to deal with this high complexity.

The schedule tree is constructed by adding small partial trees to its leaves. These small partial trees which are the building blocks of the schedule tree are called sub-trees. A sub-tree consists of a small number of linear schedule and branching tables which makes it possible to be clustered and optimized (scheduled) at once. The tree that is under construction with unfinished tests is called an unfinished tree.

For example, assume that there is an unfinished tree, Tree 1, as shown in Fig. 5a. The linear schedule tables of Fig. 2 correspond to the edges of Tree 1 while the branching table corresponds to node 1, as shown in Fig. 5a. Two sub-trees with one and with two edges are shown in Fig. 5b. Tree 1 has two leaves and combinations of the sub-trees are added to them in order to generate the offspring as shown in Fig. 5c. Offspring 2 is generated by attaching the Sub-tree 1 to node 2 of Tree 1 and attaching the Sub-tree 2 to node 3 of Tree 1.

The proposed constructive algorithm is shown in Fig. 6. The inputs to the algorithm include the switching activities of the tests in order to compute the dynamic power, the thermal error model in order to estimate the temperature errors, and the thermal model of the chip in order to predict the temperatures. Furthermore, the algorithm requires the electrical model of the chip in order to compute the static power and the dynamic power and in order to be informed about the test bus width limit. The test facility specification is also an input to the algorithm which provides the knowledge of the available ATE memory, delay overheads, and the balancing coefficient (i.e., *BC* in Eq. 4).

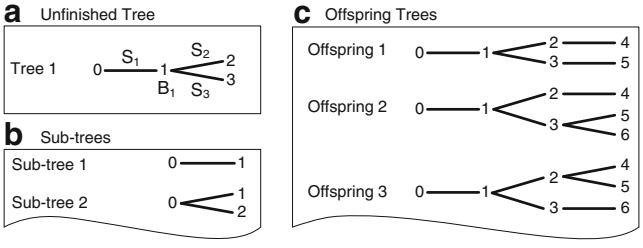


Fig. 5 Constructive method. Main components are **a** unfinished tree, **b** sub-tree topologies, and **c** offspring trees (for S_1, S_2, S_3, and B_1 in (**a**) refer to Fig. 2)

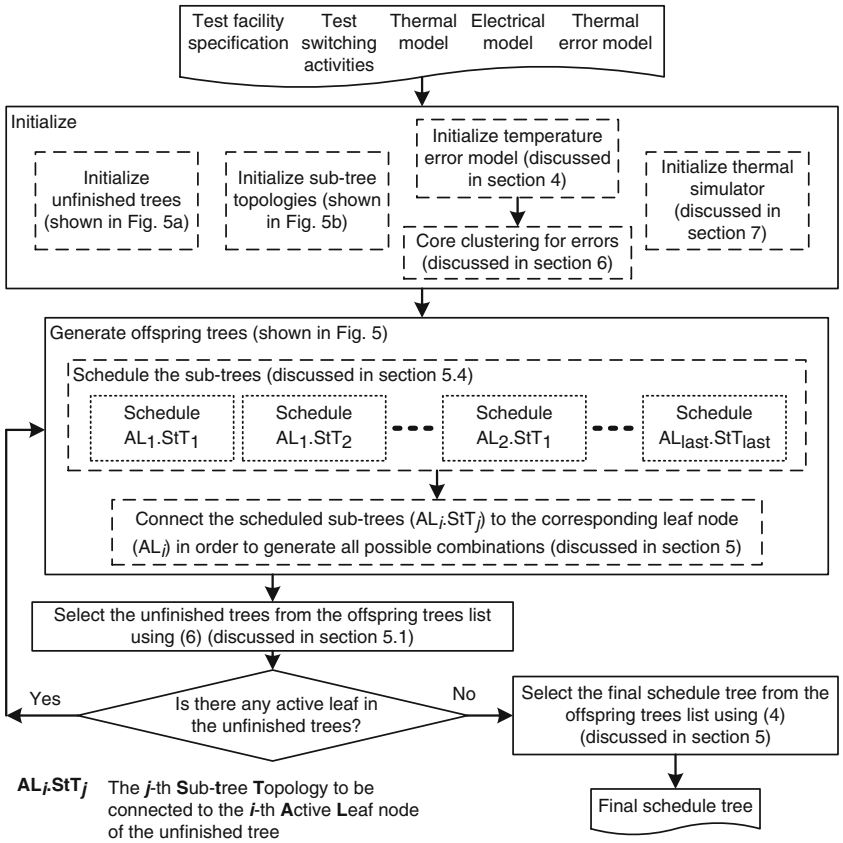


Fig. 6 The proposed constructive method

The algorithm starts with an initialization phase, as shown in Fig. 6. Here, the unfinished tree, sub-tree topologies, temperature error model, and thermal simulator are initialized. Then it proceeds with constructing the schedule tree as will be explained in Section 5.1. The procedure of constructing the schedule tree out of the sub-trees is presented in Section 5.1. The linear schedule tables are discussed in Section 5.2. The sub-tree evaluation is explained in Section 5.3. The sub-tree scheduling which is based on an optimization heuristic is explained in Section 5.4. Some remaining remarks are given in Section 5.5.

5.1 Tree Construction

The schedule tree construction starts with a root node and in each iteration an unfinished tree extends and multiplies by adding possible combinations of sub-trees to its active leaf nodes, as shown in Fig. 5. Then, a small number of promising under-construction trees are selected as unfinished trees from the offspring list for the next iteration. For example, an unfinished tree list (similar to Fig. 5a) will be selected from the offspring list (partially shown in Fig. 5c). The algorithm, as shown in Fig. 6, ends when all the unfinished trees have completed the test.

The selection process keeps the ATE memory constraint satisfied by not selecting the candidates that will exceed the memory limit. A naïve algorithm will have a tendency to create many edges in all iterations at the beginning since it reduces the cost. As a result of this naïve approach, if not taken care of, the algorithm will put many edges near the root of the tree and later on as the memory fills up there will not be any possibility to add a new edge. In order to provide the algorithm with the freedom to put more edges in the more beneficial regions, in our proposed algorithm, the selection is done based on the Scaled Cost Function (*SCF*) as defined in the following.

$$SCF = NCF \times (adjusting\_offset + number\_of\_nodes) \quad (6)$$

The normalized cost function, *NCF* (Eq. 4), is scaled by the tree's number of nodes plus an adjusting offset. Now, adding nodes to the tree is only beneficial if it gives a reasonable cost reduction, otherwise a smaller tree may get a lower scaled cost and manage to survive to the next iteration, while bigger trees are discarded. The effect of the number of nodes is adjusted by the adjusting offset. A small adjusting offset promotes having fewer edges compared to a large adjusting offset which promotes having more edges.

To satisfy the memory constraint, when unfinished tree is selected based on its scaled cost function, it is scheduled for the rest of test by just using the linear schedule tables which mean no further branching. During this scheduling, the

linear scheduling aborts as soon as the memory limit is violated. If the linear scheduling succeeds in respecting the memory limit, the iterations continue. Otherwise, the currently chosen unfinished tree is dropped and the next candidate with larger or equal scaled cost is tested for its compliance with memory limit. The scheduling will fail if no candidate could meet the memory limit, meaning that the limit is too tight even for a linear schedule.

5.2 Linear Schedule Tables

A linear schedule table, as discussed in Section 2, captures a schedule without branching. The linear schedule table entries (start/stop times for each and all cores) are optimized in the offline phase to reduce the probability of overheating. The temperatures are checked frequently in order to keep the overheating probability small.

The start/stop states in the linear schedule tables are generated using the heuristic proposed in [11]. According to this heuristic, the test of the cores with lower temperature and higher remaining test size will be started or resumed earlier. Activating the cores with lower temperature is desirable because it provides longer testing states and therefore reduces the number of test partitions and their corresponding overheads.

By choosing the colder cores while the effect of adjacent cores are taken into account by thermal simulation, in fact, the algorithm activates the cores which are far from the current active cores. This will save the newly activated cores from the accumulated heat in their possible neighbors and furthermore by not activating the adjacent cores, the newly deactivated cores will experience a faster cooling. The heuristic gives also advantage to the cores with longer remaining tests, thus maximizing the interleaving opportunities and also to avoid the situation that a long test sequence leads to a long total test application time.

As mentioned before, each chip cluster is tested with a dedicated linear schedule. Every chip cluster is represented by an error value which will be used to estimate the actual temperature based on the simulated temperature; this error value is called representative temperature error. The estimated temperature is updated periodically by correcting the cores' simulated temperatures with the representative temperature error. The estimated temperature is then used to compute the static power and to determine the 'state' of the cores.

For example, assume that there are two chips $\{D_0, D_1\}$ in a certain chip cluster and chips consist of only one core. Therefore, at a certain moment in time, there are two error values $\{E_0, E_1\}$ corresponding to the two chips. But the linear scheduling heuristic works with one error value for one chip cluster. Therefore, the representative temperature error, r, which is a real number ($r \in \mathbb{R}$) is defined as a value which represents chips error values, $\{E_0, E_1\}$.

The representative temperature error is updated periodically with the temporal error period (see Section 4) while the estimated temperature, static power, and state of the cores are updated more frequently. After updating the state of the cores, the dynamic power sequence is computed. The initial temperatures are available as the results of the previous thermal simulation. Having dynamic and static power sequences in addition to the initial temperatures, the next thermal simulation is performed.

The representative temperature error for a chip cluster is viewed as a safety margin in [3] and its optimal value is experimentally computed for a number of examples. These experiments suggest that the optimal value for a representative temperature error is equal to the border between the chip cluster and the adjacent chip cluster that has larger error (i.e., hottest possible chip in the chip cluster). This is true for all chip clusters except the last one that has the largest error. For example, for a chip cluster stretching from EB_i to EB_j ($EB_i < EB_j$; $j < L$), EB_j would be a good choice to be the representative temperature error for this chip cluster. The representative temperature errors are assigned in a similar way in this paper.

To have an example from a different point of view, assume that in total there are four chips $\{D_0, D_1, D_2, D_3\}$ and chips consist of only one core. Therefore, at a certain moment in time, there are four error values $\{E_0, E_1, E_2, E_3\}$ corresponding to the four chips. Assume that $E_0 < E_1 < E_2 < E_3$. Assume that the chip-clustering algorithm (will be explained in Section 5.4) has generated two chip clusters $\{D_0, D_1\}$ and $\{D_2, D_3\}$. The representative temperature error for the chip cluster that has smaller errors (i.e., $\{D_0, D_1\}$) is $r^0 = E_1$ and the representative temperature error for the last chip cluster, r^1, is formulated as an optimization variable along with the chip-clusters borders in the chip-clustering algorithm.

A new sub-tree optimization method is proposed in this paper that encodes the problem based on chip-clusters borders. The representative temperature errors are defined as chip-clusters borders for all chip clusters but the last one. For the last error cluster (one with the largest errors), the representative temperature errors are encoded along with the chip-clusters borders as the sub-tree optimization variables. This will be explained in Section 5.4.

The optimization problem for a linear schedule table is to minimize the partial normalized cost function by finding the proper start/stop times. This is done based on the heuristic proposed in [11]. The utilized test bus width is the sum of the Test Access Mechanism (TAM) widths of the active cores for tests which utilize the TAM. The schedule size is the product of the number of the linear schedule table entries and the record size. The schedule tree is equivalent to a number of linear schedule tables (edges) in addition to a number of branching tables (nodes), as shown in Fig. 5a. The linear schedule table

is explained above and the rest of the construction process will be explained in the following sections.

5.3 Sub-Tree Evaluation

The schedule tree is constructed by attaching sub-trees to the leaves of the unfinished trees (See Fig. 5). For this purpose, the proper schedule for a sub-tree topology should be found. In a sense, a sub-tree is a tree and the cost function introduced in Section 3 should be usable. However, there is a subtle difference between their objectives. For the schedule tree the objective is its very own cost. For a sub-tree the objective is, on the other hand, the cost of the schedule tree that is to be constructed. Therefore, the cost of the final schedule tree should be estimated assuming that this particular sub-tree is used in its construction. This makes the cost evaluation different for the sub-trees.

To find the near optimal schedule for a sub-tree topology, the partial cost function must be evaluated for different sub-tree clustering alternatives. For the evaluation of the cost function (i.e., *NCF* in (4)), the expected values of the test application time, *TAT*, the applied test size, *ATS*, and the test overheating probability, *TOP*, (denoted by *ETAT*, *EATS*, and *ETOP*, respectively) should be computed by utilizing the temperature error statistics.

The expected values are computed while each edge is being scheduled. In the formulation of the schedule tree, an edge is represented by its destination node. Assuming that the number of nodes is N, the Nodes' Probabilities ($NP_n(1 \le n \le N)$), the Nodes' Applied Test Sizes ($NATS_n(1 \le n \le N)$), and the Nodes' Test Application Times ($NTAT_n(1 \le n \le N)$) are used to compute the expected applied test size and the expected test application time as follows:

$$EATS = \sum_{n=1}^{N} (NATS_n \times NP_n) \tag{7}$$

$$ETAT = \sum_{n=1}^{N} (NTAT_n \times NP_n) \tag{8}$$

In order to explain the expected test overheating probability, *ETOP*, and understand how nodes' probabilities are computed, the notion of nodes' clustering and error cells are introduced here. Temperature errors of cores constitute a C-dimensional errors space (C is the number of cores). For example in Fig. 7, there are two cores and therefore the error space is two dimensional. The horizontal axis represents the error values of the first core and the vertical axis represents the error values of the second core. There are four error clusters for each core and therefore there are 16 error-cells in the Fig. 7.

This is specifically important for the nodes at which branching takes place. Branching at a node is, in fact, a chip clustering to a number of groups, so that each chip cluster corresponds to an exclusive edge branching out of that node. Chips are identified by their cores' errors and therefore a chip clustering is a partitioning of the C-dimensional error space into a number of chip clusters.

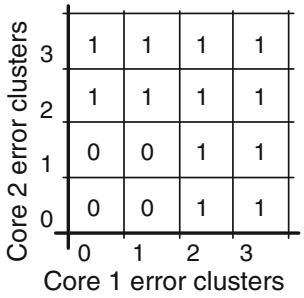


Fig. 7 An example for error-cells labeling. Four error-cells are labeled with *0* that is the ID of the chip cluster number 0 and the remaining 12 error-cells are labeled with *1* that is the ID of the chip cluster number 1

This means that a chip cluster is a combination of specific error intervals of the cores. A candidate 'sub-tree clustering' is a set of chip clustering alternatives for nodes. Furthermore, a candidate 'sub-tree clustering' could be viewed as a set of nodes' clustering alternatives for a sub-tree topology. An error cell is a cell in C-dimensional error space separated by cores' error-clusters borders and therefore its projection on a core error axis is an error cluster for that core. Therefore, a node clustering could be seen as assigning error cells to chip clusters or equivalently labeling error cells with chip clusters. An example for labeling of the error cells is shown in Fig. 7. There are two cores (C=2) in the figure, and the numbers inside the rectangular error cells are labels.

A candidate sub-tree topology will have a number of candidate clustering alternatives which label the nodes' error cells with the relevant chip clusters. Each chip cluster for a node corresponds to an edge branching out of that node and corresponds to a linear schedule table. Each node has its own dedicated Error-Cell Labeling $\left(ECL_{n,l_1,l_2,\ldots,l_C}(1 \le n \le N; 1 \le l_i \le L)\right)$. Looking from a branching node, a succeeding node corresponds to a chip cluster and therefore it receives a Node's Cluster Label ($NCL_n(1 \le n \le N)$) to represent that chip cluster (or equivalently the preceding edge and corresponding linear schedule). This label indicates which of the branching node's chip clusters will lead to a certain succeeding node.

The probabilities of error cells for different nodes and consequently the probabilities of those nodes are computed based on the temperature error model and based on the chip clusterings of the preceding nodes. In order to speed up the computation of the Error-Cells Probabilities (*ECP*) the Error Cell Change Probabilities (*ECCP*) are pre-computed as shown in (10). The error cell change probabilities are, in fact, the concentrated effect of the temporal error model which is repeatedly used to compute the error-cells and nodes probabilities.

It is assumed that the variation in the probabilities inside an error cluster is negligible. Furthermore, it is assumed that the error change probabilities for different cores are independent. The error-cell probabilities change from node to node and therefore most of the time the equations are about two nodes, the *origin* and the *destination*. The error cells for the *origin* node are superscripted with *O* and for the *destination* node with *D*. *ECCP* is computed as follows:

$$ECCP^{\text{before normalization}}_{l_1^O,l_2^O,\ldots,l_C^O,l_1^D,l_2^D,\ldots,l_C^D} = \prod_{c=1}^{C} \sum_{i=EB\left(l_c^O\right)}^{EB\left(l_c^O\right)} \sum_{j=EB\left(l_c^D\right)}^{EB\left(l_c^D\right)} TTEP\left(TEV_j - TEV_i\right) \tag{9}$$

TTEP is the temporal temperature error probability, *TEV* is temperature error value, and *EB* is error cluster border. *ECCP* is computed as follows:

$$ECCP_{l_1^O,l_2^O,\ldots,l_C^O,l_1^D,l_2^D,\ldots,l_C^D} = \frac{ECCP^{\text{before normalization}}_{l_1^O,l_2^O,\ldots,l_C^O,l_1^D,l_2^D,\ldots,l_C^D}}{\sum_{[i_1,i_2,\ldots,i_C]=[0,0,\ldots,0]}^{[L,L,\ldots,L]} ECCP^{\text{before normalization}}_{l_1^O,l_2^O,\ldots,l_C^O,l_1^D,l_2^D,\ldots,l_C^D}} \tag{10}$$

The error-cell probabilities for the root node (i.e., $n=0$) are computed based on the spatial temperature error probabilities (*STEP*) as follows:

$$ECP_{n,l_1,l_2,\ldots,l_C} = \prod_{c=1}^{C} \sum_{i=EB(l_c-1)}^{EB(l_c)} STEP_{c,i}; \text{for } n = 0 \tag{11}$$

The error-cell probabilities for non-root nodes (i.e., $n>0$) are computed based on the predecessor node which is denoted by *pn*. First, error-cell probabilities just after the branching are extracted from the predecessor node as follows:

$$ECP^{\text{after branching}}_{n,l_1,l_2,\ldots,l_C} = \begin{cases} ECP_{pn,l_1,l_2,\ldots,l_C}; \text{if } ECL_{pn,l_1,l_2,\ldots,l_C} = NCL_n \\ 0; \text{otherwise} \end{cases} \tag{12}$$

While scheduling an edge, overheating may occur to some of the cells (ranges of chips) which have larger temperature errors. Consequently, the probability of these cells at the end of the edge (after the corresponding chunk of the test is applied) is considered to be zero. The error-cell probabilities, *ECP*, after overheating are computed based on Representative Temperature Error (*RTE*) (*RTE* is introduced in Section 5.2) as represented in Eq. 13. Overheating of a core occurs when the core's actual temperature which is estimated by adding *RTE* to the Simulated Temperature (*ST*) exceeds the High Temperature Threshold (*HTT*). A chip is considered as being overheated if at least one of its cores overheat.

$$ECP^{\text{after overheating}}_{n,l_1,l_2,\ldots,l_C} = \begin{cases} ECP^{\text{after branching}}_{n,l_1,l_2,\ldots,l_C}; \text{ if } \forall c \left(RTE_n^c + ST_n^c\right) < HTT \\ 0; \text{otherwise} \end{cases} \tag{13}$$

According to the temperature error models (introduced in Section 4) the error-cell probabilities, *ECP*, after temporal changes are computed as:

$$ECP^{\text{after temporal changes}}_{n,l_1^D,l_2^D,\ldots,l_C^D} = \sum_{[l_1,l_2,\ldots,l_C]=[0,0,\ldots,0]}^{[L,L,\ldots,L]} ECP^{\text{after overheating}}_{n,l_1,l_2,\ldots,l_C} \times ECCP_{l_1,l_2,\ldots,l_C,l_1^D,l_2^D,\ldots,l_C^D} \tag{14}$$

The node's probability, *NP*, is computed as follows:

$$NP_n = \sum_{[l_1,l_2,\ldots,l_C]=[0,0,\ldots,0]}^{[L,L,\ldots,L]} ECP^{\text{after branching}}_{n,l_1,l_2,\ldots,l_C} \tag{15}$$

Node's not Overheating Probability (*NnOP*) is the probability that a chip which corresponds to this edge according to the chip clustering scheme, is not overheated after traversing this edge. *NnOP* for a node, *n*, is computed as follows:

$$NnOP_n = \frac{\sum_{[l_1,l_2,\ldots,l_C]=[0,0,\ldots,0]}^{[L,L,\ldots,L]} ECP^{\text{after overheating}}_{n,l_1,l_2,\ldots,l_C}}{NP_n} \tag{16}$$

Finally, error-cell probabilities, *ECP*, are computed as:

$$ECP_{n,l_1,l_2,\ldots,l_C} = \frac{ECP^{\text{after temporal changes}}_{n,l_1,l_2,\ldots,l_C}}{NP_n}; \text{for } n > 0 \tag{17}$$

Edges are scheduled by determining the linear schedule tables as explained in Section 5.2. Then the candidate subtree clustering is evaluated using the partial cost function which is based on the expected applied test size, the expected test application time, and the predicted test overheating probability. The first two are already introduced in (7–8), and the last one is explained below.

Evaluation of a partial tree is in fact an attempt to predict the cost of the completed schedule tree, based on the current situation of that partial tree. For this purpose, it is assumed that the final schedule tree will be composed of a number of similar partial trees (building blocks for the final schedule tree). These partial trees are assumed to have similar

expected applied test size, expected test application time, and expected test overheating probability. These expected values are assumed to be similar to those of the partial tree that we are evaluating.

Therefore, a good prediction for the test application time and the applied test size would be their current expected values multiplied by the predicted total number of partial trees. Since only the ratio of the predicted test application time to the predicted applied test size matters in the cost function (the first term in Eq. 4), a good choice for predicted values of these variables is their expected values. But the situation for Predicted Test Overheating Probability (*PTOP*) is different since its value does not change linearly when a number of similar partial trees (building blocks) are put one after the other (unlike *EATS* and *ETAT*).

Assuming that there are Q leaves in the tree, the Leaf's Overheating Probability ($LOP_q(1 \le q \le Q)$) is the overheating probability for the path from the root node to the specified leaf node. Its computation includes multiplication over nodes that belong to the specified root-to-leaf path. The overheating probability for leaf q is computed as:

$$LOP_q = 1 - \prod_n NnOP_n; \quad \begin{matrix} q \text{ is a leaf node} \\ \text{for all nodes, } n, \text{ belonging to the root–to–}q \text{ path} \end{matrix} \tag{18}$$

The expected test overheating, assuming a total of Q leaf nodes, is computed as:

$$ETOP = \sum_{q=1}^{Q} \left(LOP_q \times NP_q\right) \tag{19}$$

ETOP can be used in Eq. 4 to evaluate a fully constructed schedule tree, but for partial cost function when the tree is not yet fully constructed, the predicted test overheating probability, *PTOP*, is used in Eq. 6 to evaluate the partial cost function. *PTOP* is computed as:

$$PTOP = 1 - (1 - ETOP)^{\lambda} \tag{20}$$

Think of λ as the total number of partial trees (building blocks) that are assumed to be similar to the current partial tree and will construct the final schedule tree. *ETOP* is computed for the partial tree as expressed in Eq. 19 and then the predicted test overheating probability is computed by assuming that these λ partial trees have overheating rates equal to the current partial tree's overheating rate. λ is computed based on the expected Number of Partial Trees (*NPT*) which is defined as the total test size divided by the expected applied test size, *EATS*, for the current partial tree.

A naïve algorithm will use the *NPT* instead of the λ in (20). But, because of the localities in the schedule tree, partial trees (building blocks) with a lot of cooling may exist. For these partial trees, the expected applied test size is small and consequently the expected number of partial trees, *NPT*, will be estimated pessimistically. This unrealistic estimation may result in exceedingly large predicted test overheating probabilities, *PTOP*, and consequently a long schedule tree with too much of cooling may receive a low cost and be selected.

Therefore, limiting the expected number of partial trees, *NPT*, would be helpful for good schedules to receive a more realistic cost. A reciprocal limiter is used here which amplifies small inputs and attenuates large inputs. In the proposed reciprocal limiter, the output is always one when the input is one and the output is equal to the input in a point that is called *Knee*. The output will be always smaller than the limit which is (*Knee* + 1). The relation of the limited output (λ) to the relaxed input (*NPT*) is

$$\lambda = (Knee + 1) - \frac{Knee}{NPT}. \tag{21}$$

A larger *Knee* promotes lower overheating since the maximal value of λ increases and also because of the increased limiter's amplification for the *NPT* values which are less than *Knee*. On the other hand, a smaller *Knee* will result in schedules with shorter test application time. Other types of limiters might be used provided that they do not cause difficulties for the convergence of the Particle Swarm Optimization (PSO) by introducing abrupt changes in the cost and consequently in the velocities when particles travel throughout the search space. PSO is explained in more details in Section 5.4.

At this point, the introduction to computation of the expected test application time, expected applied test size, expected test overheating probability, and predicted test overheating probability is completed and the expected cost for a sub-tree could be computed using them. Therefore, the clustering alternatives for a sub-tree topology could be evaluated using the scaled cost (Eq. 6). The clustering alternatives are explored by Particle Swarm Optimization (PSO) and the best scheduled sub-tree is selected at the end. This optimization is further discussed in Section 5.4.

5.4 Sub-Tree Scheduling

As mentioned before, the schedule tree is constructed by attaching sub-trees to unfinished trees' leaves (See Fig. 5). For this purpose, the proper schedule for a sub-tree topology should be found. In order to schedule a sub-tree topology which is going to be connected to the specified leaf node of the unfinished tree ($AL_i.StT_j$ in Fig. 6) a heuristic, as shown in Fig. 8, iteratively generates alternative chip clustering schemes and evaluates them. The evaluation is explained in Section 5.3 and requires the sub-trees' edges to be scheduled as explained in Section 5.2.

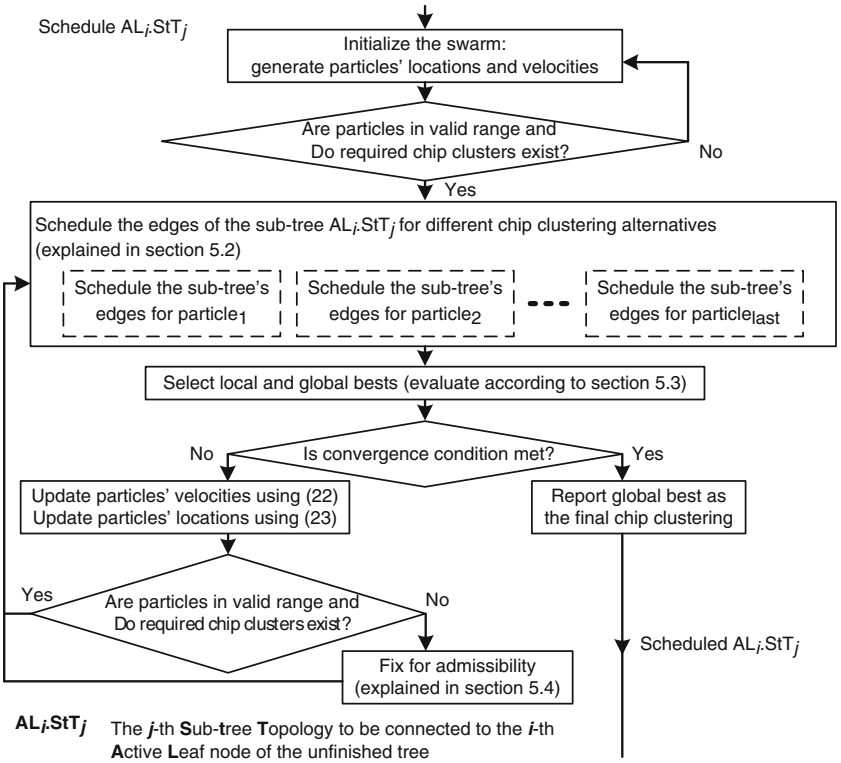


Fig. 8 Sub-tree optimization algorithm

A chip clustering scheme for a sub-tree specifies which chips will take which edges. The chips are specified by their cores' errors and therefore the problem could be seen as assigning chip clusters to the error cells located in the C-dimensional error space. The search space could be seen as the collection of different alternatives for $ECL_{n,l_1,l_2,\ldots,l_C}$. For example for a chip with two cores, the general form is ECL_{n,l_1,l_2} and therefore, for a sub-tree with two nodes, the solutions will be similar to the two alternatives given in Fig. 9.

A solution encoding scheme is suggested in [1, 2] which labels the error cells with chip clusters. The number of the decision variables grows exponentially with the number of cores and therefore the computational complexity is very high. In this paper, we suggest a solution encoding scheme which encodes the chip-cluster borders instead of the error cells. For a node with S succeeding chip clusters the number of decision variables is $S \times C$. For S chip clusters, there are $(S-1)$ chip-clusters borders and the *S-th* value is the representative temperature error for the last cluster. Here, the number of the decision variables grows in proportion to the number of cores and therefore the computational complexity is much smaller compared with the scheme suggested in [1, 2].

Two examples for the suggested solution encoding for a sub-tree with only one node, similar to sub-tree 2 in Fig. 5b, are illustrated in Fig. 10. The solutions correspond to a SoC which has two cores. There are three temperature error clusters per core and the number of edges (i.e., number of chip clusters) in the corresponding sub-tree is two. r_1 and r_2 are representative temperature errors for the last chip cluster. The 0-th chip cluster in Fig. 10a is larger than the 0-th chip cluster in Fig. 10b; one could realize by counting the number of error cells

Alternative 1			
$\mathrm{ECL}_{0,0,0}=0$	$\mathrm{ECL}_{0,0,1}=0$	$\mathrm{ECL}_{0,1,0}=1$	$\mathrm{ECL}_{0,1,1}=1$
$\mathrm{ECL}_{1,0,0}=1$	$\mathrm{ECL}_{1,0,1}=1$	$\mathrm{ECL}_{1,1,0}=1$	$\mathrm{ECL}_{1,1,1}=0$
Alternative 2			
$\mathrm{ECL}_{0,0,0}=0$	$\mathrm{ECL}_{0,0,1}=0$	$\mathrm{ECL}_{0,1,0}=0$	$\mathrm{ECL}_{0,1,1}=1$
$\mathrm{ECL}_{1,0,0}=0$	$\mathrm{ECL}_{1,0,1}=1$	$\mathrm{ECL}_{1,1,0}=1$	$\mathrm{ECL}_{1,1,1}=1$

Fig. 9 Two error-cell labeling alternatives for a chip with two cores and a sub-tree with two nodes. The general form is ECL_{n,l_1,l_2}, n being the node index. Cells are indexed by l_1 and l_2

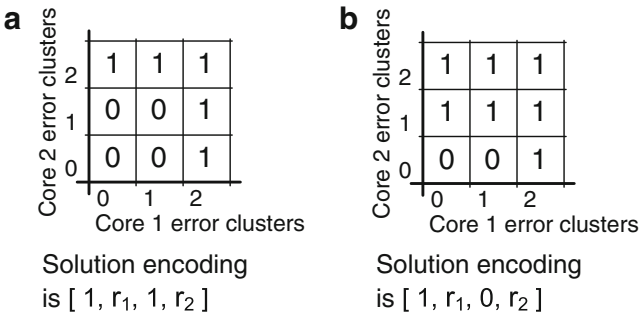


Fig. 10 Two examples for error-cells labeling. Error cells are labeled with chip clusters' IDs (*numbers inside the small rectangles*). The solution encodings are given below the error spaces

which are indexed by 0 or equivalently by comparing the chip-clusters borders for the vertical axes (third element in the solutions encodings).

The possible solutions are then explored using particle swarm optimization that is a meta-heuristic. A candidate solution is called a particle and is represented by its location and its velocity. The locations are the encoded solutions and the velocities are used to determine the next candidate solutions. Each particle remembers its previous best location, and the swarm remembers the global best solution that is the best location any of its particles have visited ever. The previous bests and the global best are then used to give a hint to the random velocities.

A canonical form of PSO uses Eq. (22) to update the velocities. The coefficients in (22) are given as a part of the chosen canonical form [20]. $random_0$ and $random_1$ are two distinct randomly generated numbers between 0 and 1. The location and the velocity on the right hand side of (22) are the current values, and the left hand side velocity is the next value. Since the location, in this paper, is a natural number, the next location is the rounded sum of the current location and the next velocity, as expressed in Eq. (23).

$$velocity = 0.7298 \times (velocity + 2.05 \times random_0 \times (previousBest{-}location) + 2.05 \times random_1 \times (globalBest{-}location)) \quad (22)$$

$$location = \mathrm{Round}(location + velocity) \quad (23)$$

There are two admissibility conditions to ensure that the particles are valid solutions. The first condition is the valid range and the other is the presence of required chip clusters. For example assume that errors range from −10 to +10 and therefore smaller or larger errors will never happen in practice. If it happens that one element in the next particle's location is +11, then this particle is out of range. An example for a required chip cluster not being present is as follows. Assume that there are three edges in a certain node and therefore three chip clusters are necessary for that node. It may happen that in the next particle's location, the first and the second chip-cluster borders are assigned with identical values and therefore the second chip cluster is missing.

The proposed solution encoding which is based on chip-clusters borders works well with particle swarm optimization, since the location and velocity in particle swarm optimization's terms correspond to the location and velocity for chip-clusters borders. A typical particle in the beginning is far from being good and experiences a high velocity towards the better location since typically the difference between the best location and the current location is large at the beginning. Therefore a rapid convergence towards the preferred value for the chip cluster border will take place.

Later on, a typical particle will be close to the optimal location and according to (22) it will move slower, thus pin pointing the preferred value for the chip cluster border. Some experiments, for chip clustering optimization for sub-trees using particle swarm optimization, are reported in [2]. The experiments showed that the particle swarm optimization performs well for this purpose; therefore, it is used in this paper as a part of the proposed SoC test scheduling technique.

5.5 Remarks

Even though the thermal simulator errors and sensor measurement errors are not addressed explicitly in this paper, in practice when the temperature error model is being tuned empirically, a great amount of these errors will also be covered. There still might be small residual errors which are not captured by the temperature error model. These small residual errors are addressed by introducing a small safety margin (a slightly lower temperature limit is used in practice). The effect of this small safety margin on cost is negligible as shown in [3].

In this paper it is assumed that different cores may have different errors and every core has a temperature sensor. The sensor readings may take place only on multiples of the temporal error period. The proposed approach takes the overhead of sensor access and the overhead of reacting to the readout value (i.e., jumping to a new schedule table) into account and, therefore, the number of sensor accesses is automatically kept within a reasonable size.

The proposed optimization technique is structured so that it enables parallel implementations with different granularities. The alternative sub-tree topologies ($AL_i.StT_j$ in Fig. 6) as described in Section 5.1, could be optimized in parallel. For example, assuming one unfinished tree with two leaf nodes and three sub-tree topologies, there will be $3^2=9$ combinations to optimize in parallel.

Furthermore, at the lower level of sub-tree scheduling, each alternative chip clustering in particle swarm optimization ($particle_i$ in Fig. 8), as described in Section 5.4, could be generated (corresponding edges being scheduled) in parallel with other alternative schedules. The scheduling of the edges (i.e., optimizing the linear schedule tables) is the part that requires thermal simulation (dashed-line blocks in Fig. 8). Therefore, these most computationally expensive parts could be implemented in parallel in two different nested levels. The main structure of this work is presented

by this point. There remains the proposed thermal simulation approach which helps to efficiently simulate the temperatures.

The proposed adaptive approach in this paper combines the benefits of an online scheduling technique with the benefits of an offline scheduling approach and avoids their shortcomings. An online schedule will introduce very large overheads that are associated with sensor readouts, decision making process, and pausing/resuming the tests. An offline schedule, on the other hand, is not capable of reacting to variations but has no run-time overheads. In a fully online approach, reading the temperature sensors for all cores as often as it is necessary and making the corresponding decisions for the acquired data will cause a very large load on the test access mechanism and will introduce large delays to the schedule. Our proposed approach uses temperature simulations as much as possible offline and accesses carefully-selected cores' sensors at carefully-selected times during the test.

There is one schedule tree for a chip that addresses all cores individually. For example, in a linear schedule table that corresponds to an edge, it is stated that at time t_1 cores C_0 and C_2 are being tested, while cores C_1 and C_3 are cooling. It might be that in another time, t_5, cores C_1 and C_2 are being tested, while cores C_0 and C_3 are cooling. The linear schedule table is similar to S_1 in Fig. 2, but instead of the second column that shows only one column for *state* (in Fig. 2), there are as many *state* columns as the number of cores. There is only one branching table for one node in the schedule tree (similar to B_1 in Fig. 2) but it contains, in every row, conditions that include at least one core and at most as many cores as the total number of cores.

6 Thermal Simulation Approach

In order to evaluate the candidates, the test application time and the test overheating probability should be computed as explained in Section 5. In order to calculate the test application time and the test overheating probability, the temperatures of the cores are required. Therefore, for every candidate schedule which is examined by the meta-heuristic ($particle_i$ in Fig. 8), thermal simulation should be performed. Thermal simulation is in the main loop in Fig. 8 which itself is in the main loop in Fig. 6. This means that the thermal simulation which is performed inside the optimization loop is repeated numerously.

On the other hand, the thermal simulation is the slowest step in the iterative part of the algorithm. Therefore, the thermal simulation is the bottleneck for the number of the SoC cores which can be handled by the proposed method. In addition to being the limiting factor for the number of cores, the slow thermal simulation is also a bottleneck for the quality of the schedules. Since the optimization meta-heuristic will have a time consuming process inside its main loop, the time required to achieve a quality schedule will be excessively large and impractical, thus the quality should be sacrificed by ending the optimization process prematurely. It is, therefore, important to use a fast thermal simulation approach. In the following, the generic thermal simulation scheme is reviewed and then a fast solution is proposed.

The thermal simulation involves two parts, the thermal model and a way to solve the model response to the given input power. A widely used thermal model for chips is a lumped element model. It means that the chip is modeled as a combination of thermal resistances and thermal capacitances. As an example, HotSpot which is a thermal simulation tool uses such a model [14]. An example for such a thermal model is given in Fig. 11. A typical thermal model consists of a number of thermal elements connected to each other. A connection point of thermal elements is called a node. In Fig. 11, two cores are modeled as two nodes which are connected to two exclusive power sources. Power sources represent the power consumed by the cores.

The input power consists of the static power and the dynamic power. The static power depends on the chip and on the temperature, while the dynamic power depends on the chip and on the input test sequence. Both of the static power and the dynamic power are time-variant, but for practicality reasons, it is assumed that the power is constant during a simulation cycle (a discrete-time model is assumed). Therefore, in the following we focus on a single simulation cycle in which the input power is constant. The input power is updated with new static and dynamic power values, based on the results of the previous simulation cycle and then the simulation for the next cycle is performed.

Assume that the thermal model consists of W nodes and C is the number of cores. In a good thermal model, usually the number of nodes is much larger than the number of cores, $C \leq W$, (e.g., six thermal nodes for two cores) as shown in Fig. 11. Assume that $\boldsymbol{P}$ is the power vector and $\boldsymbol{\Theta}$ is the temperature vector. The mathematical representation of the thermal model is a

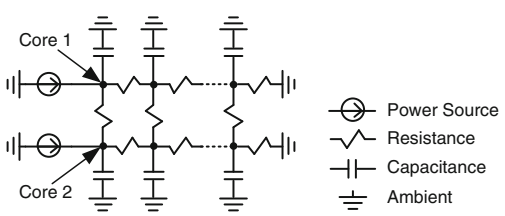


Fig. 11 An example of a lumped element thermal model

system of ordinary differential equations as shown in the following.

$$\boldsymbol{A} \times \frac{d}{dt}\boldsymbol{\Theta} + \boldsymbol{B} \times \boldsymbol{\Theta} = \boldsymbol{P} \tag{24}$$

The properties of the thermal model are encapsulated into two $W \times W$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. $\boldsymbol{\Theta}$ and $\boldsymbol{P}$ are $W \times 1$ temperature and power vectors. The model which is used in this paper is similar to what described here and its corresponding characteristic Eq. (24) is identical to HotSpot's [14]. The mathematical representation of this model, Eq. (24), is a system of linear constant-coefficient differential equations and therefore it is a Linear Time-Invariant (LTI) system [19]. In fact the thermal model is a linear time-invariant lumped element model and the thermal elements are linear and time invariant.

As an example, assume that a SoC has two cores (C=2) and assume that the model has four nodes (W=4). The expanded characteristic equation of the model is

$$\begin{bmatrix} a_0 & 0 & 0 & 0 \\ 0 & a_1 & 0 & 0 \\ 0 & 0 & a_2 & 0 \\ 0 & 0 & 0 & a_3 \end{bmatrix} \times \frac{d}{dt} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} + \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{0,1} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{0,2} & b_{1,2} & b_{2,2} & b_{2,3} \\ b_{0,3} & b_{1,3} & b_{2,3} & b_{3,3} \end{bmatrix} \times \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} P_0 \\ P_1 \\ 0 \\ 0 \end{bmatrix}.$$

θ_0 and θ_1 are core temperatures which should be taken care of. P_0 and P_1 are the powers applied to the cores.

The second part of the thermal simulation is to solve the model in order to find its response to the input power. Usually, the simulation time is divided into smaller intervals in which the power could be assumed to be fixed. Then Eq. (24) is solved iteratively for each interval.

In order to solve (24) there are two distinct approaches, the numerical approximation and the closed form solution. The numerical approximations are usually done with very small intermediate steps, and as a result, the complete temperature curve for the interval is constructed. HotSpot uses the Runge–Kutta method which is for numerical approximation [14]. Though only the temperature at the end of the interval is registered, many points of the temperature curve are calculated. Since we do not need such a detailed temperature curve and we only need the temperature at the end of the intervals, the equation is solved analytically in order to give the temperature at the end of the intervals in a closed form.

In addition to the granularity of the temperature curve, another important factor, which affects the simulation speed, is how frequently Eq. (24) is required to be solved. The scheduling technique presented in this paper, requires large number of simulations. When the system is LTI and the only variation in the inputs (within the simulation intervals) is them being scaled, the differential equation needs to be solved only once at the very beginning. The responses to the scaled versions of the previous inputs are obtained by scaling the previous outputs by the same factor. Since the computational cost of the scaling is less than the computational cost of solving the equation from scratch, a method which utilizes the LTI properties (i.e., scaling and superposition [19]) is faster than the Runge–Kutta method when numerous simulations are required.

In situations that the thermal simulator is invoked quite frequently, the input power is just being scaled from cycle to cycle, and the thermal model is kept unchanged, the closed form solution is faster. Therefore, we continue with the simulation approach which is based on the closed form solution. By using Laplace transform [19] and assuming that θ^0 is the initial temperature vector and θ^t is the temperature at the end of an interval, the closed form solution is

$$\begin{aligned} \theta^t = {} & \exp\left(-\boldsymbol{A}^{-1} \times \boldsymbol{B} \times t\right) \times \theta^0 \\ & + \left(\boldsymbol{I} - \exp\left(-\boldsymbol{A}^{-1} \times \boldsymbol{B} \times t\right)\right) \times \boldsymbol{B}^{-1} \times \boldsymbol{P}. \end{aligned} \tag{25}$$

$\boldsymbol{I}$ is the identity matrix of size W and t is the length of the interval. Now, α and β matrices are defined as follows.

$$\boldsymbol{\alpha} = \exp\left(-\boldsymbol{A}^{-1} \times \boldsymbol{B} \times t\right) \tag{26}$$

$$\boldsymbol{\beta} = (\boldsymbol{I} - \boldsymbol{\alpha}) \times \boldsymbol{B}^{-1} \tag{27}$$

With the help of α and β, Eq. (25) could be written as

$$\theta^t = \boldsymbol{\alpha} \times \theta^0 + \boldsymbol{\beta} \times \boldsymbol{P}. \tag{28}$$

Equation (28) could be understood intuitively by thinking about the system being LTI. According to the superposition principle, the effects of the initial value and the input power will add up, thus the plus sign between the two terms. The scaling property of the system could also be verified rapidly, as the scaling of an input, θ^0 or $\boldsymbol{P}$ with a certain factor, will scale its own effect by the same factor.

The thermal simulation is done in two phases, an initialization phase and then the operational phase. In the initialization phase the model is invoked and based on it $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are computed (this is shown in Fig. 6 in regard to the overall scheduling method). The

operational phase is the iterative computation of the temperatures for different times using (28). Since the thermal model is time invariant, the initialization is done only once at the very beginning of the design process. Throughout the offline scheduling phase, only the iterative computations are performed.

In the closed form solution, the most computationally expensive part is the matrix exponential for α which is a part of (26). The matrix exponential could be computed using numerical methods such as Padé approximation [13]. In fact the initialization phase for the closed form solution includes calculating (26) and therefore it is very time consuming, however the operational phase only includes computing (28) and therefore it is fast.

On the other hand, for the Runge–Kutta approach [21], the initialization is fast since there is no need for computations which are as heavy as (26), but the operational phase is slow since the equation is required to be solved in many fine steps through large number of intermediate time instances. The conclusion is that the Runge–Kutta method is faster for limited number of simulations and the closed form method is faster for large number of simulations. The experiments in Section 7.1 will support this statement.

7 Experimental Results

Two distinct contributions of this paper, the thermal simulation approach and the scheduling technique are experimentally examined in this section. All experiments are performed on a desktop computer with Intel® Xeon® W3520 processor and 8 GB of memory. The experiments for thermal simulation are presented first.

7.1 Experiments for the Suggested Thermal Simulation Approach

A thermal simulation approach based on the closed form solution is suggested in Section 6 in order to elevate the simulation speed. The problem with numerical approximation approaches for temperature simulation is that they are very slow for large number of simulation cycles especially when there are a large number of cores. Thermal simulations for a SoC with 100 cores and for different numbers of simulation cycles are performed using the proposed approach and using HotSpot [14], and the CPU times are plotted in Fig. 12a.

The numerical approximation approaches, such as the one used by HotSpot, perform faster than the suggested approach for a small number of simulation cycles, as detailed in Section 6. But for simulations longer than 1,700 cycles, the proposed approach is faster than HotSpot, as shown in Fig. 12a. In general, this difference increases with a rate close to 0.011 s per cycle and it reached a CPU time difference of 100 s for 10,000 simulation cycles. This is important since for every edge in every candidate schedule tree thermal simulation is performed for the number of test cycles plus cooling cycles.

Thermal simulations are performed using the proposed approach and using HotSpot [14] for 10,000 simulation cycles for different numbers of cores, and the CPU times are plotted in Fig. 12b. In general, the CPU time difference increases rapidly with the number of cores and the difference reaches 100 s for 100 cores. This is also important, since achieving a good schedule in reasonable time becomes infeasible with a small increase in the number of cores, when the slower approach is in use.

7.2 Experiments for the Proposed Test Scheduling Technique

The proposed SoC test scheduling technique is experimentally evaluated in this section. The first set of experiments is performed on SoCs with different number of cores and the CPU times are reported. Then, experiments are done for ITC'02 [17] benchmark chips with random test switching activities generated using a Markov chain similar to [24]. Finally, an experiment is performed for the d695 benchmark chip from ITC'02 with real switching activities based on real test data from [22]. The costs of the test schedules and

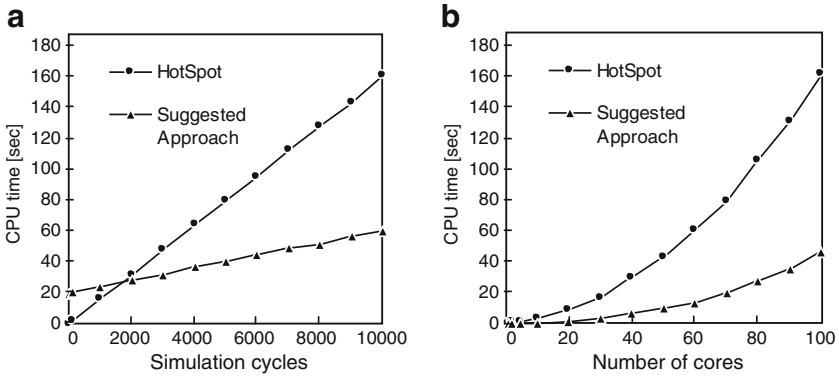


Fig. 12 CPU times for thermal simulation with HotSpot [14] and with the suggested approach. The simulations are performed **a** for 100 cores for different numbers of simulation cycles and **b** for 10,000 simulation cycles for different numbers of cores

Table 1 CPU times for SoCs with different number of cores

Number of cores	5	10	15	20	25	30	35	40	45	50
CPU time [Sec]	9	46	52	132	208	308	590	762	1141	1367

the test schedule sizes are reported for the last two sets of experiments. The experimental setup is briefly introduced at the beginning and then the results are presented.

The static power is computed using the temperature dependent model given in [16]. The temperature simulations are performed using the approach proposed in Section 6. The spatial temperature error is assumed to have normal distribution $\mathcal{N}(0, 10)$ ranging from −20 °C to +20 °C with a resolution of 0.5 °C (M=80). The temporal temperature error is also assumed to have a normal distribution $\mathcal{N}(0, 10)$ ranging from −40 °C to +40 °C with a resolution of 0.5 °C. It is assumed that there are 20 temperature error clusters (L=20).

The balancing coefficient is assumed to be equal to ten (BC=10). It is assumed that each entry in a linear schedule table occupies 64 bits and each entry in a branching table per core per edge occupies 32 bits. For example a node with two succeeding edges for a SoC with two cores, occupies $(2\times2\times32) = 128$ bits.

The first set of experiments is performed on a number of SoCs with different number of cores ranging from 5 to 50 cores. Markov chains are used to generate random test switching activity sequences having random averages and random lengths. The experiments are performed for at least five randomly generated sets of tests for each chip and the average CPU times are reported in Table 1. Note that even for a 50-core SoC, the CPU time remains in an affordable range.

The second set of experiments is performed on ITC'02 SoCs with randomly generated test switching activities similar to the first set of the experiments but this time tests for a chip have constant power averages and length. The proposed technique is compared with two traditional methods similar to the methods proposed in [3]. The first traditional method is an Offline method which uses only one linear schedule and the other traditional method is a Hybrid method which selects a linear schedule out of a set of pre-generated schedules only once during the test process. The test costs offered by traditional methods and by the proposed technique are computed according to Eq. (4) and are reported in Table 2.

The cost improvements reported in [1, 2] are at most 20 % better than the Offline method and at most 7 % better than the Hybrid method, while in this paper, in average, the adaptive method reduces the cost by 76 % over the Offline method and 43 % over the Hybrid method. The proposed adaptive method has reduced the cost by 76 % compared to the Offline method, while the cost reduction achieved by the Hybrid method is 52 %. This difference demonstrates the advantage of the proposed adaptive method.

The ATE memory occupied to store the schedules (i.e., the schedule size) is reported in Table 3. The cost reduction comes with increase in the schedule size because of increased number of linear schedules and branching tables, which consume ATE memory space. The average increase in schedule size compared to Offline is 87 % for Hybrid and 308 % for the proposed method. When compared to Hybrid, the average schedule size increase for the proposed method is 117 %. The increase in the schedule sizes reported in [1, 2] are in the range 187–291 % compared to Offline and

Table 2 Test cost for traditional and proposed test scheduling techniques

ITC'02 chips	Cost			Percentage reduction relative to the Offline		Percentage reduction relative to the Hybrid
	Offline	Hybrid	Proposed	Hybrid	Proposed	Proposed
a586710	1.44	0.56	0.54	61	62	4
d281	0.69	0.45	0.03	35	96	93
d695	0.50	0.12	0.06	76	88	50
f2126	2.71	1.39	0.51	49	81	63
g1023	5.09	4.27	1.99	16	61	53
h953	0.46	0.14	0.11	70	76	21
p22810	1.22	0.70	0.69	43	43	1
p34392	0.75	0.72	0.06	4	92	92
p93791	1.02	0.13	0.08	87	92	38
q12710	1.32	0.40	0.23	70	83	42
t512505	0.48	0.23	0.13	52	73	43
u226	1.05	0.43	0.37	59	65	14
Average				52	76	43

Table 3 ATE memory utilized only for schedule in traditional and in proposed techniques

ITC'02 chips	Utilized memory for schedule [bit]			Percentage increase relative to the Offline		Percentage increase relative to the Hybrid
	Offline	Hybrid	Proposed	Hybrid	Proposed	Proposed
a586710	1216	1888	4768	55	292	152
d281	1088	1280	2624	18	141	105
d695	1280	2176	3392	70	165	54
f2126	704	960	2368	36	236	147
g1023	576	1088	4480	89	678	312
h953	576	1088	1472	89	155	35
p22810	704	1888	5568	168	691	195
p34392	832	1472	2688	77	223	83
p93791	704	1920	3136	173	345	63
q12710	640	1024	1664	60	160	62
t512505	1152	2336	3712	103	222	59
u226	320	672	1568	110	390	133
Average				87	308	117

43–84 % compared to the Hybrid. These numbers fall within the ranges reported for the proposed method in this paper.

The increase in the usage of ATE memory (as given in Table 3) refers only to the memory space used to store the schedule. This is usually small, compared with the memory space used to store the test patterns. Therefore a large increase in the schedule size is very likely to be translated into a small increase in the usage of the ATE memory as a whole.

The proposed scheduling method will utilize the available ATE memory even if a very small reduction in cost (e.g., from 0.70 to 0.69 for p22810 in Table 2) is achieved. Since the number of nodes contributes to the scaled cost function (Eq. 6), a larger schedule will not be generated (e.g., 195 % larger for p22810 in Table 3 compared with hybrid solution) if it does not reduce the cost compared with a smaller schedule.

The ATE memory constraint will affect the quality of the adaptive test schedules. The proposed algorithm will not generate even an offline schedule when the available memory is too small. By increasing the available ATE memory, first an offline schedule and then a hybrid schedule will be generated. With the further increase of the memory constraint, better schedules with lower costs will be generated. This trend continues until the cost reaches a minimum beyond which further cost reduction is impossible. The minimum cost is usually dictated by the branching overheads (time to read sensors and react accordingly). Such a trend is experimentally demonstrated in [1].

The last experiment is performed on d695 (one of the ITC'02 chips) using the real test switching activities. The costs and schedule sizes are reported in Table 4. The Hybrid method improves the cost compared to Offline method by 59 % while the proposed adaptive technique achieves a reduction of 71 %. The proposed technique improves the cost by 30 % over Hybrid method. The schedule size for the proposed method is 169 % and 49 % larger than Offline and Hybrid, respectively. As we expected the improvement in cost and the increase in the schedule size are in the ranges suggested by the second set of experiments.

The test sequences are stored elsewhere and are not affected by the schedule. The test size is larger than the schedule size, and therefore the effect of increased schedule size on the total consumed ATE memory is small. For example consider the experiments with the d695 chip with real switching activities. The size of the schedule for the adaptive solution is approximately 7 Kbit while the test size is approximately 1,324 Kbit. Therefore the percentage increase in total utilized ATE memory from the offline solution to the adaptive solution is 0.34 %. This means that the

Table 4 Cost and ATE memory utilized for schedule for d695 with real test data

				Percentage change relative to the Offline		Percentage change relative to the Hybrid
	Offline	Hybrid	Proposed	Hybrid	Proposed	Proposed
Cost	20.84	8.53	5.93	−59	−71	−30
Utilized memory for schedule [bit]	2688	4992	7232	+86	+169	+49

adaptive method achieves 71 % reduction in cost relative to the offline method, with a small expense of 0.34 % increase in the occupied ATE memory.

8 Conclusion

This paper presents an adaptive SoC test scheduling technique to deal with spatial and temporal temperature deviations, caused by process variations in deep submicron technologies. The main contribution is an algorithm to generate a set of efficient test schedules, each corresponding to a different thermal behavior of different cores during test. The on-chip temperature sensors are used to monitor the actual temperatures of the different cores and to guide the selection of the corresponding test schedules accordingly, during the test. This way, the overall test efficiency will be improved considerably.

The proposed technique consists of two distinct algorithms, the test scheduler and the thermal simulator. The thermal-aware test scheduler is a constructive algorithm which generates tree-based test schedules by putting the optimized sub-trees together. Sub-tree optimization is basically a chip-clustering algorithm which involves a linear test scheduling algorithm. A new sub-tree scheduling algorithm is proposed in this paper. The linear scheduling algorithm requires a thermal simulator in its main loop. A fast thermal simulation approach is proposed in order to speed up the thermal-aware test scheduling algorithm.

The proposed adaptive test scheduling technique generates process-variation and temperature aware test schedules for SoCs with large number of cores. The algorithm has a relatively short run-time and generates high quality test schedules. The proposed technique has been experimentally evaluated using a number of experiments including ITC'02 benchmark SoCs. The proposed technique outperforms the Offline and the Hybrid methods in average by 76 % and 43 %, respectively.

9 Quick Reference

Notation	Description	Equation
$\boldsymbol{A}$	Capacitances vector in the thermal model	24–26
ATS	Applied Test Size	2–4
$\boldsymbol{B}$	Resistances vector in the thermal model	24–27
BC	Balancing Coefficient	4
Branching table	The table that determines with which linear schedule table a specific chip should be tested. (See the example in Section 2)	
C	Number of cores	
Chip cluster	A group of chips with similar thermal behavior that are tested with the same **Linear schedule table**. A chip cluster corresponds to an edge in the schedule tree.	
Chip-cluster border	The border line between two **Chip clusters**. For two adjacent **Chip clusters** the border is a set of natural numbers, each corresponding to an individual core. A border represents a particular error value. (See Section 5.4)	
Chip clustering	In short, it is finding the optimal partitioning of the *C*-dimensional error space into an already known number of **Chip clusters** for the nodes of a tree. (See full explanation in Section 5.4)	
CTF	Cost of the Test Facility per time unit	1
EATS	Expected Applied Test Size	7
EB	temperature Error-clusters Borders	9, 11
ECCP	Error Cell Change Probabilities	10, 14
$ECCP^{\text{before normalization}}$	***ECCP*** before being normalized	9–10
ECL	Error-Cell Labeling	12
ECP	Error-Cells Probabilities	11–12, 17
$ECP^{\text{after branching}}$	***ECP*** just after branching	12–13, 15
$ECP^{\text{after overheating}}$	***ECP*** just after overheating	13–14, 16
$ECP^{\text{after temporal changes}}$	***ECP*** after temporal changes (according to temperature error model)	14, 17
Error cluster	A range of error values which are to be treated as one single error value. Error clusters are separated by Error-clusters Borders, ***EB***.	
ETAT	Expected Test Application Time	8
ETOP	Expected Test Overheating Probability	19–20
ETTpS	Effective Test Time per Second	2
HTT	High Temperature Threshold	13
$\boldsymbol{I}$	Identity matrix	25, 27
Knee	The point that the output is equal to the input and not equal to one, in the proposed reciprocal limiter.	21
L	Number of temperature error clusters	
Linear schedule table	A schedule that specifies stop/start times for the test of all cores individually. This will correspond to an edge or to a single **Chip cluster**. (See the example in Section 2)	

LOP	Leaves' Overheating Probabilities	18–19
M	Number of temperature error values	
N	Number of nodes in a tree	
NATS	Nodes' Applied Test Sizes	7
NCF	Normalized Cost Function	4, 6
NCL	Node's Cluster Label	12
NnOP	Node's not Overheating Probability	16, 18
Node	A node in the schedule tree that corresponds to the ending of a **Linear schedule table** (i.e., a place that branching is possible).	
NP	Nodes' Probabilities	7–8, 15, 19
NPT	expected Number of Partial Trees, similar to the current partial tree, that are required to construct the complete schedule tree	21
NTAT	Nodes' Test Application Times	8
NTT	Normalized Test Throughput	3–4
P	Power vector	24–25, 28
Partial cost function	***NCF*** evaluated for a part of the schedule tree (e.g., a sub-tree).	
POC	Price of One Chip	1
PSO	Particle Swarm Optimization	
PTOP	Predicted Test Overheating Probability	20
Q	Number of leaf nodes	19
S	Number of succeeding edges for a node	
SCF	Scaled Cost Function that is used to select the unfinished trees out of a group of offspring trees.	6
ST	Simulated Temperature	13
STEP	Spatial Temperature Error Probabilities	11
TAM	Test Access Mechanism	
TAT	Test Application Time	2–4
Temporal error period	The period for the discrete-time temperature error model. The error values are updated regularly with a frequency equal to 1/*Temporal_error_period*. (See Section 4)	
TEV	Temperature Error Values	5, 9
THT	Test Handling Time	2
TOP	Test Overheating Probability	1–4
TT	Test Throughput	1–2
TTEP	Temporal Temperature Error Probability	5, 9
W	Number of nodes in the thermal model	
α	Transfer matrix for initial temperatures	26–28
β	Transfer matrix for power values	27–28
Θ	Temperatures vector in thermal model	24
θ^0	Initial temperatures	25, 28
θ^t	Temperatures at the end of the interval of size t	25, 28
	Temperatures at t-th time sample (in Section 2)	
θ_w	Temperature of w-th thermal node	
λ	The output of the proposed limiter, applied on the expected number of partial trees, ***NPT***. $\lambda \leq (\boldsymbol{knee}+1)$	20–21

Acknowledgments The authors would like to thank Soheil Samii for providing cycle-accurate test switching-activities data for SoC d695.

References

1. Aghaee N, Peng Z, Eles P (2011) Adaptive temperature-aware SoC test scheduling considering process variation. Digit syst des, pp 197–204
2. Aghaee N, Peng Z, Eles P (2011) Process-variation and temperature aware soc test scheduling using particle swarm optimization. Int des and test, pp 1–6
3. Aghaee N, He Z, Peng Z, Eles P (2010) Temperature-aware SoC test scheduling considering inter-chip process variation. Asian test symposium, pp 395–398
4. Bonhomme Y, Girard P, Landrault C, Pravossoudovitch S (2002) Test power: a big issue in large SOC designs. Electron des, test, and appl, pp 447–449
5. Chandran U, Zhao D (2009) Thermal driven test access routing in hyper-interconnected three-dimensional System-on-Chip. Defect and fault toler in VLSI syst, pp 410–418
6. Clabes JG et al (2004) Design and implementation of the POWER5 microprocessor. Des autom conf, pp 670–672
7. Cheng K-T, Dey S, Rodgers M, Roy K (2000) Test challenges for deep sub-micron technologies. Des autom conf, pp 142–149
8. Choi JH, Murthy J, Roy K (2007) The effect of process variation on device temperature in finFET circuits. Comput-aided des, pp 747–751
9. He Z, Peng Z, Eles P (2010) Multi-temperature testing for core-based system-on-chip. Des, autom and test in Eur, pp 208–213
10. He Z, Peng Z, Eles P (2009) Thermal-aware test scheduling for core-based SoC in an Abort-on-First-Fail test environment. Digit syst des, pp 239–246
11. He Z, Peng Z, Eles P (2008) Simulation-driven thermal-safe test time minimization for System-on-Chip. Asian test symposium, pp 283–288
12. He Z, Peng Z, Eles P, Rosinger P, Al-Hashimi BM (2008) Thermal-aware SoC test scheduling with test set partitioning and interleaving. J Electron Testing 24(1–3):247–257
13. Higham NJ (2005) The scaling and squaring method for the matrix exponential revisited. SIAM J Matrix Anal Appl 26(4):1179–1193
14. Huang W, Stan MR, Skadron K, Sankaranarayanan K, Ghosh S, Velusamy S (2004) Compact thermal modeling for temperature-aware design. Des autom conf, pp 878–883
15. IEEE P1687 (IJTAG) http://grouper.ieee.org/groups/1687/
16. Liao WP, He L, Lepak KM (2005) Temperature and supply voltage aware performance and power modeling at microarchitecture level. IEEE Trans Comput Aided Design 24(7):1042–1053

17. Marinissen EJ, Iyengar V, Chakrabarty K (2002) A Set of Benchmarks for Modular Testing of SOCs. Int test conf, pp 519–528
18. Nebel W, Mermet J (1997) Low power design in deep submicron electronics. Kluwer, Dordrecht
19. Oppenheim AV, Willsky AS, Nawab SH (1996) Signals and systems, 2nd edn. Prentice-Hall, Upper Saddle River
20. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization, an overview. Swarm Intell 1(1):33–57
21. Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2002) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge University Press, New York
22. Samii S, Selkala M, Larsson E, Chakrabarty K, Peng Z (2008) Cycle-accurate test power modeling and its application to SoC test architecture design and scheduling. IEEE Trans Comput Aided Des Integr Circuits Syst 27(5):973–977
23. Yang Y, Gu Z, Zhu C, Dick RP, Shang L (2007) ISAC: integrated space-and-time-adaptive chip-package thermal analysis. IEEE Trans Comput Aided Des Integr Circuits Syst 26(1):86–99
24. Yao C, Saluja KK, Ramanathan P (2011) Thermal-aware test scheduling using on-chip temperature sensors. VLSI des, pp 376–381
25. Yao C, Saluja KK, Ramanathan P (2011) Temperature dependent test scheduling for multi-core system-on-chip. Asian test symposium, pp 27–32
26. Yao C, Saluja KK, Ramanathan P (2009) Partition based SoC test scheduling with thermal and power constraints under deep submicron technologies. Asian test symposium, pp 281–286
27. Yu TE, Yoneda T, Chakrabarty K, Fujiwara H (2009) Test infrastructure design for core-based system-on-chip under cycle-accurate thermal constraints. Asia and s pac des autom conf, pp 793–798
28. Zorian YA (1993) Distributed BIST control scheme for complex VLSI devices. VLSI test symposium, pp 4–9

Nima Aghaee is a Ph.D. student at Embedded Systems Laboratory, Linkoping University, Sweden. His research interests include test technology and digital signal processing. He has a M.Sc. in embedded systems design from ALaRI Institute, Switzerland, a M.Sc. in electronics engineering from TMU University, Iran, and a B.Sc. in electronics engineering from SBU University, Iran.

Zebo Peng received the Ph.D. degree in computer science from Linkoping University, Linkoping, Sweden, in 1987. Currently, he is a Professor of computer systems and Director of the Embedded Systems Laboratory, Linkoping University. His research interests include design and test of embedded systems, SoC testing, hardware/software co-design, and real-time systems. He has published over 300 technical papers and co-authored several books in these areas.

Petru Eles is a Professor of embedded computer systems with the Department of Computer and Information Science (IDA), Linkoping University, Linkoping, Sweden. His current research interests include embedded systems, real-time systems, electronic design automation, cyber-physical systems, hardware/software codesign, low power system design, fault-tolerant systems, design for test. He has published a large number of technical papers in these areas and co-authored several books.