On-Line Temperature-Aware Idle Time Distribution for Leakage Energy Optimization

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

***Abstract*—With new technologies, temperature has become an important issue to be considered at system level design. In this paper, we address the issue of leakage energy optimization through temperature aware idle time distribution (ITD). We propose an on-line ITD technique for leakage energy consumption minimization, 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.**

I. Introduction

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, temperature is an important parameter to be considered for system level power-aware techniques.

Many approaches to thermal aware system-level design aiming at energy optimization or temperature management have been proposed, e.g., [2], [3] and [4]. Static approaches are exclusively based on temperature models used at design time, e.g., Hotspot [5] which is both an architectural level and system level temperature simulator. A similar temperature modeling approach, proposed in the work in [6], speeds up the thermal analysis through dynamic adaptation of the resolution. Some fast system level temperature analysis techniques are proposed, e.g., [7] and [8] which are efficient to be used inside the temperature aware system level optimization loop. Several temperature aware system level design approaches, e.g., [9] and [10], are proposed in which decisions are taken on-line, based on the actual chip temperature information. For on-line temperature monitoring, sensors have been used together with techniques for collecting and analyzing their values with adequate accuracy, e.g., [11].

At system level, dynamic voltage selection (DVS) is one of the preferred approaches for reducing the overall energy consumption [12] and [13]. This technique exploits the available slack times 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 cycles than their WNC. However, very often, not all available slack should or can be exploited and certain amount of slack may still exist after DVS due to the existence of critical voltage [14]. 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 idle time. The only work, to our best knowledge, previously addressing this issue is [15] and [16]. However, the proposed ITD approaches only consider applications consisting of one single task executing at a constant given supply voltage. Thus, their ITD approach cannot optimize the distribution of idle time among multiple tasks which also execute at different voltages. In [8], we have proposed an ITD technique which eliminates the above limitations. However, only static slack is considered in the presentted ITD technique. *In this paper, we present an on-line ITD technique for leakage energy optimization which considers both static and dynamic slack. This approach is look up table (LUT) based and is composed of: an off-line and an on-line phase: (i) The off-line phase prepares a LUT for each task. (ii) At runtime, when a task is finished, the idle time length following the finishing of this task is decided by checking the task's LUT.*

II. Preliminaries

A. Power Model

For dynamic power we use the following equation [17]: $P_d = C_{eff} \cdot f \cdot V^2$, where C_{eff}, V, and f denote the effective switched capacitance, supply voltage, and frequency, respectively.

The leakage power is expressed as follows [18]: $P_{leak} = I_{sr} \cdot T^2 \cdot e^{\frac{\beta \cdot V + \gamma}{T}} \cdot V$, where I_{sr} is the reference leakage current at reference temperature, T is the current temperature, and β and γ are technology dependent coefficients.

B. Application and System Model

The application is captured as a task graph $G(\Pi, \Gamma)$. A node $\tau_i \in \Pi$ represents a computational task τ_i, 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 worse 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. Further, dl_i and $Ceff_i$ represent the deadline and the effectively charged 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

[1]Dept. of Computer and Information Science, Linköping University, Linköping, 58183, Sweden
[2]Ericsson AB, Linkoping, Sweden

be put to the idle state, consuming a very small amount of leakage power. We assume this leakage power P_{idle} to be constant due to its small amount. Switching the processor between the idle and active state as well as between different voltage levels incurs time and energy overheads (denoted as t_o and E_o respectively).

C. Thermal Analysis

Given is a periodic application made up of a set of tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ as described in Section II-B. The task set is mapped and scheduled to be executed on a processor. Each task is executed at given voltage levels. Given initial temperatures and workload of each task (te_i^A is given), the thermal model presented in [8] computes the corresponding transient temperature curve (TTC) of the processor during one execution iteration of the application.

Thermal Circuit. Our TTC estimation is based on an equivalent RC thermal circuit built with physical parameters of the die and the package of a given platform [19]. Due to the fact that the application period t_p can safely be considered significantly smaller than the RC time of the heat sink [20], the heat sink temperature stays constant during one iteration of the application. Hence, for TTC estimation, the thermal capacitance of the heat sink is ignored when we build the thermal circuit.

TTC Estimation. As illustrated in Fig. 1a, one input to TTC estimation is a voltage pattern which is determined by: 1) the given voltage levels at which tasks are executed, and 2) a given idle time distribution. The input voltage pattern is divided into

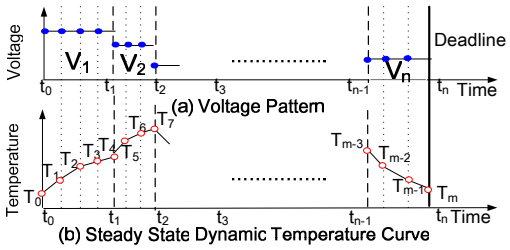

Fig. 1. Temperature Analysis

a number of m sub-intervals as shown in Fig. 1a. 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. The corresponding TTC is constructed by calculating the temperature values T_0 to T_m (Fig. 1b). Temperature points T_0 to T_m are obtained by solving a linear system established as follows:

1) The initial temperatures T_0 are given as input.
2) As leakage power stays constant inside one sub-interval, each temperature point T_i can be related to its prior point T_{i-1} via a linear equation whose coefficients are determined based on the inputs of the voltage pattern and our equivalent thermal circuit.

Solving the linear system, we get the values for T_0 to T_m and, hence, obtain the corresponding TTC.

D. Static Temperature Aware ITD (SITD)

In [8], we presented a static ITD technique (SITD). Given is a set of periodic tasks $(\tau_1, \tau_2, \ldots, \tau_n)$ executed in the order,τ_1, τ_2, …, τ_n, on a processor. Each task is characterized by a four-tuple: $\tau_i =< WNC_i, dl_i, Ceff_i, V_i >$. Our SITD approach optimize energy consumption by allocating idle slots between the execution of two neighbouring tasks. The optimization variables to be decided are t_i $(1 \leq i \leq n)$, which represents the length of the idle time slot allocated after the finishing of task τ_i. The objective function to be optimized is the total energy sum: $E_{tot}^{dyn} + E_{tot}^{leak} + E_{tot}^{o}$, where E_{tot}^{dyn} and E_{tot}^{leak} are the total dynamic energy and leakage energy consumption of the task set, respectively. E_{tot}^{o} is the total energy overhead due to switching between power states. As the voltage level V_i at which each task are executed are fixed, the total dynamic energy consumption, E_{tot}^{dyn}, is fixed. Thus, the actual objective function to be optimized is: $E_{tot}^{leak} + E_{tot}^{o}$.

Our SITD approach has been presented in [8]. The limitation of this SITD is that it assumes that tasks are always executed with their worst number of clock cycles and, thus, only static slack can be distributed for leakage optimization. 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. As we will demonstrate in the following sections, it is very important to consider dynamic slack in ITD for energy efficiency.

III. Motivational Example

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 C_{eff} are given in Table I. The tasks are executed on a processor with a fixed supply voltage and frequency level of $0.6V$ and 132MHZ respectively. The corresponding worst case execution times te^W of the 7 tasks are given in Table I. Given the performance of this processor, there exists $6ms$ static slack, ts, in each execution iteration of this application.

TABLE I
Motivational Example: Application Parameters

	$C_{eff}(f)$	WNC	$te^W(ms)$	ANC	$te^A(ms)$	$td^i(ms)$
τ_1	5.0e-10	8.26e+6	6.22	5.95e+5	0.45	5.77
τ_2	5.0e-10	1.20e+7	9.07	5.20e+5	0.40	8.67
τ_3	9.0e-8	2.32e+7	18.76	2.49e+7	18.76	0.0
τ_4	1.7e-7	2.25e+7	17.46	2.32e+7	17.46	0.0
τ_5	1.8e-7	1.46e+7	16.94	2.25e+7	16.94	0.0
τ_6	1.9e-7	2.15e+7	16.18	2.15e+7	16.18	0.0
τ_7	5.0e-10	8.26e+6	6.22	2.60.e+6	1.96	4.26

For the above example, we perform leakage energy minimization using the static temperature aware ITD method outlined in Section II-D. Fig. 2 gives the result, where t_s is divided into 3 segments of idle slots and the 3 idle slots are placed after execution of task τ_3, τ_4 and τ_5, respectively. For simplicity, in this example, we ignore both energy and time overhead due to switching between active and idle mode.

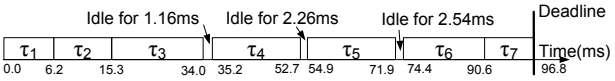

Fig. 2. Motivational Example: Static Idle Time Distribution

For this task set, let us imagine the activation scenario given in Table I where the columns ANC and te^A contains 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 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. Fig. 3a illustrates the distribution of idle time slots during the on-line activation scenario if we use the off-line ITD approach which distributes static slack as illustrated in Fig. 2. In this case, the dynamic slack td^i is placed where

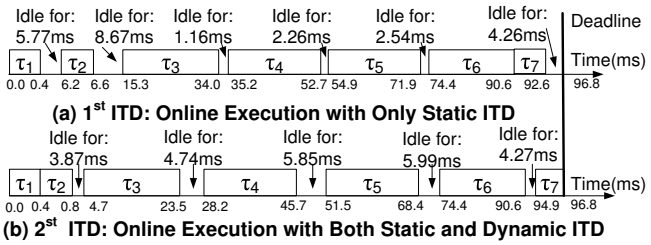

Fig. 3. Motivational Example: Idle Time Distribution

it is generated (td^i is placed after τ_i terminates). Table II (column SITD) shows the corresponding working temperature and leakage energy consumption of each task as well as the total leakage energy consumption which is 7.97J. 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. 3b shows the ITD determined in this way. The corresponding total leakage energy consumed, as given in Table II (column DITD), is 7.32J which means a leakage energy reduction of 9%.

TABLE II
LEAKAGE ENERGY COMPARISON

	SITD		DITD	
	$Tw(°C)$	$E^{leak}(J)$	$Tw(°C)$	$E^{leak}(J)$
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
$E^{leak}_{tot}(J)$		7.97		7.32

The example presented in this section demonstrates the importance of considering dynamic slack at ITD for energy efficiency. Hence, an on-line ITD approach is needed in order to make use of the dynamic slack created due to the execution time variation of tasks at runtime.

IV. ITD WITH BOTH DYNAMIC AND STATIC SLACK (DITD)

In order to exploit the dynamic slack in ITD, the slot length t_i (following the termination of task τ_i) has to be determined at the run-time based on the values of the current time and temperature at the termination of task τ_i. In principle, calculating the appropriate t_i implies the execution of a temperature aware ITD algorithm similar with the one outlined in Section II-D However, running this algorithm on-line, after execution of each task, implies a time and energy overhead which is not acceptable.

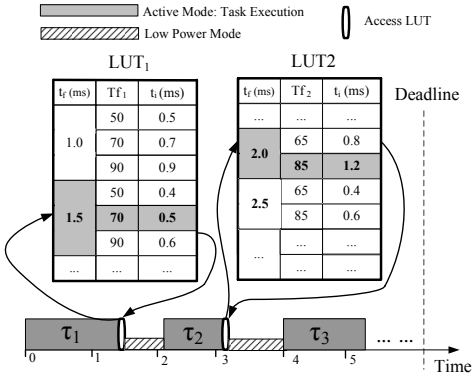

Fig. 4. DITD On-line Step

To overcome the above problem, we have divided our dynamic ITD (DITD) approach into two phases: an off-line phase 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 resulting idle time settings are stored in look-up tables (LUTs), one for each task. In Fig. 4, we show two such tables. They contain idle time settings for combinations of possible termination time t_f and finishing temperature Tf of a task τ_i. For example, in LUT_1, the line with finishing time 1.5 and temperature 70 stores the idle time slot length following the termination of task τ_1 in the situation when τ_1 finishes in the time interval $(1.0ms, 1.5ms]$ and the die temperature is in the interval $(60°C, 80°C]$[1]. In section IV-B we will present the generation procedure of the LUTs.

A. On-line Phase

The on-line phase is illustrated in Fig. 4. Each time a task τ_i terminates, the length of the idle time slot t_i following 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 the 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. 4, 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 based on the time and temperature values. As there is no exact entry with $t_1^f = 1.35ms$ and $Tf_1 = 78°C$, 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 $Tf = 78°C$) is chosen. The idle time following the termination of task τ_1 is determined to be 0.5ms. 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 time complexity, thus, very efficient.

B. Off-line Phase

In the off-line phase one LUT table is generated for each task such that the leakage energy consumption during execution is minimized. It is important to notice that the idle time is distributed so that leakage energy is minimized in case that tasks execute their expected number of cycles ENC which, in reality, happens with a much higher probability than e.g., WNC. Nevertheless, the idle time slot lengths are fixed such that, even in the worst case (tasks execute WNC), deadlines are satisfied.

The LUT table generation algorithm is illustrated in Fig. 5. 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 corresponding to the various possible finishing temperatures Tf_i 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) and latest possible finishing times (LFT), as well as the lowest Tf_i^l and highest possible finishing temperature Tf_i^h for each task. With a given finishing time t_i^f and finishing temperature Tf_i of task τ_i, in the innermost loop, the actual LUT entry for a task τ_i is calculated using the SITD algorithm together with the TTC estimation (outlined in Section II-D [8] and Section II-C, respectively).

[1]Unlike the time entry, the temperature entry is chosen based on the principle that the entry with temperature value nearest to the sensor reading T^{se} is chosen. If T^{se} is in the interval $(60°C, 80°C]$, than the temperature entry with value 70 is the nearest one to T^{se} among all temperature entries.

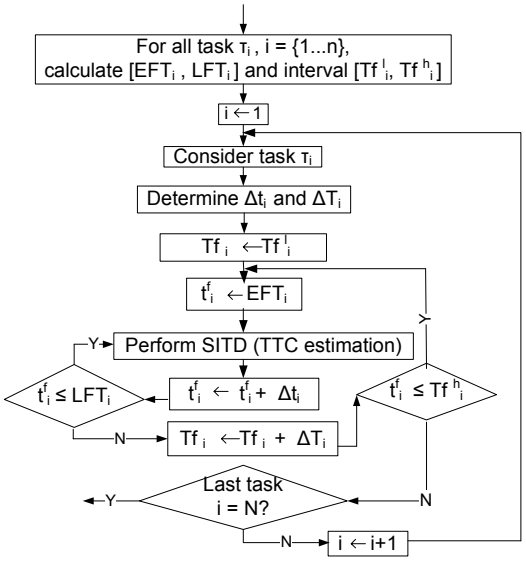

Fig. 5. DITD Off-line Step

For successive iterations, the finishing temperature and time will be increased with the time and temperature quanta $\triangle t_i$ and $\triangle T_i$. 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 IV-C and Section IV-D, respectively.

C. Time Bounds and Granularity

In the first step of the algorithm in Fig. 5, 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 by: $Nt_i = NL_t \cdot \frac{(LFT_i - EFT_i)}{\sum_{i=1}^{n}(LFT_i - EFT_i)}$

The corresponding granularity along the time dimension $\triangle t_i$ for task τ_i is the same for all tasks and is obtained as: $\triangle t_i = \frac{\sum_{i=1}^{n}(LFT_i - EFT_i)}{NL_t}$.

D. Temperature Bounds and Granularity

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

To determine the number of entries along the temperature dimension, we still need to determine the temperature interval $[Tf_i^l, Tf_i^h]$ at the termination of each task. Intuitively, the temperature variation is strongly dependent on the tasks' execution time variation. Theoretically, with a given thermal circuit, the temperature interval can be obtained knowing the tasks' execution time variation. Unfortunately, performing precise analysis of temperature variation for each task is not feasible in practice due to the analysis complexity. On the other hand, it would be too pessimistic to simply assume that all tasks have 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. As this can lead to huge amounts of wasted memory space (for storing LUT tables) as well as computation time in the off-line phase.

To balance the computation complexity and accuracy of the temperature interval analysis, we propose the following approach for estimating the temperature interval $[Tf_i^l, Tf_i^h]$ for each task τ_i. We should note that it is not important 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 exceeds 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.

Temperature Bounds Analysis for A Simple Example. Let us first look at the following simplified example. As shown

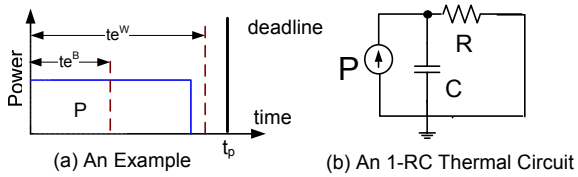

Fig. 6. An 1-RC Thermal Circuit

in Fig. 6a, a periodic application (period is t_p) containing a single task τ_1 is executing on a processor. The actual execution time of τ_1 is te^A, which is in the interval $[te^B, te^W]$, where te^B and te^W are the best and worst case execution time of τ_1, respectively. During the execution of τ_1, the processor has a constant power consumption, P. For simplicity, we use a simple 1-RC thermal circuit, as shown in Fig. 6b, to model the processor's temperature behaviour, where R and C represent the thermal resistance and capacitance of the processor respectively.

The temperature at the termination of the task τ_1 is expressed as follows:

$$T^f = P \cdot R - (P \cdot R - T^{st}) \cdot e^{(\frac{-te^A}{\tau_{rc}})} \quad (1)$$

where T^{st} is the initial temperature, τ_{rc} is the RC time constant of the processor: $\tau_{rc} = R \cdot C$, and $P \cdot R$ is the steady steady temperature the processor would reach if task τ_1 is execute for sufficiently long time. After the termination of τ_1 and before τ_1 starts for the next iteration, the processor is switched to idle state[2] during the idle slot with length $t_p - te^A$. The temperature at the end of the idle slot is expressed as follows.

$$T^I = T_a + (T^f - T_a) \cdot e^{(\frac{-t_p + te^A}{\tau_{rc}})} \quad (2)$$

From Eq. (1) and Eq. (2), we can see that, with given initial temperature T^{st}, both T^f and T^I are monotonically increasing as te^A increases. If the application is executed for number of N iterations, then, the starting and finishing temperature of the task τ_1 for each iteration j^{th} (denoted as T_j^{st} and T_j^f respectively) can be expressed by Eq. (3) and Eq. (4), where te_j^A represents the actual execution time of τ_i in j^{th} iteration. It is important to notice from Eq. (3)

[2]For simplicity, we assume no energy and time overheads is incurred when power state is changed , and the processor power consumption in idle state is zero.

and Eq. (4) that, for each iteration, temperature values T_j^f and T_{j+1}^{st} are both monotonically increasing as te_j^A increases. Therefore, the highest possible finishing temperature of the task after number of N iterations, T_N^f, is achieved when the task is always executed with its worst case execution time: $te_j^A = te^W$ ($\forall j\, 1 \leq j \leq N$). Similarly, the lowest finishing temperature is reached when the task is always executed with its best case execution time: $te_j^A = te^B$ ($\forall j\, 1 \leq j \leq N$).

$$T_j^{st} = T_a + (T_{j-1}^f - T_a) \cdot e^{(\frac{-t_p + te_{j-1}^A}{\tau_{rc}})} \quad (3)$$

$$T_j^f = P \cdot R - (P \cdot R - T_j^{st}) \cdot e^{(\frac{-te_j^A}{\tau_{rc}})} \quad (4)$$

Temperature Bounds Estimation. Our method of estimating temperature bounds [Tf_i^h,Tf_i^l] is based on the observation from the above example. We first define two on-line execution scenarios.

- *Worst case on-line execution scenario*: it is an on-line execution scenario when 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 on-line execution scenario*: it is an on-line execution scenario when the actual execution time of each task τ_i is always equal to its worst case execution time: $te_i^A = te_i^B$.

In both scenarios above, the processor will execute the corresponding periodic power pattern repeatedly and the processor temperature will eventually reach to steady state dynamic temperature curves (denoted as $SSDTC^w$ and $SSDTC^b$ respectively), after executing the application for sufficiently large number of times. From both $SSDTC^w$ and $SSDTC^b$, we can obtain, for each task τ_i, its finishing temperature. We use the finishing temperature of task τ_i corresponding to the $SSDTC^w$ (arrived in the *worst case on-line execution scenario*), Tf_i^w, as the upper bound of the finishing temperature of task τ_i: $T_i^h = Tf_i^w$; and the finishing temperature of task τ_i corresponding to the $SSDTC^b$ (arrived in the *best case on-line execution scenario*), Tf_i^b, as the lower bound: $T_i^l = Tf_i^b$.

We obtain $SSDTC^w$ by performing the SITD outlined in Section II-D. To obtain $SSDTC^b$, we perform the SITD for the *best case on-line execution scenario*, with the only difference that the available idle time is computed with the consideration that task execution time is always te_i^B (instead of te_i^W, as we do in $SSDTC^w$ computation).

With obtained upper and lower bounds [Tf_i^l, Tf_i^h] for each task τ_i, the considered temperature values in LUT_i are determined as follows: $Tf_i^{die} = k \cdot \triangle T_i + Tf_i^l$, where $\triangle T_i$ is the granularity along temperature dimension and the integer k is in the interval: $0 \leq k < \lceil \frac{(Tf_i^h - Tf_i^l)}{\triangle T_i} \rceil$.

V. Experimental Results

We have used both generated test applications as well as a real life example in our experiments to evaluate our on-line ITD approach presented in Section IV.

A. Test Applications Generation

We have randomly generated 100 test applications consisting of 10 to 100 tasks. The workload in the worst case (WNC) for each task is generated randomly in the range [10^6, $5.0\,e10^6$] clock cycles, while the workload in the best case is generated in the range [10^5, $5.0\,e10^5$] clock cycles. To generate the expected workload of each task ENC_i, the following steps are performed:

1) The value of the expected total dynamic idle time, t_d^E, is given as an input, where 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) Divide t_d^E into a number n_{sub} of multiple sub-intervals with equal length (t_{sub}).
3) The n_{sub} sub-intervals are allocated among all tasks within application based on a uniform distribution, as result, each task is allocated number of p 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.

B. Leakage Energy Reduction

To evaluate our DITD approach described in Section IV, we have compared it with the static ITD approach as outlined in Section II-D. The SITD approach is applied to each generated test applications, and the following steps are performed.

1) Firstly, SITD is performed to distribute static slack, as result, we obtain t_i ($1 \leq i \leq n$) which are the idle slot lengths following the termination of τ_{i-1}. The start time of each task t_i^{st} is, thus, determined off-line by: $t_i^{st} = te_{i-1}^W + t_{i-1}^{st} + t_{i-1}$, where te_{i-1}^W is the worst case execution time of the previous task τ_{i-1}.
2) On-line execution of the application is simulated. Whenever a task τ_i terminates, we compute the gap t_g: $t_g = t_{i+1}^{st} - t_i^f$, where t_i^f is the termination time of the current task and t_{i+1}^{st} is the start time of the next task.
3) If $t_g = 0$, the next task τ_{i+1} starts immediately after the termination of task τ_i. For the cases when $t_g > 0$, the processor will be switched to idle state during t_g, if the following two conditions are both satisfied; otherwise the processor will stay in the same active state at which task τ_i is executed: (a) $t_g > t_o$, where t_o is the time overhead due to power 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 same active state at which τ_i is executed. 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 processor power consumption in idle state.

We have applied both DITD and SITD approaches on the same test applications simulated for number of N iterations[3]. The Hotspot system [5] 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 granularities along time and temperature dimensions for the LUT table 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 produced 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 [21] and [22]. The energy and time overheads due to power state switching is set to $E_o = 0.5mJ$ and $t_o = 0.4ms$, respectively, according to [14]. After performing both DITD and SITD approaches on a test application and run it for N iterations, we compute the corresponding leakage energy reduction by dynamic approach compared to the static

[3]We execute the test applications for large number of iterations, such that the time spent in the initial heating up can be ignored

one for each iteration: $I_j = (E_j^{SITD} - E_j^{DITD})/E_j^{SITD} \cdot 100\%$ where E_j^{SITD} and E_j^{DITD} are leakage energy corresponding to SITD and DITD approach at j^{th} iteration, respectively. The averaged leakage energy reduction over N iterations is, then, calculated: $I = \sum_{j=1}^{N} I_j/N$.

We assume, for each task τ_i, that the actual executed workload at runtime conforms to the beta distribution Beta($\alpha 1_i, \alpha 2_i$), and when we simulate the on-line execution of test applications, the actual executed clock cycles of each task is generated using the corresponding beta distributed random generator. The parameters $\alpha 1_i$ and $\alpha 2_i$ are determined by (1) the expected workload ENC_i (which is determined as described in Section V-A), and (2) a given standard deviation of the executed clock cycles of task τ_i, σ_i. We have considered four different values of standard deviation σ_i: $5\% \cdot (WNC_i - BNC_i)$, $10\% \cdot (WNC_i - BNC_i)$, $15\% \cdot (WNC_i - BNC_i)$, and $20\% \cdot (WNC_i - BNC_i)$ in our experiments. As the amount of available dynamic slack has strong influence on the potential efficiency of a dynamic ITD approach, we have performed the experiments considering different dynamic slack ratios, r_d, which is calculated as: $r_d = t_d^E/dl$, where t_d^E is the expected total dynamic slack as described in Section V-A. Fig. 7 shows, for each consideration of the standard deivation setting σ_i, the averaged leakage energy reduction I relative to the static one. As can be observed, the efficiency of the dynamic approach, compared to the static one, increases as the dynamic slack ratio r_d grows. As our DITD approach is

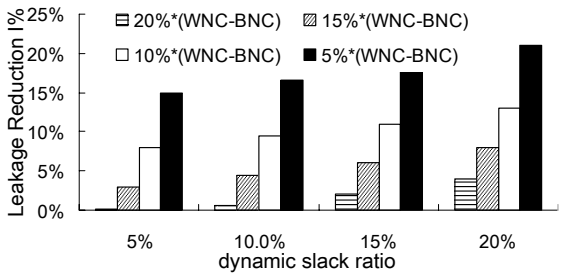

Fig. 7. Leakage Energy Reduction

targeted towards optimizing the energy consumption for the case that tasks execute their expected number of cycles ENC. Therefore, energy savings are larger, compared to the static approach, when the standard deviation is smaller (more of the actual executed number of clock cycles are clustering around the ENC).

We have also evaluated the computation time for the off-line phase of our DITD approach. The results are given in Fig. 8. As can be observed from Fig. 8, the computation time of the off-line phase of our DITD approach is a function of the application size (characterized by the number of tasks). The computation time is around 2 hours for very large applications, e.g., applications containing 100 tasks.

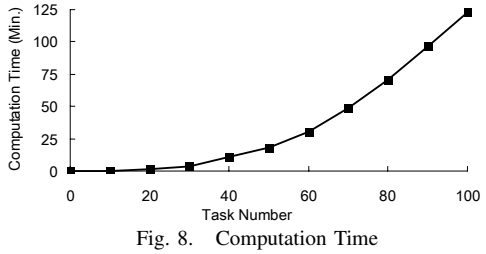

Fig. 8. Computation Time

C. Real Life Example

We have also applied our DITD approach to a real life case, namely an MPEG2 decoder which consists of 34 tasks and is described in detail in [23]. The leakage energy reduction applying our DITD approach relative to the static ITD approach is 26.2%.

VI. Conclusion

We have proposed a dynamic idle time distribution heuristic for energy minimization. This ITD approach considers both static and dynamic idle time and consists of an off-line and an on-line step. Our experiments demonstrate that considerable energy reduction can be achieved by our on-line idle time distribution compared to a static approach.

References

[1] "http://public.itrs.net., international technology roadmap for semiconductors."

[2] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," *Design Automation Conference*, pp. 579–584, Jun. 2010.

[3] S. Zhang and K. Chatha, "Thermal aware task sequencing on embedded processors," *Design Automation Conference*, pp. 585–590, Jun. 2010.

[4] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," *Design, Automation and Test in Europe*, pp. 288–293, Mar. 2008.

[5] 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 VLSI Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[6] Y. Yang, Z. P. Gu, R. P. Dick, and L. Shang, "Isac: Integrated space and time adaptive chip-package thermal analysis," *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, pp. 86–99, Jan. 2007.

[7] R. Rao and S. Vrudhula, "Fast and accurate prediction of the steady-state throughput of multicore processors under thermal constraints," *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems*, vol. 28, no. 10, pp. 1559–1572, Oct. 2009.

[8] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling," *Design Automation and Test in Europe*, pp. 21–26, Mar. 2010.

[9] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in mpsocs," *Design, Automation Test in Europe*, no. No.7, pp. 1–6, Apr. 2007.

[10] H. Jung, P. Rong, and M. Pedram, "Stochastic modeling of a thermally-managed multi-core system," *Design Automation Conference*, pp. 728–733, Jun. 2008.

[11] Y. F. Zhang and A. Srivastava, "Adaptive and autonomous thermal tracking for high performance computing systems," *Design Automation Conference*, pp. 68–73, Jun. 2010.

[12] A. Andrei, P. Eles, and Z. Peng, "Energy optimization of multiprocessor systems on chip by voltage selection," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, no. 3, pp. 262–275, Mar. 2007.

[13] Y. Liu, H. Yang, R. Dick, H. Wang, and L. Shang, "Thermal vs energy optimization for dvfs-enabled processors in embedded systems," *International Symposium on Quality Electronic Design*, pp. 204–209, Mar. 2007.

[14] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for realtime embedded systems," *Design automation Conference*, pp. 275–280, Jun. 2004.

[15] L. Yuan, S. Leventhal, and G. Qu, "Temperature-aware leakage minimization technique for real-time systems," *International Conference on Computer Aided Design*, pp. 761–764, 2006.

[16] C. Yang, J. Chen, L. Thiele, and T. Kuo, "Energy-efficient real-time task scheduling with temperature-dependent leakage," *Design Automation and Test in Europe*, pp. 9–14, Mar. 2010.

[17] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," *International Conference on Computer Aided Design*, pp. 721–725, Nov. 2002.

[18] W. P. Liao, L. He, and K. M. Lepak, "Temperature and supply voltage aware performance and power modeling at micro-architecture level," *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. No.7, pp. 1042–1053, Jul. 2005.

[19] A. Krum, *Thermal management. In The CRC Handbook of Thermal Engineering*. Boca Raton: F. Kreith, Ed. CRC Press, 2000.

[20] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 257–266, Nov. 2007.

[21] S. Hsu, A. Alvandpour, S. Mathew, and et al., "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*, pp. 755–761, Mar. 2003.

[22] A. Macii, E. Macii, and M. Poncino, "Improving the efficiency of memory partitioning by address clustering," *Design Automation and Test in Europe*, pp. 18–23, 2003.

[23] "http://ffmpeg.mplayerhq.hu/."