Temperature-Aware Idle Time Distribution for Leakage Energy Optimization

Min Bao[1], Alexandru Andrei [1,2] , Petru Eles [1], Zebo Peng [1]

***Abstract*—Large-scale integration with deep sub-micron technologies has led to high power densities and high chip working temperatures. At the same time, leakage energy has become the dominant energy consumption source of circuits due to reduced threshold voltages. Given the close interdependence between temperature and leakage current, temperature has become a major issue to be considered for power-aware system level design techniques. In this paper, we address the issue of leakage energy optimization through temperature aware idle time distribution (ITD). We first propose an off-line ITD technique to optimize leakage energy consumption, where only static idle time is distributed. To account for the dynamic slack, we then propose an on-line ITD technique where both static and dynamic idle time are considered. Experimental results have demonstrated that an important amount of leakage energy reduction can be achieved by applying our ITD techniques. To improve the efficiency of our ITD techniques, we also propose an analytical temperature analysis approach which is accurate and, yet, sufficiently fast to be used inside the energy optimization loop.**

***Index Terms*—Temperature aware design, leakage energy optimization, idle time distribution, system level design.**

I. Introduction

A. Background

Technology scaling and ever increasing demand for performance have resulted in high power densities in current circuits, which have also led to increased chip temperature. At the same time, leakage energy has become the dominant energy consumption source of circuits [1]. Due to the strong dependence of leakage current on temperature, growing temperature leads to an increase in leakage current and, consequently, energy, which, again, produces higher temperature. Thus, temperature is an important parameter to be taken into consideration for energy optimization.

Energy optimization for embedded systems has been extensively researched. At system level, dynamic voltage selection (DVS) is one of the preferred approaches for reducing the overall energy consumption [2], [3]. This technique exploits the available slack time to achieve energy efficiency by reducing the supply voltage and frequency such that the execution of tasks is stretched within their deadline.

There are two types of slacks: (1) static slack, which is due to the fact that, when executing at the highest (nominal) voltage level, tasks finish before their deadlines even when executing their worst numbers of cycles (WNC); (2) dynamic slack, due to the fact that most of the time tasks execute less than their WNC. Off-line DVS techniques [4], [5] can only exploit static slack, while on-line approaches [6], [7], [8] are able to further reduce energy consumption by exploiting the dynamic slack.

However, very often, not all available slack should or can be exploited and certain slack may still exist after DVS. An obvious situation is when the lowest supply voltage is such that, even if selected, a certain slack interval is left. Another reason is the existence of the critical voltage [9]. To achieve the optimal energy efficiency, DVS would not execute a task at a voltage lower than the critical one, since, otherwise, the additional static energy consumed due to the longer execution time is larger than the energy saving due to the lowered voltage. During the available slack interval, the processor remains idle and can be switched to a low power state.

Due to the strong inter-dependence between leakage power and temperature, different distributions of idle time will lead to different temperature distributions and, consequentially, energy consumption. In this paper, we address the issue of optimizing leakage energy consumption through distribution of both static and dynamic slack time.

B. System Level Temperature Modeling

Temperature aware system level design methods rely on the availability of temperature modeling and analysis tools. System level temperature modeling approaches are mostly based on the duality between heat transfer and electrical phenomena [10]. Hotspot [11] is both an architectural level and system level temperature simulator. The basic idea of Hotspot is to build an equivalent circuit of thermal resistances and capacitances capturing both the architecture blocks and the elements of the thermal package. In [12], a similar temperature modeling approach was proposed which speeds up the thermal analysis through dynamic adaptation of the resolution.

However, temperature analysis time with approaches like the two mentioned above are too long to be affordable inside a temperature aware system level optimization loop. There has been some work on establishing fast system level temperature analysis techniques. They also build on the duality between heat transfer and electrical phenomena and are based on very restrictive assumptions in order to simplify the model. In [13], the authors have assumed that (1) no cooling layer is present, (2) there is no interdependency between leakage current and temperature, and (3) the whole application executes at a constant voltage. The models in [14] and [15] consider variable voltage levels but maintain the first two limitations above. The most general analytical model is proposed in [16] which considers cooling layers as well as the dependency between leakage and temperature. However, this approach is limited to the case of a unique voltage level throughout the application. In order to support our ITD technique proposed in this paper, we introduce a fast and accurate temperature analysis technique, which eliminates all three limitations mentioned above and can be used inside a temperature aware system level optimization loop.

C. Temperature Sensing and Tracking

In this paper, in addition to an off-line approach, we also propose an on-line ITD approach which relies on on-line temperature monitoring, where sensors [17] are used together

[1]Dept. of Computer and Information Science, Linköping University, Linköping, 58183, Sweden

[2]Ericsson AB, Linkoping, Sweden

with techniques for collecting and analyzing their values with adequate accuracy. Several approaches have been proposed in literature to improve the accuracy of temperature measurement and estimation. For example, in [18] and [19], authors have proposed techniques to determine the optimal locations and allocations for thermal sensors with the target of accurate hot spots detection as well as full chip thermal characterization. In [20], [21], and [22] the authors have addressed the issue of how to process/analyze readings from sparse and noisy thermal sensors to accurately estimate temperatures where various estimation schemes such as spectral methods and Kalman filters are utilized.

D. Related Work

Several approaches to system level temperature aware design have been discussed in literature. Temperature management is utilized to control the temperature of processors for improving system reliability [23]. In [24], the authors proposed a technique for temperature management by scaling the processor speed. In [25], the authors addressed the issue of scheduling and mapping of a set of tasks with real-time constraints on multi-processors for peak temperature minimization. Techniques for task sequencing combined with DVS to reduce the peak temperature of a processor were proposed in [14]. Several approaches aiming at reducing temperature variations or temperature gradients across the chip, e.g. [26], were proposed.

A considerable amount of work has been published on performance optimization under thermal and real-time constraints. Zhang et al. [27] proposed voltage assignment techniques to optimize the performance of a set of periodic tasks working under thermal constraints. In [28], the authors proposed approaches to optimize throughput by task sequencing under thermal constraints. An on-line speed adaptation technique for homogeneous multi-processors with the target of maximizing total throughput was proposed by Rao et al. in [29]. Temperature aware DVS techniques considering the leakage/temperature dependency were proposed in [30] and [3].

In this paper we address the issue of optimizing leakage energy consumption through distribution of idle time. The only work, to our best knowledge, previously addressing this issue is [31] and [32]. In [31], the authors proposed an approach to distribute idle time for applications consisting of one single task executing at a constant given supply voltage. Thus, their approach cannot optimize the distribution of idle time among multiple tasks which also execute at different voltages. The same limitation also holds for [32], where a pattern based ITD for leakage energy optimization considering one single task was proposed. The pattern based approach generates uniform idle time distribution over the whole application and, thus, is not appropriate for ITD among multi-task applications where tasks have different amounts of energy consumption and execute at different voltage levels.

E. Main Contributions

In this paper, we make the following main contributions: [1]

1) We propose an off-line ITD approach to optimize leakage energy consumption for a set of periodic tasks. Static slack is distributed globally among tasks which are executed at different discrete voltage levels.
2) We propose, based on the off-line ITD approach, an on-line ITD technique where both static and dynamic slack are distributed.
3) We propose a fast and accurate analytical temperature model which eliminates all the three limitations mentioned in Section I-B, by considering the following aspects: a) the interdependence between leakage power and temperature; b) multiple cooling layers of the chip; c) non-smooth power consumption generated due to multiple discrete supply voltage levels of the processor.

[1]Preliminary results regarding the off-line ITD approach have been published in *"Temperature-Aware Idle Time Distribution for Energy Optimization with Dynamic Voltage Scaling"* in the Proc. of DATE, 2010 [33].

F. Paper Organization

In Section II we introduce the power and application models. In Section III we give a motivational example. We formulate the problem in Section IV. In Section V we introduce our analytical thermal model. We then propose the off-line ITD approach, which distributes only static slack, in Section VI. Based on the off-line ITD approach, we present our on-line ITD technique in Section VII. Finally, experimental results and conclusions are presented in Sections VIII and IX.

II. Preliminaries

A. Power Model

For dynamic power we use the following equation [34]:

$$P^d \;=\; C_{eff} \cdot f \cdot V^2$$

where C$_{eff}$, V, and f denote the effective switched capacitance, supply voltage, and frequency, respectively.

The leakage power is expressed as follows [35]:

$$P^{leak} = I_{sr} \cdot T^2 \cdot e^{\frac{\beta \cdot V + \gamma}{T}} \cdot V \qquad (1)$$

where I_{sr} is the leakage current at a reference temperature, T is the current temperature, and β and γ are technology dependent coefficients. In Section V-B we will use a piecewise linear approximation of this model, as proposed, for example, in [36]. According to it, the working temperature range $[T_a, T_{max}]$ where T_a and T_{max} are the ambient and the maximal working temperature of the chip, is divided into several sub-ranges. The leakage power inside each sub-range $[T_i, T_{i+1}]$ is modeled by a linear function: $P_i = K_i \cdot T + B_i$, where K_i and B_i are constants characteristic to each interval.

B. Application Model

The application is captured as a task graph $G(\Pi, \Gamma)$. A node $\tau_i \in \Pi$ represents a computational task, while an edge $e \in \Gamma$ indicates the data dependency between two tasks. Each task τ_i is characterized by the following six-tuple:

$$\tau_i =< WNC_i, BNC_i, ENC_i, dl_i, Ceff_i, V_i >$$

where WNC_i, BNC_i and ENC_i are task τ_i's worst case, best case and expected number of clock cycles to be executed. The expected number of clock cycles ENC_i is the arithmetic mean value of the probability density function of the actual executed cycles ANC_i, i.e. $ENC_i = \sum_{j=BNC_i}^{WNC_i} (j \cdot p^i(j))$, where $p^i(j)$ is the probability that a number j of clock cycles are executed by task τ_i. We assume that the probability density functions of the execution cycles of different tasks are independent. V_i represents the supply voltage at which the task τ_i is executed. The supply voltage V_i can be either constant for all tasks, or it can be calculated by a DVS algorithm, e.g. our temperature aware DVS technique proposed in [30]. Further, dl_i and $Ceff_i$ represent the deadline and the effective switched capacitance.

The application is mapped and scheduled on a processor which has two power states: active and idle. In the active state the processor can operate at several discrete supply voltage levels. When the processor does not execute any task, it can be put to the idle state, consuming a very small amount of leakage power P_{idle}. The variation of P_{idle} with temperature is neglected due to the very small value of P_{idle}. Switching the processor between the idle and active state (as well as between different voltage levels) incurs time and energy overheads.

III. Motivational Example

A. Static Idle Time Distribution

Let us consider an application consisting of 7 tasks which share a global deadline of 96.85ms. The worst case workload, WNC (in clock cycles), and average switched capacitance, C_{eff}, are given in Table I. The tasks run on a processor with a fixed supply voltage and frequency of $0.6V$ and 132MHZ, respectively. The corresponding execution times te^W are given in Table I. Based on the performance of this processor, there exists 6ms static slack, ts, in each execution period of this application. Fig. 1 gives two ways of distributing ts. The first distribution (1st ITD), as shown in Fig. 1a, places the whole ts after the last task, while the second distribution (2nd ITD), in Fig. 1b, divides the static slack ts into 3 segments and places the 3 idle slots after execution of task τ_3, τ_4 and τ_5, respectively.

TABLE I
Motivational Example: Application Parameters

	WNC	C_{eff}(f)	te^W(ms)
τ_1	8.26e+06	5.0e-10	6.22
τ_2	1.20e+07	5.0e-10	9.07
τ_3	2.32e+07	9.0e-8	18.76
τ_4	2.25e+07	1.7e-7	17.46
τ_5	1.46e+07	1.8e-7	16.94
τ_6	2.15e+07	1.9e-7	16.18
τ_7	8.26e+06	5.0e-10	6.22

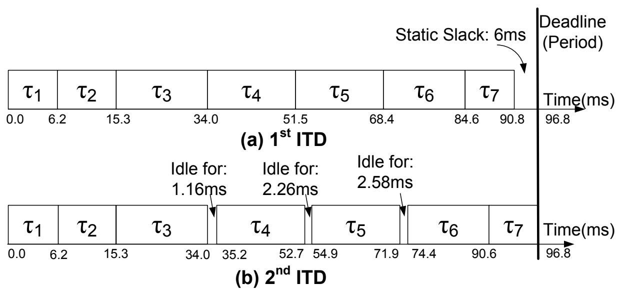


Fig. 1. Motivational Example: Static Idle Time Distribution

For simplicity, in this example, we ignore both energy and time overhead due to switching between the active and idle mode. The two different ITDs will lead to different temperature and leakage power profiles. The average working temperature Tw of each task, as well as the leakage energy E^{leak} consumption, are shown in Table II. E^{leak}_{tot} is the total leakage energy consumption of the whole application. Comparing E^{leak}_{tot} for the 1st and 2nd ITD, we can observe that around 10% reduction of leakage energy consumption can be achieved.

The leakage energy reduction is due to the modified working temperature of the chip which has a strong impact on the leakage power. It is also important to mention that the table reflects the steady state (not the start-up mode), for which energy minimization is targeted. This means that the starting temperature for τ_1 is identical to the temperature at the end of the previous period.

TABLE II
Static ITD: Leakage Energy Comparison

	1st ITD		2nd ITD	
	Tw(°C)	E^{leak}(J)	Tw(°C)	E^{leak}(J)
τ_1	101	0.81	110	0.96
τ_2	102	1.20	107	1.30
τ_3	108	2.73	108	2.73
τ_4	119	3.08	113	2.78
τ_5	125	3.32	115	2.79
τ_6	129	3.39	117	2.68
τ_7	122	1.24	117	1.05
E^{leak}_{tot}		15.77		14.29

B. Dynamic Idle Time Distribution

The ITD approach outlined in the previous section is an off-line static one which assumes that tasks execute their WNC and, thus, it only distributes the static slack. However, in reality, most of the time, there are huge variations in the number of cycles executed by a task, from one activation to the other, which leads to a large amount of dynamic slack.

For the task set introduced in the previous section, let us imagine the activation scenario given in Table III where the columns ANC and te^A contain the actual executed workload (in clock cycles) and the corresponding actual execution time of each task, respectively. td^i represents the dynamic slack generated due to the actual number of cycles executed by task τ_i (it is the difference between the te^W and te^A of the task). For this activation scenario, tasks τ_3, τ_4, τ_5 and τ_6 execute their worst case workload, while τ_1, τ_2 and τ_7 execute less than their worst case workload and, thus, generate dynamic slack. The total amount of dynamic slack is $td = \sum_{i=1}^{7} td^i = 18.7$ms.

TABLE III
Motivational Example: An activation Scenario

	ANC	te^W(ms)	te^A(ms)	td^i(ms)
τ_1	5.95e+05	6.22	0.45	5.77
τ_2	5.20e+05	9.07	0.40	8.67
τ_3	2.49e+07	18.76	18.76	0.0
τ_4	2.32e+07	17.46	17.46	0.0
τ_5	2.25e+07	16.94	16.94	0.0
τ_6	2.15e+07	16.18	16.18	0.0
τ_7	2.60e+06	6.22	1.96	4.26

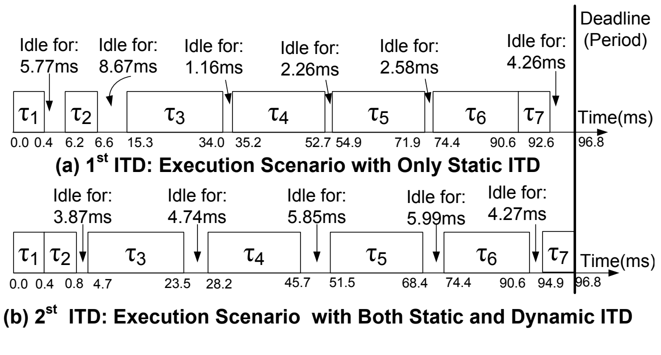


Fig. 2. Motivational Example: Idle Time Distribution

Fig. 2a illustrates the distribution of idle time slots during the above on-line activation scenario if we use the off-line ITD approach which distributes static slack as illustrated in Fig. 1b. In this case, the dynamic slack td^i is placed where it is generated (td^i is placed after τ_i terminates). Table IV shows the corresponding working temperature and leakage energy consumption of each task as well as the total leakage energy consumption, which is 7.98J. However, leakage energy can be reduced by distributing the dynamic slack more wisely. For example, at run-time, whenever a task terminates, the idle time slot length following this task is calculated by taking into consideration the current time and the current chip temperature. Fig. 2b shows the ITD determined in this way. The corresponding total leakage energy consumed, as given in Table IV, is 7.32J which means a leakage energy reduction of 8%. This reduction is due to the further lowered working temperature of the energy hungry tasks τ_4, τ_5 and τ_6, which is achieved by ITD considering both static and dynamic slack.

TABLE IV
Dynamic ITD: Leakage Energy Comparison

	1st ITD		2nd ITD	
	Tw(°C)	E^{leak}(J)	Tw(°C)	E^{leak}(J)
τ_1	89	0.05	83	0.04
τ_2	78	0.03	83	0.04
τ_3	79	1.67	84	1.80
τ_4	91	1.92	87	1.78
τ_5	97	2.04	90	1.80
τ_6	99	2.02	91	1.73
τ_7	102	0.25	84	0.13
E^{leak}_{tot}		7.98		7.32

The above examples have demonstrated that leakage energy can be reduced through both static and dynamic ITD.

IV. Problem Formulation

We consider a set of periodic tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ executed in the order τ_1, τ_2, …, τ_n. For each task τ_i, the six-tuple: $< WNC_i, BNC, ENC, dl_i, Ceff_i, V_i >$ is given. Corresponding to the supply voltage V_i that task τ_i is executed at, the worst case execution time te_i^W, best case execution time te_i^B, and expected execution time te_i^E can be directly calculated.

For each iteration of the application, the total static slack ts is constant and computed by Eq. (2):

$$ts = dl_n - \sum_{i=1}^{n} te_i^W \tag{2}$$

where dl_n represents the deadline of the last task τ_n in the execution order, and $\sum_{i=1}^{n} te_i^W$ is the sum of the worst case execution time of all tasks. The total dynamic slack for each execution iteration is varying due to execution time variation of tasks. For one iteration, td is calculated as follows:

$$td = \sum_{i=1}^{n} te_i^W - \sum_{i=1}^{n} te_i^A$$

where te_i^A represents the actual execution time of task τ_i in this iteration. te_i^A conforms to a distribution with the expected execution time te_i^E as the arithmetic mean value of the probability density function $Pb(te_i^A)$: $te_i^E = \int_{te_i^B}^{te_i^W} Pb(te_i^A) \cdot te_i^A \, \mathrm{d}(te_i^A)$.

The total available slack t_{tot} for one iteration is equal to the sum of the static slack ts and dynamic slack td: $t_{tot} = ts + td$. During t_{tot} the processor can be switched to idle mode consuming the power P_{idle}. The time and energy overhead for switching the processor to and from the idle state are t_o and E_o respectively. Idle slots can be placed after the execution of any task. The length of an idle slot i after task τ_i is denoted as t_i, and the sum of all idle slots $\sum_{j=1}^{n} t_i$ should be equal with the total available idle time t_{tot}. Note that the time overhead t_o is included in the slot length t_i.

We will, formulate the following two ITD problems:

1) ITD with only static slack: static idle time distribution (SITD)
2) ITD with both static and dynamic slack: static and dynamic idle time distribution (DITD)

A. ITD with only Static Slack: SITD

Let us consider the scenario in which each task τ_i is always executed with the worst case workload: $te_i^A = te_i^W$. In this scenario, for each iteration, the available slack is constant and known: $t_{tot} = ts$ where ts is computed by Eq. (2).

For one iteration, the total energy consumption of the task set can be expressed as follows:

$$E^{tot} = \sum_{i=1}^{n} E_i^{dyn} + \sum_{i=1}^{n} E_i^{leak} + \sum_{i=1}^{n} (E_o \cdot x_i) + E^I$$

where $\sum_{i=1}^{n} E_i^{dyn}$ and $\sum_{i=1}^{n} E_i^{leak}$ are the total dynamic and leakage energy of all tasks. $\sum_{i=1}^{n} (E_o \cdot x_i)$ is the total energy overhead when the processor is switched to/from idle state, where x_i is a binary variable indicating whether task τ_i is followed ($x_i = 1$) or not ($x_i = 0$) by an idle slot. E^I is the total energy consumption during the idle time t_{tot}.

The dynamic energy consumption of each task E_i^{dyn}, is further computed as:

$$E_i^{dyn} = P_i^d(V_i) \cdot te_i^W$$

where V_i is the supply voltage the task τ_i is executed at. te_i^W represents the worst case execution time of task τ_i. As the supply voltage V_i and te_i^W are constants, the total dynamic energy $\sum_{i=1}^{n} E_i^{dyn}$ is hence constant and independent from the distribution of idle time.

The total energy consumption during idle time E^I is:

$$E^I = P_{idle} \cdot t_{tot}$$

where P_{idle} is the power consumption of the processor in the low power mode and $t_{tot} = ts$. Similar to E_i^{dyn}, E^I is also fixed and independent from ITD, as ts is constant with given supply voltages.

The leakage energy consumption of each task E_i^{leak} is a function of both temperature and supply voltage:

$$E_i^{leak} = \int_0^{te_i^W} P_i^{leak}(V_i, T_i(t)) \, \mathrm{d}t \tag{3}$$

where $T_i(t)$ describes the temperature of the processor during execution of task τ_i. With given supply voltages V_i, $T_i(t)$ is influenced by the distribution of idle time slots, so the leakage energy consumption E_i^{leak} depends on the ITD.

We need to distribute the static slack to minimize the total leakage energy consumption and the energy overheads due to switching: $\sum_{i=1}^{n} E_i^{leak} + \sum_{i=1}^{n} (E_o \cdot x_i)$. With given supply voltages V_i, and a fixed distribution of idle time slots, the same power pattern is periodically executed on the processor. As the task set is executed for a large number of iterations, the processor temperature is converging to a steady state dynamic temperature curve (SSDTC). Once the processor has reached steady state, the SSDTC will repeat periodically.

Our SITD problem can be formulated as follows: given is a set of tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ as defined earlier in this section. The idle time slot length t_i following each task τ_i and, implicitly, x_i (the binary variable which represents whether task τ_i is followed by an idle time slot or not) are to be determined such that the objective function Eq. (4) is minimized with the constraints Eq. (5) and Eq. (7) to be satisfied. Es_i^{leak} in

Problem Formulation 1 Static Idle Time Distribution

$$E = \sum_{i=1}^{n} Es_i^{leak} + \sum_{i=1}^{n} (E_o \cdot x_i) \tag{4}$$

subject to :

$$ts = \sum_{i=1}^{n} t_i \tag{5}$$

$$dl_i \geq \sum_{j=1}^{i-1} t_i + \sum_{j=1}^{i} te_j^W \; (\forall i, 1 \leq i \leq n) \tag{6}$$

$$T_i(t) \leq T_{max} \; (\forall i, 1 \leq i \leq n) \tag{7}$$

Eq. (4) represents the steady state leakage energy consumption. The constraint in Eq. (5) requires that the sum of all idle slots lengths should be equal with the total available static slack ts where ts is calculated by Eq. (2). The constraint in Eq. (6) guarantees that the deadline of each task is satisfied. Finally, the constraint in Eq. (7) requires that the processor temperature throughout the execution of the task set should not exceed the maximal allowable working temperature of the chip T_{max}, where $T_i(t)$ describes the processor temperature during execution of task τ_i.

B. ITD with both static and dynamic slack: DITD

The above problem formulation ignores the execution time variations of tasks at run-time and, implicitly, ignores the dynamic slack. To deal with execution time variation and perform dynamic slack distribution, the idle slot length t_i following the termination of a task τ_i should be determined, at run-time, based on the actual time and processor temperature.

Our problem formulation for DITD is as follows: given is a set of periodic tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ as defined earlier in this section. When task τ_i terminates at time t_i^f, the idle time slot t_i following task τ_i's termination is determined such that Eq. (8) is minimized, where $\sum_{j=i+1}^{n} E_j^{leak}$ is the total leakage energy consumption of the remaining tasks τ_j, $(\forall j, i < j \leq n)$, to be executed within the current iteration. The leakage energy consumption E_j^{leak} of each remaining task τ_j is estimated corresponding to the case when the expected workload is executed. E_j^{leak} is calculated according to Eq. (3) with the difference that the expected execution time te_j^E is used instead of te_j^W as the upper limit for the integral. The constraint in Eq. (9) requires that the sum of all idle slots lengths should be equal with the total available slack where t_i^f is the time the current task τ_i terminates. The total available slack is computed with the assumption that all the future tasks τ_{i+1} to τ_n are executed with their expected workload te_j^E $(\forall j, i < j \leq n)$. The deadline of each task is guaranteed by the constraint in Eq. (10), where dl_j represents the deadline of task τ_j. Note that, the worst case execution time te_k^W is used in Eq. (10) in order to guarantee the deadline of each task in the worst case. The constraint in Eq. (11) requires, similarly to Eq. (7), the processor temperature during execution of the task set to be lower than the maximal allowable working temperature of the chip T_{max}.

Problem Formulation 2 Dynamic Idle Time Distribution

Minimize :

$$E = \sum_{j=i+1}^{n} E_j^{leak} + \sum_{j=i}^{n} (E_o \cdot x_j) \tag{8}$$

subject to :

$$\sum_{j=i}^{n} t_j = dl_n - t_i^f - \sum_{j=i+1}^{n} te_j^E \tag{9}$$

$$dl_j \geq t_i^f + \sum_{k=i}^{j-1} t_k + \sum_{k=i+1}^{j} te_k^W \quad (\forall j, i+1 \leq j \leq n) \tag{10}$$

$$T_i(t) \leq T_{max} \; (\forall i, 1 \leq i \leq n) \tag{11}$$

V. Temperature Analysis

A. Temperature Model

Thermal Circuit.

In order to analyze the thermal behavior, we build an equivalent RC thermal circuit based on the physical parameters of the die and the package [10]. Due to the fact that the application period t_p can safely be considered significantly smaller than the RC time of the heat sink, which, usually, is in the order of minutes [37], the heat sink temperature stays constant after the state corresponding to the SSDTC is reached. For SSDTC estimation, we, hence, can ignore the thermal capacitance (not the thermal resistance!) of the heat sink and build the 2-RC thermal circuit shown in Fig. 3a. B_1 and B_2 represent the temperature node for the die and the heat spreader respectively. $P(t)$ stands for the processor power consumption as a function of time. We obtain the values of R_1, R_2, C_1 and C_2 from an RC network similar to the one constructed in Hotspot [11]. R_1 is calculated as the sum of the thermal resistance of the die and the thermal interface material (TIM), and C_1 as the sum of the thermal capacitance of the die and the TIM. R_2 is the equivalent thermal resistance from the heat spreader to the ground through the heat sink, and C_2 is the equivalent thermal capacitance of the heat spreader layer.

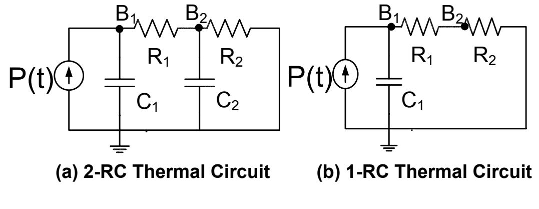


Fig. 3. Thermal Circuit

When the application period t_p is significantly smaller than the RC time of the heat *spreader* in the 2-RC thermal circuit, the heat *spreader* temperature stays constant after SSDTC is reached. In this case, we can simplify the 2-RC to an 1-RC thermal circuit (Fig. 3b).

Temperature Equations.

For the 2-RC thermal circuit in Fig. 3a, we can describe the temperatures of B_1 and B_2 as follows:

$$C_1 \cdot \frac{dT^{die}}{dt} + \frac{T^{die} - T^{sp}}{R_1} = P(t) \tag{12}$$

$$C_2 \cdot \frac{dT^{sp}}{dt} + \frac{T^{sp}}{R_2} = \frac{T^{die} - T^{sp}}{R_1} \tag{13}$$

where T^{die} and T^{sp} represent the temperatures at B_1 and B_2 respectively. The power consumption $P(t)$ is the sum of the dynamic and leakage power, which are dependent on the supply voltage V and T^{die}.

If, within a time interval, the power consumption P stays constant, we can solve the differential equations Eq. (12) and Eq. (13) (with given initial temperatures of the node B_1, T_b^{die}, and node B_2, T_b^{sp}). The corresponding solution takes the following form:

$$T^{die}(t) = A_1 * e^{\frac{-t}{tc_1}} + B_1 * e^{\frac{-t}{tc_2}} + C_1 \tag{14}$$

$$T^{sp}(t) = A_2 * e^{\frac{-t}{tc_1}} + B_2 * e^{\frac{-t}{tc_2}} + C_2 \tag{15}$$

where tc_1 and tc_2 are the two time constants of the 2-RC thermal circuit shown in Fig. 3. tc_1 and tc_2 are determined by the thermal resistances R_1, R_2, and thermal capacitances C_1, C_2. Coefficients A_1, B_1, C_1, A_2, B_2, C_2 are determined by the initial temperatures at B_1 T_b^{die} and B_2 T_b^{sp}, power consumption P, thermal resistances R_1, R_2, and thermal capacitances C_1, C_2.

Eq. (14) describes the temperature at B_1 as a function of time, while Eq. (15) describes the one for B_2. Now, let us treat the initial temperatures at B_1, T_b^{die}, and B_2, T_b^{sp}, as variables in Eq. (14) and Eq. (15). With a given length of a time interval, we can rewrite Eq. (14) and Eq. (15) as Eq. (16) and Eq. (17), where T_b^{die} and T_b^{sp} are the temperatures of B_1 and B_2 at the beginning of the time interval, while T_e^{die} and T_e^{sp} are the temperatures at the end of the time interval. a_1, a_2, b_1, b_2, c_1 and c_2 are constant coefficients determined by the length of time interval, and by the values of R_1, R_2, C_1, C_2 and P.

$$T_e^{die} = a_1 \cdot T_b^{die} + b_1 \cdot T_b^{sp} + c_1 \tag{16}$$

$$T_e^{sp} = a_2 \cdot T_b^{die} + b_2 \cdot T_b^{sp} + c_2 \tag{17}$$

Eq. (16) gives the temperature of the chip at the end of the time interval as a functions of the initial temperature, and Eq. (16) gives the one for the heat spreader.

B. SSDTC Estimation

As an input to the SSDTC calculation we have the voltage levels, calculated by the DVS algorithm, and a given idle time distribution, as illustrated in Fig. 4a.

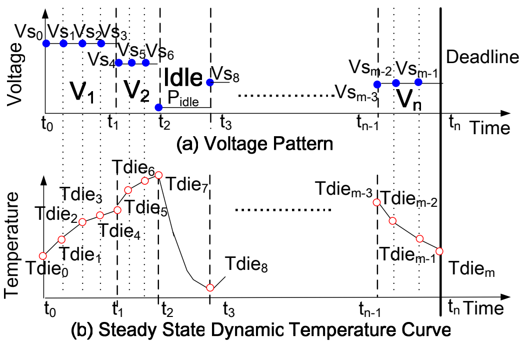


Fig. 4. Temperature Analysis

When the processor is working in the active state, the leakage power varies with the working temperature of the processor. In Fig. 4a, we divide the execution interval of each active state step into several sub-intervals. The total number of sub-intervals is denoted as m. Each sub-interval is short enough such that the temperature variation is small and the leakage power can be treated as constant inside the sub-interval.

P_i is the power consumption for each sub-interval i $(\forall i, 1 \leq i \leq m)$. When the processor is in the active state during the i^{th} sub-interval, P_i is computed by Eq. (18), where Vs_{i-1} and T_{i-1}^{die} are the supply voltage and processor temperature at the start of the i^{th} sub-interval.

$$P_i = P_i^d(Vs_{i-1}) + P_i^{leak}(T_{i-1}^{die}, Vs_{i-1}) \tag{18}$$

$P_i^d(Vs_{i-1})$ represents the dynamic power consumption while $P_i^{leak}(T_{i-1}^{die}, Vs_{i-1})$ represents the leakage power consumption based on the piecewise linear leakage model discussed in Section II-A. When the processor is in idle state during the i^{th} sub-interval, the power consumption $P_i = P_{idle}$.

As shown in Fig. 4b, we construct the SSDTC by calculating the temperature values T_0^{die} to T_m^{die}. The relationship between the start and end temperature of each sub-interval can be described by applying Eq. (16) and Eq. (17) to all sub-intervals. Thus, we can establish a linear system with $2m$ equations as shown by Eq. (19) to Eq. (22). T_i^{die} and T_i^{sp} are the temperature at the beginning of the $i+1^{th}$ sub-interval.

$$T_1^{die} = a_{11} \cdot T_0^{die} + b_{11} \cdot T_0^{sp} + c_{11} \tag{19}$$

$$T_1^{sp} = a_{12} \cdot T_0^{die} + b_{12} \cdot T_0^{sp} + c_{12} \tag{20}$$

$$\dots\dots\dots$$

$$T_m^{die} = a_{m1} \cdot T_{m-1}^{die} + b_{m1} \cdot T_{m-1}^{sp} + c_{m1} \tag{21}$$

$$T_m^{sp} = a_{m2} \cdot T_{m-1}^{die} + b_{m2} \cdot T_{m-1}^{sp} + c_{m2} \tag{22}$$

Due to periodicity, when dynamic steady state is reached, the processor and heat spreader temperature at the beginning of the period should be equal to the temperature at the end of the previous period:

$$T_0^{die} = T_m^{die}; \; T_0^{sp} = T_m^{sp} \tag{23}$$

Solving the above linear system Eq. (19)–Eq. (23), we get the values for T_0^{die} to T_m^{die} and, hence, obtain the corresponding SSDTC. As this system is a tridiagonal linear system, it can be solved efficiently, e.g. through LU decomposition with only $O(m)$ operations [38]. It should be mentioned that, in fact, two SSDTCs are obtained, one reflecting the temperature of the chip, and the other based on that of the heat spreader.

C. Transient Temperature Curve Estimation

The temperature calculated in the previous section (SSDTC) corresponds to the dynamic steady state reached after a sufficient number of iterations have been executed. However, the same technique can be used to calculate any transient temperature curve (TTC), corresponding to an arbitrary time interval, as long as the length of the time interval is significantly smaller than the RC time of the heat sink (which is in the order of minutes). Under this assumption, as discussed earlier in this section, the thermal model in Fig. 3 can be used. The only difference relative to the SSDTC calculation is that Eq. (23) is no longer valid:

$$T_0^{die} \neq T_m^{die}; \; T_0^{sp} \neq T_m^{sp}$$

To estimate the transient temperature curve (TTC), the temperature of T_0^{die} and T_0^{sp} are given as input. The temperature values: T_1^{die},T_2^{die}, …, T_m^{die} and T_1^{sp}, T_2^{sp}, …, T_m^{sp} are calculated by solving equations Eq. (19)–Eq. (22).

VI. ITD with only Static Slack (SITD)

In this section we discuss our solutions to the SITD problem, as formulated in Section IV-A, which only considers static slack. We first introduce our approach ignoring the overheads E_o and t_o in Section VI-A. This approach will be used in Section VI-B where a general SITD technique is presented.

A. SITD without overhead (SITDNOH)

Since, in this section, we ignore the overheads ($E_o = t_o = 0$), it results from Eq. (4) that the cost to be minimized is $\sum_{i=1}^{n} Es_i^{leak}$, which is the total leakage energy consumed during task execution.

Assuming that the execution interval of task τ_i is divided into $q_i - 1$ sub-intervals, the leakage energy consumption of τ_i is the sum of the leakage energy of all sub-intervals:

$$Es_i^{leak} = \sum_{j=1}^{q_i-1} (P_{ij}^{leak}(V_{ij}, \frac{T_{ij}^{die} + T_{i(j+1)}^{die}}{2}) \cdot t_{ij}^{sub}) \tag{24}$$

where T_{ij}^{die}, $T_{i(j+1)}^{die}$ and t_{ij}^{sub} represent the processor SSDTC temperatures at the beginning and end of the j^{th} sub-interval and the length of this sub-interval, respectively. The model in Eq. (1) is used to compute the leakage power, P_{ij}^{leak}, in each sub-interval.

Let us first assume that the chip (as well as the heat spreader) temperature at the termination of each task is known and is independent of the starting temperature of the task. Under this assumption, we can formulate SITDNOH as a convex nonlinear problem shown in Eq. (25)–Eq. (38), where the objective function to be minimized is the total leakage energy for all tasks $\sum_{i=1}^{n} Es_i^{leak}$. The optimization variables to be calculated are the idle slot lengths t_i, ($\forall i, 1 \le i \le n$). Eq. (27) requires the sum of idle slots lengths to be equal with the total available idle time: $dl_n - \sum_{i=1}^{n} te_i^W$. Eq. (28) guarantees that the deadline of each task is satisfied. As mentioned above, the processor and heat spreader temperatures at the end of task τ_i, $T_{iq_i}^{die}$ and $T_{iq_i}^{sp}$, are considered known and assigned by Eq. (29) and Eq. (30), respectively, where Tg_i^{die} and Tg_i^{sp} are given constants. T_{ij}^{die} and $T_{i(j+1)}^{die}$ are the processor temperature at the beginning and end of j^{th} sub-interval in the execution of task τ_i, and are given by Eq. (31) similar to Eq. (19) and Eq. (21) in Section V-B. Eq. (32) describes the same relationship for the heat spreader temperature. $T_{(i+1)1}^{die}$ and $T_{(i+1)1}^{sp}$ are the processor and heat spreader temperatures at the start of task τ_{i+1}, and are dependent on the finishing temperature of the previous task τ_i and the idle slot t_i placed after τ_i. If we assume that all idle slots t_i are significantly shorter than the RC time of the heat spreader, then we can describe the processor temperature behavior during the idle slot i by Eq. (33) and Eq. (35), based on the 1-RC thermal circuit described in Section V-A. TIs is the steady state temperature that the processor would reach if P_{idle} would be consumed for a sufficiently long time and is calculated according to Eq. (37). R_g is the sum of the two thermal resistances R_1 and R_2 in Fig. 3b. Under the same assumption as above, the heat spreader temperature stays constant during the idle slot as shown in Eq. (34) and Eq. (36)[2]. Eq. (33) and Eq. (34) calculate the processor and heat spreader temperature at the end of the idle slot following task τ_i and, implicitly, the starting temperature of τ_{i+1}. Eq. (35) and Eq. (36) compute the temperature at the start of task τ_1, taking into consideration that this task starts after the idle period following task τ_n (the task set is executed periodically). Finally, the constraint in Eq. (38) requires that the processor temperatures during execution of the task set, T_{ij}^{die} ($\forall i, 1 \le i \le n, \forall j, 1 \le j \le q_i$), do not exceed the maximal allowable working temperature of the chip T_{max}. The presented formulation is a convex non-linear problem, and can be solved efficiently in polynomial time [39].

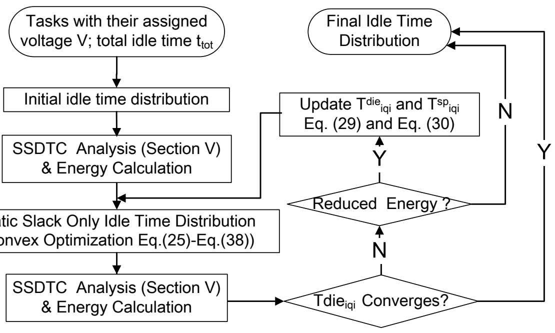
[2]Idle periods are supposed to be short. If, exceptionally, they are not significantly shorter than the heat spreader RC time, we use the 2-RC circuit to model the temperature during the idle period in Eq. (33)–Eq. (36). This will not affect the convexity of the formulation.

Formulation 1 SITD with No Overheads Consideration

$$\begin{aligned}
&\text{Minimize :} \\
\sum_{i=1}^{n} Es_i^{leak} &= \sum_{i=1}^{n} (\sum_{j=1}^{q_i-1} (t_{ij}^{sub} \cdot P_{ij}^{leak})) && (25)\\
&\text{Subject to :} \\
t_i &\ge 0 \, (\forall i, 1 \le i \le n) && (26)\\
\sum_{i=1}^{n} t_i &= dl_n - \sum_{i=1}^{n} te_i^W && (27)\\
dl_i &\ge \sum_{j=1}^{i-1} t_j + \sum_{j=1}^{i} te_j^W \, (\forall i, 1 \le i \le n) && (28)\\
T_{iq_i}^{die} &= Tg_i^{die} \, (\forall i, 1 \le i \le n) && (29)\\
T_{iq_i}^{sp} &= Tg_i^{sp} \, (\forall i, 1 \le i \le n) && (30)\\
T_{i(j+1)}^{die} &= a1_{ij} \cdot T_{ij}^{die} + b1_{ij} \cdot T_{ij}^{sp} + c1_{ij} && (31)\\
T_{i(j+1)}^{sp} &= a2_{ij} \cdot T_{ij}^{die} + b2_{ij} \cdot T_{ij}^{sp} + c2_{ij} && (32)\\
&\quad (\forall i, 1 \le i \le n; \forall j, 1 \le j \le q_i - 2) \\
T_{(i+1)1}^{die} &\ge TIs + (T_{iq_i}^{die} - TIs) \cdot e^{(\frac{-t_i}{R_g \cdot C_1})} && (33)\\
&\quad (\forall i, 1 \le i \le n-1) \\
T_{(i+1)1}^{sp} &= T_{iq_i}^{sp} \, (\forall i, 1 \le i \le n-1) && (34)\\
T_{11}^{die} &\ge TIs + (T_{nq_n}^{die} - TIs) \cdot e^{(\frac{-t_n}{R_g \cdot C_1})} && (35)\\
T_{11}^{sp} &= T_{nq_n}^{sp} && (36)\\
TIs &= P_{idle} \cdot R_g && (37)\\
T_{ij}^{die} &\le T_{max} && (38)\\
&\quad (\forall i, 1 \le i \le n; \forall j, 1 \le j \le q_i)
\end{aligned}$$

SITDNOH Algorithm.
The above formulation is based on the particular assumption that the temperature at the end of a task τ_i is known and fixed. However, in reality, this is not the case, and the temperature $T_{iq_i}^{die}$ and $T_{iq_i}^{sp}$ (Eq. (29) and Eq. (30)) at the termination of a task depend on the starting temperature of the task and, implicitly, on the distribution of the idle time. This makes the above formulation become a non-convex programming problem which is very time consuming to solve. In order to solve the problem efficiently we have developed an iterative heuristic outlined in Fig. 5.

The heuristic starts with an arbitrary initial ITD, for example, that the entire idle time t_{tot} is placed after the last task τ_n. Assuming this ITD and the given voltage levels, steady state dynamic temperature analysis is performed, as described in Section V-B. Given the obtained SSDTC, the leakage energy consumption $\sum_{i=1}^{n} E_i^{leak}$ corresponding to the assumed ITD is calculated. From the SSDTC we can also extract the final temperatures $T_{iq_i}^{die}$ and $T_{iq_i}^{sp}$ for each task τ_i. Assuming this $T_{iq_i}^{die}$



Tasks with their assigned voltage V; total idle time t_{tot}
Initial idle time distribution
SSDTC Analysis (Section V) & Energy Calculation
Static Slack Only Idle Time Distribution (Convex Optimization Eq.(25)-Eq.(38))
SSDTC Analysis (Section V) & Energy Calculation
Final Idle Time Distribution
Update T^{die}_{iqi} and T^{sp}_{iqi} Eq. (29) and Eq. (30)
Reduced Energy ?
$Tdie_{iqi}$ Converges?
Y
N
N
Y

Fig. 5. SITDNOH Heuristics

and $T^{sp}_{iq_i}$ as the final temperatures in Eq. (29) and Eq. (30), we can calculate the idle time t_i using the convex optimization formulated in Eq. (25)–Eq. (37).

From the new ITD resulted after the optimization, we calculate a new SSDTC which provides new temperatures $T^{die}_{iq_i}$ and $T^{sp}_{iq_i}$ at the end of each task τ_i. The new total leakage energy consumption $\sum_{i=1}^{n} E_i^{leak}$, corresponding to the updated ITD, is also calculated. The process is continued assuming the new end temperatures in Eq. (29) and Eq. (30) and the convex optimization produces a new ITD.

The iterations stop when the temperature $T^{die}_{iq_i}$ converges (i.e. $|T^{die^{new}}_{iq_i} - T^{die^{old}}_{iq_i}| < \varepsilon, \forall i, 1 \le i \le n$). However, it can happen that, after a certain point, additional iterations do not significantly improve the ITD. Therefore, even if convergence has not yet been reached, the optimization is stopped if no significant energy reduction has been achieved: $(E^{old}_{tot} - E^{new}_{tot})/E^{old}_{tot} < \varepsilon'$. Our experiments have shown that maximum 5 iterations are needed with $\varepsilon = 0.5°$ and $\varepsilon' = 0.001$.

B. SITD with overhead (SITDOH)

The approach presented in Section VI-A is based on the assumption that time and energy overheads for switching the processor to and from the idle state, t_o and E_o, are zero, which is not the case in reality. If we consider the hypothetical case that the end temperature of each task is known, the problem can be formulated similar to Eq. (25)–Eq. (37), with the main difference that the total energy to be minimized is given in Eq. (4). Based on this formulation, we could solve the SITDOH problem for the real case, when the end temperatures are not supposed to be known, similarly to the approach described in Fig. 5. However, the formulation with the objective function Eq. (4), due to the binary variable x_i, is a mixed integer convex programing problem which is very time consuming to solve. We, hence, propose an SITDOH heuristic based on the SITDNOH approach presented in Section VI-A.

Our SITDOH heuristic comprises two steps. In the first step an optimization of the idle time distribution is performed by eliminating idle intervals whose lengths are smaller than a certain threshold limit. In the second step, the ITD is further refined in order to improve energy efficiency.

A lower bound t^{min} on the length t_i of an idle slot can be determined by considering the following two bounds:

1) No idle slot is allowed to be shorter than t_o, the total time needed to switch to/from the idle state.
2) The energy overhead due to switching should be compensated by the gain due to putting the processor into the idle state. The energy gain for an idle interval t_i is computed as:

$$E_g = \int_0^{t_i} P_i^{leak}(V_i, T_i(t))\,\mathrm{d}t - P_{idle} \cdot t_i \tag{39}$$

where $T_i(t)$ is the processor temperature as a function of time during the idle time interval [0, t_i]. V_i is the supply voltage for τ_i. $P_i^{leak}(V_i, T_i(t))$ is the leakage power in the active state during the idle time interval. Thus, in order for the overhead to be compensated, we need $E_o < E_g$. As P_i^{leak} depends on the temperature, the threshold length of an idle slot is not a given constant. Nevertheless, this length will be always larger than $E_o/(P^{leak}_{max_i} - P_{idle})$, where $P^{leak}_{max_i} = P_i^{leak}(V_i, T_{max})$ is the leakage power at the maximum temperature T_{max} at which the processor is allowed to run.

In conclusion, for the first step of the SITDOH heuristic (Fig. 6a), we consider: $t^{min} = max(t_o, E_o/(P^{leak}_{max_i} - P_{idle}))$.

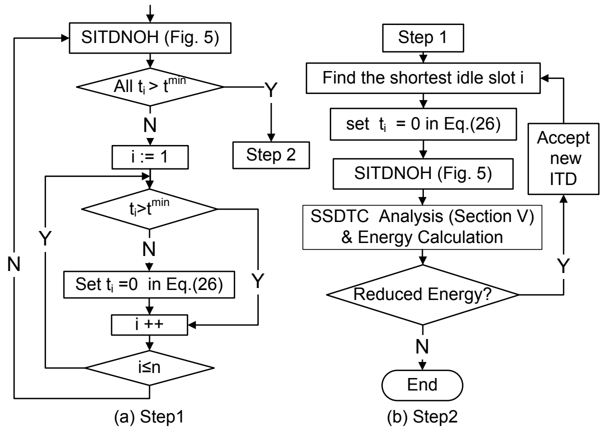


Fig. 6. SITDOH Heuristics

The basic idea of the first step is that no idle slot is allowed to be shorter than t^{min}. Thus, after running SITDNOH, the obtained ITD is checked slot by slot. If a slot length t_i is smaller than t^{min}, this slot will be removed. In order to achieve this, the particular constraint in Eq. (26), corresponding to slot i, is changed from $t_i \ge 0$ to $t_i = 0$. After all slots have been visited and Eq. (26) updated, SITDNOH is performed again. The obtained ITD is such that all slots which in the previous iteration have been found shorter than t^{min} have disappeared and the corresponding idle time has been redistributed among other tasks. The process is repeated until no slot shorter than t^{min} has been identified.

After step1, we still can be left with slots that are too short to be energy efficient. There are two reasons for this:

1) Due to the fact that the processor is running at a temperature lower than the maximum allowed T_{max}, it can happen that the real t^{min} is smaller than the one considered in step1.
2) Even if $E_o < E_g$, which means that an energy reduction due to the idle slot is obtained, energy efficiency can, possibly, be improved by eliminating the slot and distributing the corresponding idle time among other slots.

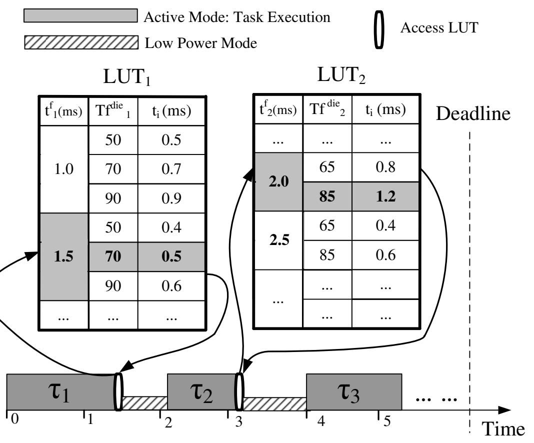
In the second step (Fig. 6b), we start from the shortest idle slot and consider to eliminate it (by setting the corresponding constraint $t_i = 0$ in Eq. (26)). If the ITD obtained after applying SITDNOH is more energy efficient, the new ITD is accepted. The process is continued as long as, by eliminating a slot, the total energy consumption is reduced.

As mentioned earlier, SITDNOH, which is at the heart of SITDOH, can be run efficiently in polynomial time. For each of the two steps implied by SITDOH, the SITDNOH algorithm is executed maximally n times, where n is the total number of tasks.

VII. ITD WITH DYNAMIC AND STATIC SLACK (DITD)

The above SITD approach determines idle time settings assuming that tasks always execute their WNC. However, due to execution time variations, large amounts of dynamic slack are created at run-time. In order to exploit the dynamic slack, the slot length t_i has to be determined at run-time based on the values of the current time and temperature after termination of task τ_i. In principle, calculating the appropriate t_i implies the execution of a temperature aware ITD algorithm similar to the one described in Section VI-B (with the objective function and constraints in Eq. (8)–Eq. (10)). Running this algorithm on-line, after execution of each task, implies a time and energy overhead which is not acceptable.

To overcome the above problem, we have divided our DITD approach into an off-line and an on-line phase. In the off-line phase, idle time settings for all tasks are pre-computed, based on possible finishing times and finishing temperatures of the task. The results are stored in look-up tables (LUTs), one for each task. In Fig. 7, we show two such tables. They contain idle time settings for combinations of possible termination times t_i^f and finishing temperatures Tf_i^{die} of a task τ_i.



LUT$_1$

t^f_1(ms)	Tf^{die}_1	t_i (ms)
1.0	50	0.5
	70	0.7
	90	0.9
1.5	50	0.4
	70	**0.5**
	90	0.6
...	...	...

LUT$_2$

t^f_2(ms)	Tf^{die}_2	t_i (ms)
...	...	...
2.0	65	0.8
	85	**1.2**
2.5	65	0.4
	85	0.6
...	...	...
	...	...

Fig. 7. DITD On-line Phase

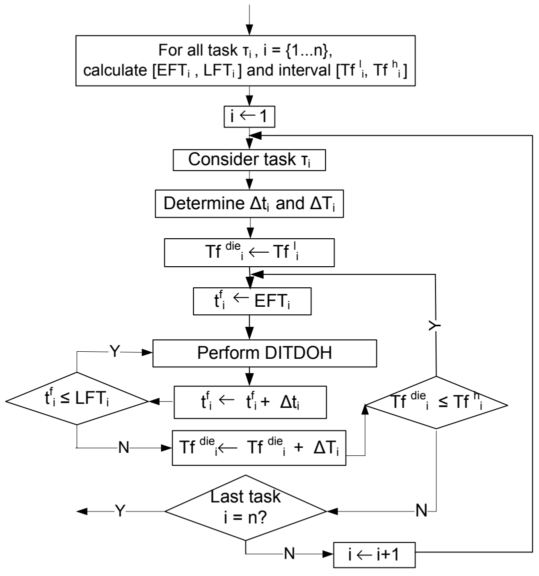
A. On-line Phase

The on-line phase is illustrated in Fig. 7. Each time a task τ_i terminates, the length of the idle time slot t_i following the termination of τ_i has to be fixed; the on-line scheme chooses the appropriate setting from the lookup table LUT_i, depending on the actual time and temperature sensor reading. If there is no exact entry in LUT$_i$, corresponding to the actual time/temperature, the entry corresponding to the immediately higher time and closest temperature value is selected. For example, in Fig. 7, τ_1 finishes at time 1.35ms with a temperature $78°C$. To determine the appropriate idle time slot length t_1, LUT_1 is accessed. As there is no exact entry with t_1^f=1.35ms and Tf_1^{die}= $78°C$, the entry corresponding to termination time 1.5ms (1.5ms is immediately higher than 1.35ms) and temperature 70°C (as it is the closest one to $Tf^{die} = 78$°C) is chosen. Hence, the processor will be switched to the idle state for 0.5ms before the next task, τ_2, starts. This on-line phase is of very low, constant time complexity O(1) and, thus, very efficient.

We should notice that, according to our temperature model presented in Section V, the state of the system is defined by both the die and the heat spreader temperatures. In our LUTs, however, we only consider the die temperature for taking the decision on the idle slack. This is due to the following reasons:

1) It is both impractical and potentially expensive to obtain, at run-time, temperature readings from the heat spreader.
2) The variations of the heat spreader temperature are small compared to those of the chip. This is due to the fact that the heat capacitance of the heat spreader is much larger than that of the chip.
3) Considering also the heat spreader temperature as an additional dimension in the LUTs would dramatically increase the size of the tables without significant contribution to energy efficiency.

Thus, when generating the LUTs, we will consider that, at the termination of a task τ_i, the heat spreader has a certain expected temperature Tf_i^{sp}. In Section VII-E we will show how Tf_i^{sp} is calculated.



For all task τ_i, i = {1...n}, calculate [EFT$_i$, LFT$_i$] and interval [Tf^l_i, Tf^h_i]
$i \leftarrow 1$
Consider task τ_i
Determine Δt_i and ΔT_i
$Tf^{die}_i \leftarrow Tf^l_i$
$t^f_i \leftarrow EFT_i$
Perform DITDOH
$t^f_i \le LFT_i$
$t^f_i \leftarrow t^f_i + \Delta t_i$
$Tf^{die}_i \leftarrow Tf^{die}_i + \Delta T_i$
$Tf^{die}_i \le Tf^h_i$
Last task i = n?
$i \leftarrow i+1$
Y
N
Y
Y
N
N
N

Fig. 8. DITD Off-line Phase

B. Off-line Phase

In the off-line phase, one LUT table is generated for each task. The LUT table generation algorithm is illustrated in Fig. 8. The outermost loop iterates over the set of tasks and successively constructs the table LUT_i for each task τ_i. The next loop generates LUT_i entries corresponding to the various possible finishing temperatures Tf_i^{die} of τ_i. Finally, the innermost loop iterates, for each possible finishing temperature, over all considered termination times t_i^f of task τ_i.

The algorithm starts by computing the earliest EFT_i and latest possible finishing times LFT_i, as well as the lowest Tf_i^l and highest possible finishing temperature Tf_i^h for each task τ_i. With a given finishing time t_i^f and finishing temperature Tf_i^{die} of task τ_i, the innermost loop performs the slack distribution step DITDOH, iteratively. We describe the DITDOH algorithm in Section VII-C. For successive iterations, the finishing temperature Tf_i^{die} and time t_i^f will be increased with the time and temperature quanta $\triangle t_i$ and $\triangle T_i$, respectively. The calculation of the parameters EFT_i, LFT_i, Tf_i^l and Tf_i^h as well as the determination of the granularities and number of entries along the time and temperature dimensions are presented in Section VII-D and Section VII-E, respectively.

C. DITDOH Algorithm

When calculating the actual LUT entries for a task τ_i, the ITD algorithm DITDOH is performed to determine the idle slot length t_i following the termination of τ_i, with given termination time and temperature, based on the problem formulation described in Section IV-B. DITDOH is similar to SITDOH outlined in Section VI-B. However, unlike the formulation used in SITDOH (Eq. (25)–Eq. (38)) which is based on SSDTC estimation, the formulation used for DITDOH is based on the estimation of a transient temperature curve (TTC) described in Section V-C. Since we do not rely on the fact that successive iterations of the application are identical and that tasks execute always with their worst case number of cycles, we do not calculate an SSDTC corresponding to the dynamic steady state. But, instead, we estimate a TTC.

The formulation used for DITDOH is shown in Eq. (40)–Eq. (55). As mentioned in Section IV-B, the energy is optimized

Formulation 2 DITD with No Overheads Consideration

$$\begin{aligned}
&\text{Minimize :}\\
\sum_{k=i+1}^{n} E_k^{leak} &= \sum_{k=i+1}^{n}\Big(\sum_{j=1}^{q_k-1}(t_{kj}^{sub}\cdot P_{kj}^{leak})\Big) && (40)\\
&\text{Subject to :}\\
t_j &\geq 0\,(\forall j, i\leq j\leq n) && (41)\\
\sum_{k=i}^{n} t_k &= dl_n - t_i^f - \sum_{k=i+1}^{n} te_j^E && (42)\\
dl_{i+1} &\geq t_i^f + t_i + te_{i+1}^W && (43)\\
LFT_{i+1} &\geq t_i^f + t_i + te_{i+1}^W && (44)\\
dl_j &\geq t_i^f + \sum_{k=i}^{j-1} t_k + te_{i+1}^W + \sum_{k=i+2}^{j} te_k^E && (45)\\
&\quad(\forall j, i+2\leq j\leq n)\\
T_{kq_k}^{die} &= Tg_k^{die}\,(\forall k, i+1\leq k\leq n) && (46)\\
T_{kq_k}^{sp} &= Tg_k^{sp}\,(\forall k, i+1\leq k\leq n) && (47)\\
T_{(i+1)1}^{die} &\geq TIs + (Tf_i^{die} - TIs)\cdot e^{(\frac{-t_i}{R_g\cdot C_1})} && (48)\\
T_{(i+1)1}^{sp} &= Tf_i^{sp} && (49)\\
T_{k(j+1)}^{die} &= a1_{kj}\cdot T_{kj}^{die} + b1_{kj}\cdot T_{kj}^{sp} + c1_{kj} && (50)\\
T_{k(j+1)}^{sp} &= a2_{kj}\cdot T_{kj}^{die} + b2_{kj}\cdot T_{kj}^{sp} + c2_{kj} && (51)\\
&\quad(\forall k, i<k\leq n;\forall j, 1\leq j\leq q_k-2)\\
T_{(k+1)1}^{die} &\geq TIs + (T_{kq_k}^{die} - TIs)\cdot e^{(\frac{-t_k}{R_g\cdot C_1})} && (52)\\
&\quad(\forall k, i+1\leq k\leq n-1)\\
T_{(k+1)1}^{sp} &= T_{kq_i}^{sp},\,(\forall k, i\leq k\leq n-1) && (53)\\
TIs &= P_{idle}\cdot R_g && (54)\\
T_{kj}^{die} &\leq T_{max} && (55)\\
&\quad(\forall k, i+1\leq k\leq n;\forall j, 1\leq j\leq q_k)
\end{aligned}$$

for the case that the future tasks τ_{i+1} to τ_n execute their expected time te^E which, in reality, happens with a much higher probability than, e.g. the te^W (nevertheless, idle time slots are distributed such that, even in the worst case, deadlines are satisfied). The objective function Eq. (40) to be minimized is the total leakage energy of further tasks to be executed in the current iteration: τ_k, $(\forall k, i<k\leq n)$. Eq. (40) is similar to Eq. (25) with two differences:

1) It refers only to the remaining tasks $\tau_{i+1}, \ldots, \tau_n$.
2) The execution interval of a task τ_k, which is divided into $q_k - 1$ subintervals, is not corresponding to the worst case te_k^W, but to the expected case te_k^E.

The optimization variables to be calculated are the idle slot lengths t_k, $(\forall k, i\leq k\leq n)$. Eq. (42) requires that the sum of all idle slot lengths should be equal to the total available idle time, where t_i^f is the current task's finishing time. The total available idle time is calculated based on the assumption that all future tasks are executed with their expected workload.

Eq. (43) guarantees the deadline of task τ_{i+1}—the next task to be executed after the termination of the current task τ_i in the worst case (task τ_{i+1} executed with te_{i+1}^W). In order to guarantee that all future tasks meet their deadlines in the worst case, Eq. (44) requires that τ_{i+1} finishes before LFT_{i+1}, in the worst case. The latest finishing time LFT_{i+1} (see Section VII-D) is the latest termination time of task τ_{i+1} that still allows future tasks, following τ_{i+1}, to satisfy their deadline even if their worst case workloads are executed. Thus, Eq. (43) and Eq. (44) guarantee not only that the deadline of τ_{i+1} is satisfied in the worst case but also that τ_{i+1} finishes in time for all the remaining tasks to be able to meet their deadline in the worst case. Eq. (45) enforces the deadline of the remaining tasks τ_j, $(\forall j, i+2\leq j\leq n)$, considering that they execute their expected workload. This means that the idle time t_i following task τ_i is determined such that it guarantees deadlines to be satisfied in the worst case but is optimized for the situation that tasks execute their expected workloads.

Similar to Eq. (29) and Eq. (30), Eq. (46) and Eq. (47) specify the processor and heat spreader temperatures at the finishing of task τ_k: $T_{kq_k}^{die}$ and $T_{kq_k}^{sp}$. Eq. (48) computes the processor temperature at the beginning of task τ_{i+1} similar to Eq. (35), where Tf_i^{die} is the chip temperature at the termination of the current task τ_i. Similarly, Eq. (49) computes the heat spreader temperature at the beginning of task τ_{i+1}, where Tf_i^{sp} is, as described in Section VII-A, the expected heat spreader temperature at the termination of task τ_i. Tf_i^{sp} is pre-calculated as will be explained in Section VII-E. Eq. (50)–Eq. (53) compute the TTC of processor/heat spreader based on our TTC estimation method described in Section V-C, where T_{kj}^{die} and $T_{k(j+1)}^{die}$ are the processor temperature at the beginning and end of the j^{th} sub-interval during the execution of task τ_k. Finally, throughout the execution of the future tasks τ_j $(\forall j, i+1\leq j\leq n)$, the processor temperatures T_{kj}^{die} $(\forall k, i+1\leq k\leq n;\forall j, 1\leq j\leq q_k)$ should not exceed the maximal allowable working of the chip T_{max} as imposed by the constraint in Eq. (55). The above formulation is a convex non-linear problem and can be solved efficiently in polynomial time [39].

Coming back to the DITD off-line phase in Fig. 8, the DITDOH algorithm is invoked for each line in the LUT_i corresponding to a task τ_i. This invocation will result in the calculation of the slack length t_i corresponding to the current value of termination time t_i^f and temperature Tf_i^{die}. DITDOH performs exactly like SITDOH (Fig. 6), with the exception that for solving SITDNOH (Fig. 5), instead of Eq. (25)–Eq. (38), the formulation in Eq. (40)–Eq. (55) is used.

D. Time Bounds and Granularity

In the first step of the algorithm in Fig. 8, the EFT_i and LFT_i for each task are calculated. The earliest finishing time EFT_i

is calculated based on the situation that all tasks execute their best case execution time te_i^B. The latest finishing time LFT_i is calculated as the latest termination time of τ_i that still allows all tasks τ_j, $j > i$, to satisfy their deadlines when they execute their worst case execution time te_i^W.

With the time interval $[EFT_i, LFT_i]$ for task τ_i, a straightforward approach to determine the number of entries along the time dimension would be to allocate the same number of entries for each task. However, the time interval sizes $LFT_i - EFT_i$ can differ very much among tasks, which should be taken into consideration when deciding on the number of time entries Nt_i. Therefore, given a total number of entries along the time dimension NL_t, we determine the number of time entries in each LUT_i, as follows:

$$Nt_i = \left\lceil NL_t \cdot \frac{(LFT_i - EFT_i)}{\sum\limits_{i=1}^{n}(LFT_i - EFT_i)} \right\rceil$$

The corresponding granularity along the time dimension $\triangle t_i$ is the same for all tasks and is obtained as follows:

$$\triangle t_i = \frac{\sum\limits_{i=1}^{n}(LFT_i - EFT_i)}{NL_t}, \quad (\forall i, 1 \leq i \leq n)$$

E. Temperature Bounds and Granularity

The granularity $\triangle T_i$ along the temperature dimension is the same for all task τ_i and has been determined experimentally. Our experiments have shown that values around 15° are appropriate, in the sense that finer granularities will only marginally improve energy efficiency.

To determine the number of entries along the temperature dimension, we need to calculate the temperature interval $[Tf_i^l, Tf_i^h]$ at the termination of each task. In fact, it is not needed to determine the bounds of the temperature interval exactly. A good estimation, such that, at run-time, temperature readings outside the determined interval will happen rarely, is sufficient. If the temperature readings exceed the upper/lower bound of the interval, the idle time setting corresponding to the highest/lowest temperature value available in the LUT will be used. One alternative would be to simply assume that all tasks have a finishing temperature interval $[T_a, T_{max}]$, where T_a is the ambient temperature and T_{max} is the maximum temperature at which the chip is allowed to work. This would lead to huge amounts of wasted memory space (for storing LUT tables) as well as wasted computation time in the off-line phase. We have developed an estimation technique for the temperature interval $[Tf_i^l, Tf_i^h]$, which balances computation complexity and accuracy of the results.

In order to estimate the temperature bounds Tf_i^l and Tf_i^h, we define two run-time scenarios:

- *Worst case execution scenario*: in which the actual execution time of each task τ_i is always equal to its worst case execution time: $te_i^A = te_i^W$.
- *Best case execution scenario*: in which the actual execution time of each task τ_i is always equal to its best case execution time: $te_i^A = te_i^B$.

In both scenarios, the processor will execute the corresponding periodic power pattern repeatedly and the processor temperature will eventually reach the corresponding steady state dynamic temperature curve (denoted as $SSDTC^w$ for the worst case scenario and $SSDTC^b$ for the best case scenario, respectively). From the corresponding SSDTC, we can obtain, for each task τ_i, its finishing temperature. We use the finishing temperature of task τ_i corresponding to the *worst case execution scenario*, Tf_i^w, as the upper bound of the finishing temperature of task τ_i: $Tf_i^h = Tf_i^w$; the finishing temperature of task τ_i corresponding to the *best case execution scenario*, Tf_i^b, will be used as the lower bound: $Tf_i^l = Tf_i^b$.

In order to obtain the $SSDTC^w$ we first perform the SITDOH heuristic (Fig. 6). Then, temperature analysis (Section V) produces the temperature curve for the worst case scenario with the corresponding idle time distribution generated by SITDOH. The $SSDTC^b$ curve is obtained in a similar way, by replacing te_i^W with te_i^B in the constraint in Eq. (27).

With the upper and lower bounds Tf_i^l and Tf_i^h obtained for each task, the number of the entries along the temperature dimension, for task τ_i, is:

$$NT_i = \left\lceil \frac{Tf_i^h - Tf_i^l}{\triangle T_i} \right\rceil$$

where $\triangle T_i$ is the granularity along the temperature dimension.

As mentioned in Section VII-A, when generating the LUTs, we consider that, at the termination of a task τ_i, the heat spreader has a certain expected temperature Tf_i^{sp}. In order to obtain these temperatures, we perform the same procedure as outlined above but, in this case, considering the expected execution time of each task: $te_i^A = Te_i^E$. We obtain the temperature curve $SSDTC_{sp}^{exp}$ corresponding to the heat spreader (see Section V-B), from which we extract the expected temperature of the heat spreader, Tf_i^{sp}, at the termination of each task τ_i.

VIII. Experimental Results

A. Evaluation of The Thermal Model

Experimental Setup.
We have evaluated our thermal model considering platforms with parameter settings based on values from [40], [41] and [42]. We consider die areas of 6×6, 8×8, and 10×10mm^2. The heat spreader area is five times the die area, and the heat sink area is between 1.3 and 1.4 times the area of the heat spreader. The thickness of the die and heat spreader are 0.5mm and 2mm respectively. The thickness of the heat sink is between 10mm and 20mm. The coefficients corresponding to the power model in Section II-A are based on [34] and [3]. For the temperature calculation (Section V-B and Section V-C) we have considered a piecewise linear leakage model with 3 segments, as recommended in [36].
Accuracy.
We first performed a set of experiments to evaluate the accuracy of our temperature analysis approach proposed in Section V. We randomly generated 500 periodic voltage patterns corresponding to applications with periods in the range between 5ms and 100ms. For each application, considering the coefficients and platform parameters outlined above, we have computed the SSDTC using the approach proposed in Section V-B and by using Hotspot simulation. For each pair of temperature curves obtained, we calculated the maximum deviation as the largest temperature difference between any corresponding pairs of points (in absolute value), as well as the average deviation. Fig. 9 illustrates the results for different application periods. For applications with a period of 50ms, for example, there is no single case with a maximum deviation larger than 2.1°C, and the average deviation is 0.8°C. Over all 500 applications, the average and maximum deviation are 0.8°C and 3.8°C, respectively. We can observe that the deviation increases with the increasing period of the application. This

is due to the fact that, with larger periods, accuracy can be slightly affected by neglecting the thermal capacitance of the heat sink (see Section V-A).

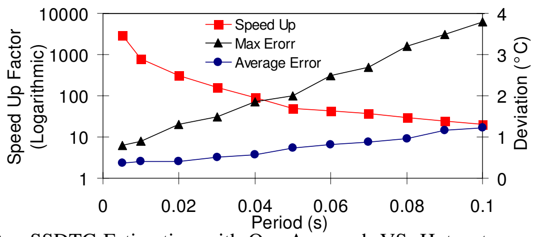


Fig. 9. SSDTC Estimation with Our Approach VS. Hotspot

Computation Time.
We have compared the corresponding computation time of our SSDTC generation approach with the time needed by Hotspot. Fig. 9 illustrates the average speedup as the ratio of the two execution times. The speedup is between 3000 for periods of 5ms and 20 for 100ms periods. An increasing period leads to a larger linear system that has to be solved for SSDTC estimation (Section V-B), which explains the shape of the speedup curve in Fig. 9.

The accuracy and speedup of our approach are also dependent on the length of the sub-interval considered for the temperature analysis (Section V-B and Fig. 4). For the experiments throughout this paper, the length of the sub-interval is 2ms. This is based on the observation that reducing the length beyond this limit does not improve the accuracy significantly.

B. Evaluation of The ITD Heuristics

We have used both generated test applications as well as a real life example in our experiments to evaluate our DITD approach presented in Section VII. This, implicitly, also evaluates the SITD approach (Section VI) since:

1) For small dynamic slack ratio, the dynamic approach converges towards the static one.
2) The DITD approach is based on the SITD for calculation of each entry in the LUTs.

Experimental Setup.
We have randomly generated 100 test applications consisting of 30 to 100 tasks. The workload in the worst case (WNC) for each task is generated randomly in the range [10^6, $5.0{\times}10^6$] clock cycles, while the workload in the best case is generated in the range [10^5, $5.0{\times}10^5$] clock cycles. To generate the expected workload ENC_i of each task, the following steps are performed:

1) The value of the expected total dynamic idle time, t_d^E, is given as an input: t_d^E is the total dynamic slack when all tasks execute their workload in the expected case: $t_d^E = \sum_{i=1}^{n}(te_i^W - te_i^E)$.
2) t_d^E is divided into a number n_{sub} of sub-intervals with equal length (t_{sub}).
3) The n_{sub} sub-intervals are allocated among all tasks based on a uniform distribution; as result, each task is allocated a number p of sub-intervals.
4) The expected workload ENC_i of task τ_i is, thus, determined as: $ENC_i = WNC_i - p \cdot t_{sub} \cdot f_i$, where f_i is the processor frequency when task τ_i is executed.

In order to evaluate our DITD technique, we have considered a straightforward approach (SFA) for comparison. This SFA scenario corresponds to the natural execution procedure for the case when no idle time distribution is performed. Following this approach, tasks are executed according to a static schedule generated based on the worst case execution time. According to this schedule, the static slack is placed at the end of the application, after the last task. At run-time, when the tasks execute less than their WNC and the generated dynamic slack is large enough, the processor is put in idle mode. More exactly, the SFA works as follows:

1) The start time of each task t_i^{st} is determined off-line by: $t_i^{st} = te_{i-1}^W + t_{i-1}^{st}$.
2) At runtime, whenever a task τ_i terminates, we compute the gap $t_g = t_{i+1}^{st} - t_i^f$, where t_i^f is the termination time of the current task.
3) If $t_g = 0$, the next task τ_{i+1} starts immediately after the termination of task τ_i. When $t_g > 0$, if the following two conditions are both satisfied, the processor will be switched to idle state during t_g (otherwise the processor will stay in the active state with the voltage level at which task τ_i is executed): (a) $t_g > t_o$, where t_o is the time overhead due to state switching; (b) the energy gain E_g is positive: $E_g = E^a - (P_{idle} \cdot t_g + E_o) > 0$, where E^a is the leakage energy consumption of the processor during t_g, if the processor stays in the active state. E^a is estimated as $P^{leak} \cdot t_g$, where P^{leak} is the leakage power consumption calculated at the temperature when task τ_i terminates. $P_{idle} \cdot t_g + E_o$ is the energy consumption if the processor is switched to idle state during t_g, where E_o is the energy overhead due to switching, and P_{idle} is the power consumption in idle state.

We have applied both the DITD and SFA approaches on the same test application. We assume, for each task τ_i, that the actual executed workload at run-time conforms to a beta distribution [38]. When we simulate the execution of the test applications, the actual number of executed clock cycles of a task is generated using a random number generator according to the beta distribution Beta($\alpha 1_i, \alpha 2_i$). The parameters $\alpha 1_i$ and $\alpha 2_i$ are determined based on (1) the expected workload ENC_i and (2) a given standard deviation σ_i of the executed clock cycles of task τ_i. The Hotspot system [11] is used to simulate the sensor readings which track the temperature behaviour of the platform during the execution of a test application.

In our experiments, the granularity along the time and temperature dimensions for the LUT tables is set to 1.5–2.0ms and 15°–20°, respectively. It is important to mention that in all our experiments we have accounted for the time and energy overhead imposed by the on-line phase of our DITD. Similarly, we have also taken into consideration the energy overhead due to the memory access. This overhead has been calculated based on the energy values given in [43] and [44]. The energy and time overheads due to power state switching are set to $E_o = 0.5$mJ and $t_o = 0.4$ms, respectively, according to [9].

After applying both the DITD and SFA approaches on a test application, we compute the corresponding leakage energy reduction due to our DITD approach compared to the SFA: $I = (E^{SFA} - E^{DITD})/E^{SFA} \cdot 100\%$, where E^{SFA} and E^{DITD} are the consumed leakage energy corresponding to the SFA and DITD approach, respectively.

Leakage Energy Reduction vs. Slack Time Ratios.
We first performed experiments considering different combinations of static (r_s) and dynamic idle time ratio (r_d). The static idle time ratio is computed as: $r_s = (dl_n - \sum_{i=1}^{n} te_i^W)/dl_n$, where dl_n is the deadline of the last task in execution order. The dynamic idle time ratio is calculated as: $r_d = t_d^E/dl$, where t_d^E is the total dynamic slack when all tasks execute their workload in the expected case, as described earlier in this section. Fig. 10 shows the averaged leakage energy reduction I over all test

applications. The energy reduction achieved by DITD grows with the available amount of static and dynamic slack. With $r_s = 0.2$ and $r_d = 0.2$, for example, leakage energy can be reduced with 20% by applying our $DITD$ approach.

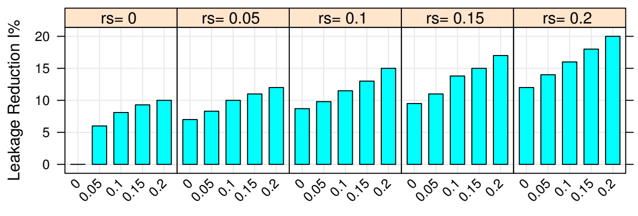


Fig. 10. Leakage Energy Reduction with Low Switching Overheads

In order to explore the influence of the energy and time overheads on the potential leakage reduction, we have repeated the previous experiments in a context where energy and time overheads are set to higher values: $E_o = 1.0$mJ and $t_o = 0.8$ms. Fig. 11 shows the corresponding averaged leakage energy reduction I. The results show a similar trend as in Fig. 10. Comparing the results in Fig. 10 and Fig. 11, we can observe that the leakage reduction achieved with the higher overhead settings is larger. The leakage reduction approaches 40% with $r_s = 0.2$ and $r_d = 0.2$. The reason is the following: with

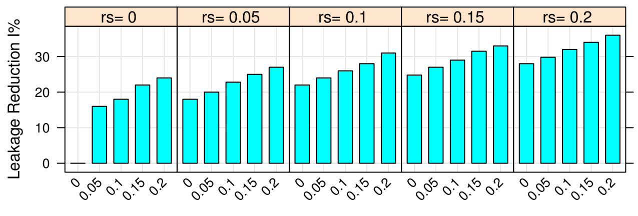


Fig. 11. Leakage Energy Reduction with High Switching Overheads

large switching overheads, it is more likely that the generated slots are too small for switching power state to be energy efficient. Thus, using the SFA approach, the processor will keep in the active power state. With the DITD approach, however, the slack time will be redistributed such that large slack slots are generated and, even with large overheads, power state switches can be performed.

The DITD approach proposed in this paper achieves leakage energy reduction due to two main features: (1) it is temperature aware, which means that idle time is distributed such that the temperature is controlled in order to minimize leakage; (2) it redistributes slack such that the number of idle slots which are too short to switch power state, is minimized. A comparison between Fig. 10 and Fig. 11 illustrates the second feature of our ITD technique. However, the following question still remains open: How much does the temperature awareness of our approach contribute to the energy reduction? In order to answer this question we have repeated the above experiments considering a hypothetical scenario with zero switching overhead: $E_o = 0$mJ and $t_o = 0$ms. The results are shown in Fig. 12. Under such a scenario, the processor can be switched to the low power state for the duration of the total idle time (regardless the length of the individual idle slots). Thus, the energy gains obtained with DITD compared to SFA, as illustrated in Fig. 12, are exclusively due to the temperature awareness of the DITD approach.

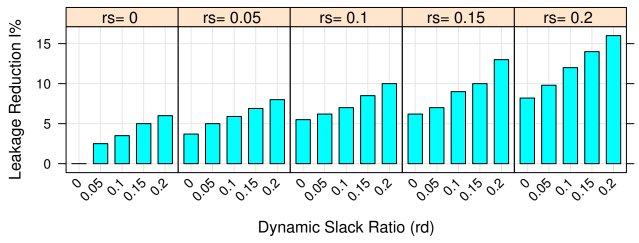


Fig. 12. Leakage Energy Reduction with No Switching Overheads

From Fig. 10, Fig. 11, and Fig. 12 one can also observe the efficiency of the ITD approach with only static slack (SITD, Section VI). The cases where $rd = 0$ (no dynamic slack) are, in fact, corresponding to those situations when only static slack is distributed. Obviously, in the cases that both $rs = 0$ and $rd = 0$, there is no slack to distribute and, thus, the energy reduction is zero.

Leakage Energy Reduction vs. Standard Deviation.
As mentioned, for our experiments we have generated workloads for each task τ_i according to a beta distribution Beta($\alpha 1_i$, $\alpha 2_i$), where $\alpha 1_i$ and $\alpha 2_i$ are determined based on the expected workload ENC_i and standard deviation σ_i of the executed workload. For the above experiments, the standard deviation σ_i for each task is considered to be: $\sigma_i = 0.1 \cdot (WNC_i - BNC_i)$. As the standard deviation has an influence on the potential leakage reduction, we have repeated the above experiments, considering three different settings of σ_i, namely, $0.2 \cdot (WNC_i - BNC_i)$, $0.15 \cdot (WNC_i - BNC_i)$, and $0.05 \cdot (WNC_i - BNC_i)$. Fig. 13 shows the leakage reduction $I\%$ by applying our DITD approach relative to the SFA, with different standard deviation settings. We have considered test applications having static and dynamic ratios of: $r_s = 0.2$ and $r_d = 0.2$. As can be observed,

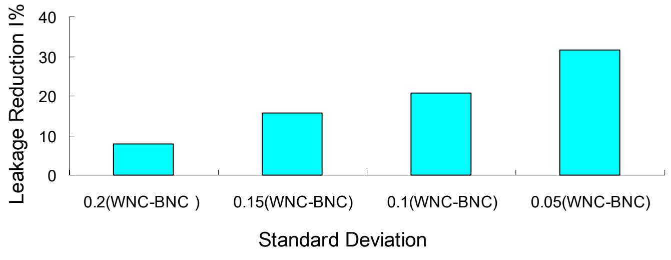


Fig. 13. Leakage Energy Reduction with Different Standard Deviations

the efficiency of the DITD approach increases as the standard deviation decreases. This is due to the fact that our DITD algorithm is targeted towards optimizing the energy consumption for the case that tasks execute the expected number of cycles ENC. When the standard deviation is smaller, more of the actual executed number of clock cycles are clustering around the ENC and, therefore, our DITD approach can achieve better leakage reduction.

Computation Time.
We have also evaluated the computation time for the off-line phase of our DITD approach. The results are given in Fig. 14.

MPEG2 Decoder.
We have applied our DITD approach to a real-life application, namely an MPEG2 decoder, which consists of 34 tasks. Details regarding the application are described in [45]. We have considered a platform with the size of the chip, heat spreader, and heat sink of 8×8mm^2, 18×18mm^2, and 22×22mm^2, respectively. The thickness of the chip, heat spreader, and the heat sink

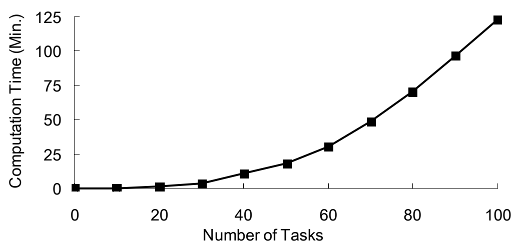


Fig. 14. Computation Time

is 0.5mm, 2mm, and 15mm, respectively. The execution time distribution of the tasks has been obtained from simulations on the MPARM platform [46]. We considered the following two overhead settings: (1) $E_o = 0.5$mJ, $t_o = 0.4$ms, (2) and $E_o = 1.0$mJ, $t_o = 0.8$ms. The leakage energy reduction by applying our DITD approach relative to the SFA approach is 32.5% and 40.8%, respectively.

IX. Conclusions

We first proposed a static temperature aware ITD approach for leakage energy optimization where only static slack is considered. In order to consider both static and dynamic slack, we then proposed a dynamic temperature aware ITD approach, which consists of an off-line and an on-line step. The experiments have demonstrated that considerable energy reduction can be achieved by our temperature aware ITD approaches. In order to efficiently perform temperature analysis inside our optimization loop for idle time distribution, we have also proposed a fast and accurate system level temperature analysis approach. Experiments show that our temperature analysis method achieves both good accuracy and high speed.

References

[1] International technology roadmap for semiconductors. http://public.itrs.net.

[2] A. Andrei, P. Eles, and Z. Peng, "Energy optimization of multiprocessor systems on chip by voltage selection," *IEEE Transactions on VLSI Systems*, vol. 15, pp. 262–275, 2007.

[3] Y. Liu, H. Yang, R. Dick, H. Wang, and L. Shang, "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," in *Proc. ISQED07*, 2007, pp. 204–209.

[4] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. ISLPED98*, 1998, pp. 197–202.

[5] W. C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 211–230, 2005.

[6] H. Aydi, P. Mejía-Alvarez, D. Mossé, and R. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. RTSS01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 95–105.

[7] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. M. Al-Hashimi, "Quasi-static voltage scaling for energy minimization with time constraints," in *Proc. DATE05*, 2005, pp. 514–519.

[8] C. Xian, Y. H. Lu, and Z. Li, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time," *IEEE Transactions on CAD*, vol. 27, no. 8, pp. 1467–1478, 2008.

[9] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for realtime embedded systems," in *Proc. DAC04*, 2004, pp. 275–280.

[10] F. Kreith, *The CRC Handbook of Thermal Engineering*. Boca Raton: CRC Press, 2000.

[11] W.Huang, S.Ghosh, S.Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on VLSI Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[12] Y. Yang, Z. P. Gu, R. P. Dick, and L. Shang, "Isac: Integrated space and time adaptive chip-package thermal analysis," *IEEE Transactions CAD*, vol. 26, no. 1, pp. 86–99, 2007.

[13] S. Wang and R. Bettatin, "Delay analysis in temp.-constrained hard real-time systems with general task arrivals," in *Proc. RTSS06*, 2006, pp. 323–334.

[14] R. Jayaseelan and T. Mitra, "A hybrid local-global approach for multi-core thermal management," in *Proc. ICCAD08*, 2008, pp. 618–623.

[15] S. Zhang and K. S. Chatha, "System-level thermal aware design of applications with uncertain execution time," in *Proc. ICCAD08*, 2008, pp. 242–249.

[16] R. Rao and S. Vrudhula, "Fast and accurate prediction of the steady-state throughput of multicore processors under thermal constraints," *IEEE Transactions on CAD*, vol. 28, no. 10, pp. 1559–1572, 2009.

[17] M. Sasaki, M. Ikeda, and K. Asada, "-1/+0.8°c error, accurate temperature sensor using 90nm 1v cmos for on-line thermal monitoring of vlsi circuits," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, pp. 201–208, 2008.

[18] M. Rajarshi and M. S. Ogrenci, "Systematic temperature sensor allocation and placement for microprocessors," in *Proc. DAC06*, 2006, pp. 542–547.

[19] A. N. Nowroz, R.Cochran, and S. Reda, "Thermal monitoring of real processors: Techniques for sensor allocation and full characterization," in *Proc. DAC10*, 2010, pp. 56–61.

[20] R. Cochran and S. Reda, "Spectral techniques for high-resolution thermal characterization with limited sensor data," in *Proc. DAC09*, 2009, pp. 478–483.

[21] S. Sharifi, C. Liu, and T. S. Rosing, "Accurate temperature estimation for efficient thermal management," in *Proc. ISQED08*, 2008, pp. 137–142.

[22] Y. F. Zhang and A. Srivastava, "Adaptive and autonomous thermal tracking for high performance computing systems," in *Proc. DAC10*, 2010, pp. 68–73.

[23] D. Brooks, R. P. Dick, R. Joseph, and L. Shang, "Power, thermal, and reliability modeling in nanometer-scale microprocessors," *IEEE Micro*, vol. 27, pp. 49–62, 2007.

[24] B. Nikhil, K. Tracy, and P. Kirk, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, pp. 1–39, 2007.

[25] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," in *Proc. DATE08*, 2008, pp. 288–293.

[26] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Proc. DAC10*, 2010, pp. 579–584.

[27] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Proc. ICCAD07*, 2007, pp. 281–288.

[28] S. Zhang and K. Chatha, "Thermal aware task sequencing on embedded processors," in *Proc. DAC10*, 2010, pp. 585–590.

[29] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," in *Proc. ICCAD08*, 2008, pp. 537–542.

[30] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware voltage selection for energy optimization," in *Proc. DATE08*, 2008, pp. 1083–1086.

[31] L. Yuan, S. Leventhal, and G. Qu, "Temperature-aware leakage minimization technique for real-time systems," in *Proc. ICCAD06*, 2006, pp. 761–764.

[32] C. Yang, J. Chen, L. Thiele, and T. Kuo, "Energy-efficient real-time task scheduling with temperature-dependent leakage," in *Proc. DATE10*, 2010, pp. 9–14.

[33] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling," in *Proc. DATE10*, 2010, pp. 21–26.

[34] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. ICCAD02*, 2002, pp. 721–725.

[35] W. P. Liao, L. He, and K. M. Lepak, "Temperature and supply voltage aware performance and power modeling at micro-architecture level," *IEEE Transactions on CAD*, vol. 24, no. No.7, pp. 1042–1053, 2005.

[36] Y. Liu, R. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," 2007, pp. 1–6.

[37] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Proc. CASES07*, 2007, pp. 257–266.

[38] W. H. Pressa, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. The Edinburgh Building, Cambridge, UK: Cambridge University Press, 2007.

[39] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. Society for Industrial Mathematics, 1987.

[40] Application note: Powerpc 970mp thermal considerations.

[41] Intel core 2 duo mobile processors on 65-nm process for embedded applications: Thermal design guide.

[42] Intel core 2 duo mobile processors on 45-nm process for embedded applications: Thermal design guide.

[43] S. Hsu, A. Alvandpour, and etc., "A 4.5-ghz 130-nm 32-kb l0 cache with a leakage-tolerant self reverse-bias bitline scheme," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 755–761, 2003.

[44] A. Macii, E. Macii, and M. Poncino, "Improving the efficiency of memory partitioning by address clustering," in *Proc. DATE03*, 2003, pp. 18–23.

[45] http://ffmpeg.mplayerhq.hu/.

[46] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Systemc cosimulation and emulation of multiprocessor soc designs," *Computer*, vol. 36, pp. 53–59, 2003.