Temperature-Aware Leakage Minimization Techniques for Real-Time Systems

Abstract

In this paper, we study the interdependencies between system's leakage and on-chip temperature. We show that the temperature variation caused by on-chip heat accumulation has a large impact in estimating the system's leakage energy. More importantly, we propose an online temperature-aware leakage minimization technique to demonstrate how to incorporate the temperature information to reduce energy consumption at real time. The basic idea is to run when the system is cool and the workload is high and to put the system to sleep when it is hot and the workload is light. The online algorithm has low run-time complexity and achieves significant leakage energy saving. In fact, we are able to get about 25% leakage reduction on both real life and artificial benchmarks. Comparing to our optimal offline algorithm, the above online algorithm provides similar energy savings with similar decisions on how to put the system to sleep and how to wake it up. Finally, our temperature-aware leakage minimization techniques can be combined with existing DVS methods to improve the total energy efficiency by further saving on leakage.

1 Introduction

As technology scales down to the deep sub-micron (DSM) domain, chip power dissipation and power density are increasing rapidly. Dynamic power and leakage power are the two main contributors for chip power dissipation. The recent trend is that leakage power is becoming more and more significant and is predicted to be the dominant source of power dissipation in the near future [2]. For example, leakage accounts for almost 50% of total power on Intel Pentium IV processors [5]. Meanwhile, the power density is doubled every three years and is projected to reach $100W/cm^2$ at process technologies below 50nm [23]. These have posed critical challenges for DSM design.

One of such challenges is power and energy efficient system design, particularly for real-time systems where energy is limited. Dynamic voltage scaling (DVS) technique is among the most effective in reducing system's dynamic power and energy consumption. To obtain the maximal dynamic energy reduction, DVS method aggressively slows down the task's execution such that the completion occurs at the task's deadline sharply. However, this comes with a longer execution time which results in larger leakage energy dissipation. With the steep increase of leakage, new DVS policies have been proposed to minimize system's total power and energy consumption, instead of the dynamic part only [8, 11, 21]. Basically, they propose to operate the system at a speed higher than the minimum and shut down the system or put it into a $sleep$ state when the task is completed earlier than deadline. This trades the dynamic energy saving for more leakage saving in order to obtain the maximal total energy reduction.

These power and energy minimization methods will help on-chip cooling device to keep the hot chip cool. However, with the rapid increase of power density, chip temperature goes up accordingly. High temperature not only affects the performance and reliability of the chip, but also has a significant impact to the leakage current. Unfortunately, most leakage reduction techniques do not consider this impact. In fact, transistor's leakage current has an exponential dependence on temperature according to the Berkeley BSIM model [22]. This interdependency between leakage and temperature implies that if the system is not designed properly, chip temperature and leakage power will interact in a positive feedback loop and lead to thermal runaway.

To prevent this from happening, several temperature management techniques have been proposed for general purpose processors and large sever systems [3, 7, 15]. The architectural level dynamic thermal management (DTM) technique [3, 7] employs a temperature monitoring mechanism with the help of on-chip thermal sensors or thermal estimation. If the chip temperature is higher than a pre-determined maximal temperature threshold, certain thermal adjustment techniques will be engaged to reduce the temperature. These techniques include clock frequency/voltage scaling, fetch-toggling, instruction throttling, and control theoretic based approaches. When the temperature drops below a pre-determined minimal temperature threshold, the processor will return to the normal execution mode.

The DTM technique is very effective in controlling the peak temperature for general purpose processors [3, 7, 15]. However, when the thermal adjustment techniques are used, the system will slow down. Therefore, DTM cannot guarantee the deadlines that most real-time tasks require and is not suitable for real-time systems. In addition, the values of max and min temperature thresholds will affect the energy consumption to complete a real-time task. DTM selects such thresholds in order to control the peak temperature, and its solution may not be good for energy minimization, another important requirement for real-time systems.

Our goal in this paper is to show (1) the importance of considering temperature variation during task execution for energy minimization, leakage in particular; and (2) how to incorporate temperature information for energy reduction in real-time systems.

1.1 Motivational Example

We consider an asynchronous digital subscriber line (ADSL) modem application with a deadline 2048ms and workload 864ms based on the ADSL standard [19]. Figure 1 shows the energy consumption for executing this application under different approaches.

No-DVS is the traditional way of running at full speed for 864ms and then going to sleep. Its energy consumption is 40.49J, in which 21.46J is on dynamic and 19.03J is on leakage. *DVS* operates at the lowest voltage that can complete at the 2048ms deadline. It reduces the dynamic energy to the minimal level of 6.56J, but consumes more leakage due to its longer execution time. *CS-DVS* [8] runs at a slightly higher speed than DVS in order to minimize total energy. Comparing to DVS, the dynamic energy using CS-DVS increases to 7.02J, but the leakage energy decreases to 9.40J, and total energy is reduced to 16.43J. Our proposed Temperature-Aware LeaKage *TALK* minimization method runs at the same speed as CS-DVS with more than 22% energy saving on leakage.

Leakage increases rapidly as temperature rises. In the above approaches, leakage is calculated with transient temperature (See section 3 for details.). However, existing literature treats temperature as a constant during the execution. Figure 1 also reports the leakage estimation with temperature fixed at the highest temperature 388K and the lowest starting temperature 300K. One can easily see that it is important to consider temperature.s impact to leakage.

The idea behind TALK is to restrict the execution at high temperature. This is clearly shown in Figure 2 where the temperature curve for each approach is given. Although No-DVS has the shortest execution time, it has the largest leakage because it executes

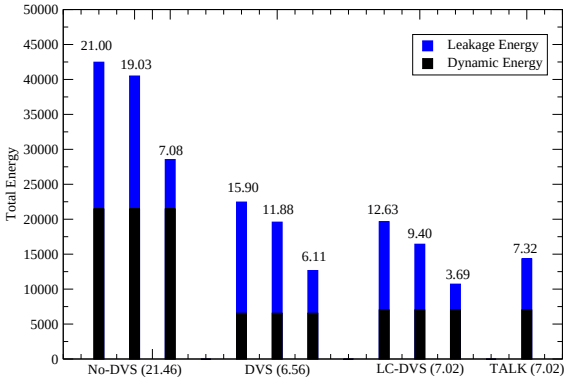


Figure 1: Energy consumption for executing ADSL application under different approaches. For each approach, three leakage numbers are obtained with temperature, from left to right, fixed at the highest, varied based on the accumulated heat, and fixed at the lowest temperature. Dynamic energy is shown in parentheses and leakage energy is shown in the figure on top of each bar.

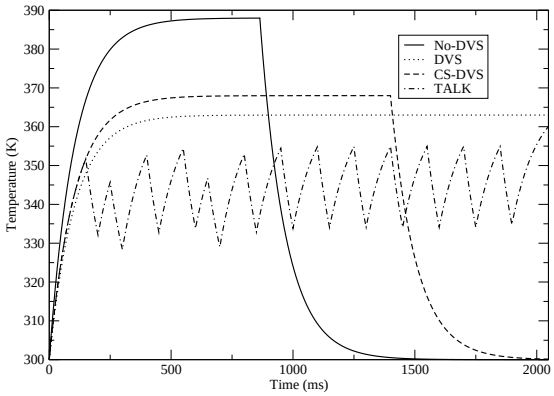


Figure 2: Temperature behavior for executing ADSL benchmark with No-DVS, DVS, CS-DVS, and TALK.

at high temperature. CS-DVS has less leakage than DVS because it completes earlier and can go to the sleep (or shut down) mode where there is no or little leak. However, CS-DVS still executes at a relatively high temperature, higher than 360K, for a long time. TALK monitors the chip temperature and decides whether to execute or to cool the system down. Consequently, its temperature curve goes up and down and most of the time it runs at temperature lower than 350K.

1.2 Contributions

The contribution of this paper are the following: we study the temperature and leakage interdependencies for real-time systems; we propose optimal offline and efficient online temperature-aware leakage minimization algorithm to adjust the system's active/sleep mode during task execution in order to reduce the leakage energy. Both algorithm can take the wakeup overhead in time and energy as well as the energy dissipation, if any, at sleep mode into consideration. The proposed online algorithm requires little hardware support and has very low run-time complexity. We simulate both algorithms on real life benchmarks. The results demonstrate that 1) both the offline and online algorithm can reduce leakage energy by about 25% with moderately small number of wake-ups; 2) both the online algorithm's energy saving and its decision on system active/wakeup are very close to those of the optimal offline algorithm; and 3) both algorithms can be combined with existing DVS techniques to further enhance their energy efficiency.

2 Related Work

A large amount of work have been done in designing DVS algorithms to reduce the dynamic power consumption in real-time systems [1, 6, 10, 13, 17]. However, the energy saving achieved by DVS comes at the cost of extended execution time, which means the system will leak for a longer period and generate more leakage energy. Recently, due to the steep increase in leakage power, several new DVS techniques have been proposed to minimize the total energy in the real-time system [8, 20, 21]. In [20], the author proposed a combined supply voltage and threshold voltage scaling technique to minimize dynamic power and leakage power simultaneously. In [8] and [21], both authors identify a critical supply voltage in the DVS design space, such that if the supply voltage is scaled down below the critical voltage, the increase in leakage energy will surpass the reduction in dynamic energy and thus increase the total energy consumption. Based on such critical voltage/speed, [8] proposed a scheduling algorithm that keeps the tasks running back-to-back and leaves a long idle duration for the system to shutdown. None of these approaches consider the temperature effect in the leakage power. That is, they assume temperature is a constant.

On the other hand, temperature management have received more attention in general purpose processor design. Skadron et al. presents a microarchitectural level thermal model that considers both the temporal and local thermal effects on chip [16]. A few dynamic temperature management techniques have been proposed to control the peak temperature in processors. A trigger mechanism is designed in [14], where an on-chip sensor is used to measure the temperature; whenever the temperature exceeds a predetermined threshold, a *throttling* technique is used to reduce the number of instructions flowing to the processor core. Huang et al. [7] proposed a framework for temperature management at operating system levels. Brooks et al. [3] eliminated the delay overhead of temperature sensor by estimating the temperature based on the power consumption in a thermal window. [15] used a feedback control theory based approach to tune the system performance at a finer granularity. Srinivasan et al.[18] proposed an thermal management technique for multimedia applications. The runtime temperature is predicted based on the starting temperature. However, they did not consider leakage power, which is an unnegligible portion of total power and is temperature sensible.

3 Preliminaries

3.1 System model

We study systems that execute a set of real-time tasks (also called applications or jobs). To keep our focus on the temperature-aware energy minimization, we assume that the tasks have already been scheduled to meet their deadlines. That is, each task has its scheduled starting time, completion time, and worst case execution time. If the system is DVS enabled, each task will also have its operating voltage and speed. Interested readers can refer to [1, 8, 10, 13, 17] for how to determine such schedule.

The system has two operation modes: an *active* mode and a *sleep* mode. During the *active* mode, the system executes the jobs and dissipates both dynamic and leakage power; during the *sleep* mode, no job is performed and there is only leakage which is negligible with leakage control mechanisms such as power gating. When the system switches from the *sleep* mode to *active* mode, an additional time, called "wakeup" time, is needed before any job can be executed. Normally, a small amount of energy is consumed during this "wakeup" time.

3.2 Thermal model

During the *active* mode, both the dynamic and leakage energy dissipated by the system will be converted into heat. Some of the heat will be removed by the cooling device such as a heatsink; the rest heat will accumulate and result in a temperature increase on the chip. Similarly, when the system enters the *sleep* mode, the chip will begin to cool down since little power is produced during the *sleep* mode. We apply the model described in [15] to characterize thermal behavior in both heating and cooling of the chip. It is based on the well-known duality between heat transfer and electrical phenomena in RC circuits.

The change in temperature T over time t on chip can be described using equation (1) below:

$$T(t) = P \cdot R_{th} + T_{amb} + (T_{cur} - P \cdot R_{th} - T_{amb}) \cdot e^{-\frac{t}{R_{th}C_{th}}} \quad (1)$$

where P is the transient power dissipated in the chip; R_{th} and C_{th} are the equivalent thermal resistance and thermal capacitance; T_{amb} is the ambient temperature of the environment, normally assumed to be room temperature; and T_{cur} is the current temperature of the chip.

When chip temperature rises, the system will reach a state where the amount of heat generated during a period of time becomes equal to the amount of heat being removed by the heatsink. Hence there will be almost no temperature variation and this state is generally called the *thermal equilibrium*. Let's denote the temperature at this state as K_1. Similarly, when the chip cools down, it will also reach a temperature K_2 where no more heat can be removed by the heatsink. K_2 is normally very close to the ambient temperature. We further denote the product $R_{th}C_{th}$ as K_3. Now, we can use the following functions to describe the temperature rise and fall:

$$T_{rise}(t) = K_1 - (K_1 - T_{cur})e^{-\frac{t}{K_3}} \quad (2)$$

$$T_{fall}(t) = K_2 + (T_{cur} - K_2)e^{-\frac{t}{K_3}} \quad (3)$$

3.3 Energy model

System's energy consumption consists of dynamic energy and leakage energy. Dynamic energy is proportional to $C_s \cdot V_{dd}^2$, where C_s is the effective capacitance and V_{dd} is the supply voltage. Note that dynamic energy is independent of temperature. Leakage energy is caused by leakage current flowing in the CMOS circuits, which includes gate leakage and subthreshold leakage. The leakage power can be calculated as:

$$P_{leakage} = N_g \cdot I_{leakage} \cdot V_{dd} \quad (4)$$

where N_g is the number of equivalent transistors in the system and $I_{leakage}$ is the leakage current that can be modeled as follows for 65nm technology [12]:

$$I_{leakage} = A \cdot T^2 \cdot e^{\frac{\alpha V_{dd} + \beta}{T}} + B \cdot e^{\gamma V_{dd} + \delta} \quad (5)$$

In this formula, $A, B, \alpha, \beta, \gamma, and \delta$ are empirical constants that can be found in [12]; T is the temperature. The first term denotes the subthreshold leakage that increases as T goes up. The second term is gate leakage, which is insensitive to temperature and is projected to be controlled by high-K material [9]. Therefore, we focus on the subthreshold leakage.

In summary, when the system stays at *active* mode for the interval of $[t_1, t_2]$ and goes to *sleep* mode for the interval of $[t_2, t_3]$, its energy dissipation will be

$$E_{total} = \int_{t_1}^{t_2} P_{active} dt + \int_{t_2}^{t_3} P_{sleep} dt$$
$$= (C_s V_{dd}^2 \cdot (t_2 - t_1) + \int_{t_1}^{t_2} P_{leakage} dt) + P_{sleep} \cdot (t_3 - t_2) \quad (6)$$

where P_{sleep} is the power when the system is at *sleep* mode. Note that the leakage power $P_{leakage}$ depends on temperature which changes over time as indicated from equations (1)-(5).

4 Temperature-Aware Leakage Minimization

In this section, we will formulate the temperature-aware leakage minimization problem and describe our offline and online techniques to solve this problem.

4.1 Problem Formulation

As we have discussed in the above section, we restrict our study on a single scheduled task and use the deadline and workload pair (D, W) to represent its scheduled starting time, completion time, and worst case execution time. Furthermore, the task is scheduled to be operated at voltage V_{dd}. Define $x(t)$ to be 1 or 0 if the system is at the *active* or *sleep* mode at time t. Our goal is to determine function $x(t)$ in interval $[0, D]$ such that the workload W can be completed and the total energy expressed below is minimized:

$$\int_0^D (P_{active} \cdot x(t) + P_{sleep} \cdot (1 - x(t)) dt \quad (7)$$

Since dynamic energy is independent of temperature and the system's total active time will be W to complete the workload, the dynamic energy part of equation (7) will be a constant. The second term in equation (7) will also be the constant $P_{sleep} \cdot (D - W)$ where P_{sleep} is constant. Hence we can formulate this as the following temperature-aware leakage (TALK) minimization problem: *determining $x(t)$ such that*

$$\int_0^D x(t) dt \geq W \quad (8)$$

$$\text{and minimizes} \quad \int_0^D P_{leakage}(t) \cdot x(t) dt \quad (9)$$

Since leakage energy in equation (9) depends on temperature and also impacts temperature through hear accumulation, this becomes a non-linear feedback control problem which is known to be hard. A practical formulation is to partition the interval $[0, D]$ into $0 = t_0 < t_1 < \cdots < t_N = D$ and to find $x(t)$ such that

$$x(t) \text{is constant in each interval} [t_i, t_{i+1}) \quad i = 0, 1, \cdots, N-1 \quad (10)$$

$$\sum_{i=1}^{N-1} x(t_i) \cdot (t_{i+1} - t_i) \geq W \quad (11)$$

$$\text{and to minimize} \quad \sum_{i=1}^{N-1} \int_{t_i}^{t_{i+1}} P_{leakage}(t) \cdot x(t_i) dt \quad (12)$$

Note that when interval $[t_i, t_{i+1})$ is small, we can approximate the leakage power as a constant. Furthermore, we can integrate the wakeup time t_{wakeup} and energy E_{wakeup} overhead by defining $w_i = 1$ if $x(t_i) > x(t_{i-1})$ and $w_i = 0$ otherwise. Thus, we can rewrite the above equations (11) and (12) by:

$$\sum_{i=1}^{N-1} x(t_i) \cdot (t_{i+1} - t_i - w_i \cdot t_{wakeup}) \geq W \quad (13)$$

$$\sum_{i=1}^{N-1} (P_{leakage}(t_i) \cdot x(t_i) \cdot (t_{i+1} - t_i) + w_i \cdot E_{wakeup}) \quad (14)$$

4.2 Offline TALK Minimization

For the simplicity of discussion, we assume that $P_{sleep} = E_{wakeup} = t_{wakeup} = 0$ and $t_{i+1} - t_i = \frac{D}{N}$. However, our proposed TALK minimization techniques can take such concerns into consideration as formulated explicitly in equations (13) and (14). A solution to this problem will be the values of $x(t_i)$ for $i = 0, 1, \cdots, N-1$. The offline algorithm determines such values based on a dynamic programming paradigm and the online algorithm decides the value of $x(t_i)$ at time t_i during the task's execution.

Suppose that according to a solution $s = \{x(t_0), \cdots, x(t_{N-1})\}$, at time t_i, W_i^s is the amount of work completed, T_i^s is the current temperature, and E_i^s is the total energy consumption. Two solutions s and s' are *equivalent* if $W_i^s = W_i^{s'}, T_i^s = T_i^{s'}$, and $E_i^s = E_i^{s'}$ for each $i = 0, 1, \cdots, N-1$. We say that s *dominates* s' at time t_i if for each $j \leq i$, $W_i^s \geq W_i^{s'}, T_i^s \leq T_i^{s'}, E_i^s \leq E_i^{s'}$, and at least one of the equal signs does not hold. Intuitively, s' is dominated because it does not complete more workload, or ends up with a higher temperature, or consumes more energy.

Our dynamic programming based offline TALK minimization algorithm is motivated by the observation that an optimal solution cannot be dominated by any other solution for each t_i. It constructs an optimal solution as follows:

1. start with two candidate solutions $s = \{x(t_0) = 1, W_i^s = 0, T_i^s = T_{amb}, E_i^s = 0\}$ and $s' = \{x(t_0) = 0, W_i^{s'} = 0, T_i^{s'} = T_{amb}, E_i^{s'} = 0\}$.
2. calculate $x(t_1), W_i^s, T_i^s$ and E_i^s for each current candidate solution s that has $x(t_0) = 1$.
3. eliminate all the dominated candidate solutions at time t_1.
4. repeat steps 2 and 3 for $t_2, \cdots, t_N$.
5. select a candidate solution s that has $W_i^s \geq W$ and the smallest E_i^s as the optimal solution.

We mention that this offline algorithm gives the optimal solution to the TALK minimization problem. The algorithm's complexity is linear to N, the number of intervals that we partition $[0, D]$ and n, the largest number of candidate solutions at a step. However, in the worst case, this (in particular n) will still be exponential to N. Nevertheless, this offline algorithm provides the optimal solution with a given partition and we will use this to evaluate the performance of our online heuristic described next.

4.3 Online TALK Minimization

Input: $D, W, t_i, T(t_0)$
Output: $x(t_i)$
1. at time t_0
2. remaining workload W_r = W;
3. remaining time D_r = D;
4. **for** the starting time t_i of each interval $[t_i, t_{i+1})$
5. **if** ($W_r \geq D_r$) **return** cannot complete;
6. **if** (W_r == D_r)
7. **then** $x(t_j) = 1$, for $i \leq j \leq N$; **return**;
8. **if** ($\frac{W_r}{D_r - W_r} < \frac{T_{cur} - K_2}{K_1 - T_{cur}}$)
9. **then** $x(t_i) = 0$;
10. **else** $x(t_i) = 1$;
11. W_r = W_r - $(t_{i+1} - t_i)$;
12. D_r = D_r - $(t_{i+1} - t_i)$;
13. **if** ($W_r \leq 0$) $x(t_j) = 1$, for $i \leq j \leq N$; **return**;

Figure 3: Pseudo-code of the online TALK minimization heuristics.

Figure 3 illustrates the online heuristics for the TALK minimization problem. Motivated by the observation in the example and the leakage current' dependency on temperature as shown in equation (5), this heuristics seeks to avoid executing at high temperature to reduce leakage. It puts the system to $sleep$ mode to cool the system down whenever the task's workload is relatively light and the current temperature is high.

To measure how demanding a task is at a decision point t_i, we calculate the ratio η between the remaining workload W_r over the remaining idle time $D_r - W_r$, that is, $\eta = \frac{W_r}{D_r - W_r}$. This also measures the ratio of the time that the system will heat up over the time that the system can cool down before the deadline D. We further calculate the ratio θ between the time for the system temperature to rise one degree over the time to go down one degree from the current temperature $T_{cur} = T(x_i)$. It indicates in which direction, up or down, that the temperature can change more. From equations (2) and (3), we have

$$\theta = |\frac{dT_{fall}/dt}{dT_{rise}/dt}| = \frac{T_{cur} - K_2}{K_i - T_{cur}} \tag{15}$$

If $\eta < \theta$ (step 7), the system goes to $sleep$ mode because small η implies not heavy workload and large θ suggests high benefit in cooling the system down. Note that θ is small at low temperature T_{cur}. This encourage system to stay at *active* mode (step 10) unless the relative workload η is even lower. On the other hand, at high temperature, the large value of θ will put the system to the *sleep* mode (step 9) as long as the relative workload is not extremely demanding (that is, very large η).

Finally, we mention that this online TALK minimization technique requires little hardware and has very low run-time complexity from the following analysis. Steps 11 and 12 update the remaining workload and remaining time with a couple of subtraction; current temperature information can be obtained either from on-chip temperature sensor or by estimation [3, 14]; the condition statement in step 8 requires a couple of subtraction and two division. In fact, we track the values of D_r and W_r for the convenience of explanation. We can instead track $D_r - W_r$ and W_r to save several subtraction.

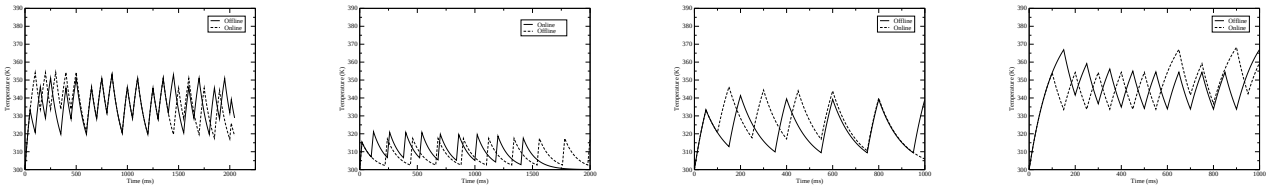
5 Simulation Results

In this section, we describe our simulation setup and report the simulation results. We are particularly interested in 1) how much leakage the offline and online TALK minimization algorithms can save; 2) how good is the online TALK heuristics comparing to the optimal offline TALK algorithm; and 3) how much can our TALK algorithms help on systems where existing DVS techniques have already been applied to minimize total energy.

5.1 Simulation Setup

We simulate the *TALK* algorithms on two types of systems: one employs a processor running at a single supply voltage 1.0V, which is the basic model for many small embedded applications; the other features a DVS-enabled processor that can run at voltages from 0.5V to 1.0V in a step of 0.5V. Both systems are implemented in a $65\mu m$ technology with a threshold voltage 0.295V. The fixed frequency in the first system is 500MHz; the highest and the lowest frequency in the second system ranges from 200MHz to 500MHz under different supply voltages. The processor in the system is implemented based on the Transmeta processor model [8]. It has a wakeup energy overhead $483\mu J$ for the processor to switch from the *sleep* mode to *active* mode. The wakeup delay overhead is $5ms$. During the *sleep* mode, the processor dissipates merely $50\mu W$ power. The thermal modeling of our system is based on [16]. We assume an ambient temperature K_2 = 300K; the maximal temperature K_1 from 363K to 388K for different supply voltages; and the thermal constant $R_{th}C_{th}$ is 105ms.

We employ both online and offline *TALK* algorithms to run eleven benchmarks. The first benchmark is an MPEG4 media encoding [17]; the second to the fourth benchmarks are taken from the Hartstone suite [4]; the fifth and sixth benchmarks are extracted from the ADSL standard's initialization sequences [19]; the rest



(a) ADSL2 (b) Airflow (c) CH2 (d) Bmk2

Figure 4: Temperature trace in two real-time benchmarks running offline and online algorithms.

five benchmarks are generated artificially based on the characteristics of real-life benchmarks. These benchmarks are representative of different system utilization ratio.

5.2 Leakage Reduction in a Single-Voltage System

In a single-voltage system without temperature awareness, the most simple and energy efficient way for the processor is to run the tasks up front and switch to the *sleep* mode to saver energy. Since the dynamic energy depends only on V_{dd} and is insensitive to temperature, they are the same in both the simple temperature awareless algorithm and our *TALK* algorithms. Our focus is on the leakage energy reduction by temperature aware algorithms and hence we report the leakage saving of *TALK* algorithms in Table 1.

The first column lists the benchmark names; the second and third column show the deadline and workload for each benchmark, and we assume the tasks' execution time is always equal to the workload. The leakage energy consumption in the system using the simple temperature awareless algorithm is shown in the fourth column. In the rest part of this table, we demonstrate the leakage savings of both online and offline *TALK* algorithms and the corresponding *wakeup* times of the processor: the first four columns show the results for interval size 100ms used in the *TALK* algorithm; the next four columns are results for interval size 50ms; and the last four columns are the results for interval size 20ms. The #wkp1 and #wkp2 columns represent the number of times the processor wakes up from *sleep* mode in online and offline algorithms, respectively.

We observe that as the interval size reduces, both online and offline algorithms can achieve larger energy saving. This is quite intuitive in that the finer granularity, potentially the more times the algorithm can put the system into *sleep* mode to save leakage energy. In fact, we see the number of times for the system to wake up is inversely proportional to the interval sizes. In practical, choosing the interval size is also restricted by the delay and energy overhead for the system to switch from *sleep* mode to *active* mode. The use of interval gives the freedom to design for different system settings.

In general the leakage saving by offline and online algorithms are very similar while the online algorithm have more wake-ups than the offline one. In a few cases of offline algorithm, we could not get the results because of the time constraints. Finally, the last row of the table shows the average results over the eleven benchmarks. Our *TALK* algorithms can achieve an average leakage energy saving between 20% to 30% percent with three different interval sizes. This saving is achieved by putting the processor to *sleep* 3 to 23 times on average. Note that the average number of wakeup times does not include that in benchmark MPEG4 because it is substantially larger than the others due to the tasks's much longer period and execution time. Thus, we think it's reasonable to treat this particular benchmark separately.

As we have explained in previous sections, the leakage energy saving of *TALK* algorithm is achieved by lowering the system temperature. Figure 4 illustrates the temperature trace when the system executes several benchmark tasks using online and offline *TALK* algorithms. We can see that the temperature curves in these two algorithms resemble each other. The peak temperature in benchmark ADSL2 is much lower (less than 325K) compared to 374K in the simple temperature awareless algorithm. This corresponds to the largest leakage saving in Table 1.

Table 2: Total energy consumption with No-DVS, traditional DVS, CS-DVS, and online and offline TALK algorithms.

Benchmark	Items	No-DVS	DVS	CS-DVS	Online	Offline
CH2	leakage	5.4	24%	45%	60%	61%
	dynamic	7.5	75%	66%	66%	66%
	total	12.9	54%	57%	63%	64%
	Vdd	1.0	0.5	0.6	0.6	0.6
CO	leakage	2.2	14%	38%	56%	57%
	dynamic	3.7	75%	65%	65%	65%
	total	5.9	52%	55%	61%	62%
	Vdd	1.0	0.5	0.6	0.6	0.6
Airflow	leakage	3.2	18%	39%	61%	63%
	dynamic	5.0	75%	65%	65%	65%
	total	8.2	53%	55%	63%	64%
	Vdd	1.0	0.5	0.6	0.6	0.6
ADSL1	leakage	5.1	30%	41%	46%	44%
	dynamic	7.1	70%	65%	65%	65%
	total	12.2	53%	57%	59%	59%
	Vdd	1.0	0.55	0.6	0.6	0.6
ADSL2	leakage	19.0	38%	51%	62%	61%
	dynamic	21.5	69%	67%	67%	67%
	total	40.5	54%	59%	64%	64%
	Vdd	1.0	0.55	0.6	0.6	0.6
Bmk1	leakage	7.8	13%	36%	54%	54%
	dynamic	9.9	75%	65%	65%	65%
	total	17.7	48%	52%	60%	60%
	Vdd	1.0	0.5	0.6	0.6	0.6

5.3 Total Energy Reduction in DVS-Enabled Systems

Next, we apply the *TALK* algorithms to a system that supports dynamic voltage scaling. We compare the total energy saving of *TALK* with the traditional DVS algorithm (DVS) and the leakage aware DVS algorithm (CS-DVS). Due to the quadratic dependency of dynamic energy on the supply voltage and the increasing of leakage energy over time, the DVS technique becomes inefficient only when the voltage is scaled down below certain point, which is defined as the critical voltage in [8]. In five out of eleven benchmarks, DVS algorithm will not be able to scale the voltage down below 0.7V, which is higher than the critical voltage for CS-DVS to be superior. Therefore, the traditional DVS algorithm that extends the execution over the entire period by running at a lowest possible voltage is still the most energy efficient way for those five benchmarks. And the CS-DVS will assign the same voltages as the traditional DVS. In these case, there will be no idle time for each task and hence *TALK* algorithms are not applicable in such cases.

We present the total energy saving using *TALK* algorithm on the rest six benchmarks in Table 2. For each benchmark, we display the leakage energy, dynamic energy, total energy and the voltage to run this benchmark in four rows. The third column lists the results for

Table 1: Leakage energy using traditional algorithm and temperature aware algorithms with different interval size.

Benchmark	D (ms)	W (ms)	leakage w/o t.a.	interval = 100ms				interval = 50ms				interval = 20ms			
				online	offline	#wkp1	#wkp2	online	offline	#wkp1	#wkp2	online	offline	#wkp1	#wkp2
MPEG4	60000	50000	1213.2	10%	n/a	413	n/a	12%	n/a	727	n/a	14%	n/a	2344	n/a
CH2	1000	300	5.4	25%	28%	3	2	32%	35%	6	5	35%	37%	15	12
CO	1000	150	2.2	16%	16%	2	1	23%	27%	3	2	30%	32%	8	7
airflow	2000	200	3.2	19%	20%	2	1	31%	33%	4	3	39%	41%	8	9
ADSL1	576	285	5.1	20%	20%	3	2	22%	23%	6	4	25%	25%	15	5
ADSL2	2048	864	19.0	33%	34%	9	8	37%	38%	18	17	39%	n/a	43	n/a
Bmk1	1000	400	7.8	26%	31%	4	3	32%	34%	8	7	34%	36%	20	15
Bmk2	1000	500	10.2	27%	29%	6	6	28%	31%	11	10	31%	32%	26	15
Bmk3	1000	600	12.6	24%	25%	4	4	26%	27%	11	8	27%	27%	29	12
Bmk4	1000	700	15.0	19%	19%	5	3	21%	21%	12	6	21%	22%	34	15
Bmk5	1000	800	17.5	12%	13%	4	2	15%	15%	11	4	15%	16%	37	10
Average				21%	23%	4.2	3.2	25%	28%	9	6.6	28%	30%	23.5	11.1

a system without using DVS and always runs at a reference voltage 1.0V. One can see the leakage energy in this case is the same as the one we have presented in Table 2. The fourth to seventh column show the energy saving over NO-DVS scheme achieved by DVS, CS-DVS, and our *TALK* online and offline algorithms respectively. for most benchmarks, the DVS algorithm will run the tasks with the lowest voltage 0.5V that results in 75% reduction in dynamic energy and 23% reduction in leakage energy on average. However, it executes the tasks longer and consumes 32% more leakage energy than CS-DVS algorithm on average. Our *TALK* algorithms will run at the same supply voltages determined by CS-DVS algorithm; however, it selectively puts the processor into *sleep* mode based on the chip temperature when executing the tasks.

We see that our online and offline algorithms can save 61.7% and 61.9% of total energy on average over the No-DVS scheme respectively; and this saving is 10.6% higher than the CS-DVS algorithms. Interestingly, we see that the leakage saving of *TALK* algorithms over CS-DVS is similar to the leakage saving in single-voltage system in Table 1.

6 Conclusions

In this paper we stress the importance of temperature consideration in designing energy efficient embedded systems. We study the temperature impact on leakage energy and propose an online and an offline temperature aware leakage minimization algorithms that adjust the processor modes at runtime based on the chip temperature. Our online algorithms can improve the energy saving by 18% over the traditional DVS algorithm and 11% over the leakage aware DVS algorithm.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems", in *Proc. of RTSS*, 2001, pp. 95-105.

[2] S. Borker, "Design Challenges of Technology Scaling", *IEEE Micro*, pp. 23-29, Aug. 1999.

[3] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors", in *Proc. of HPCA*,2001, pp.171-182.

[4] F. Golatowski, D.Timmermann, "Using Hartstone Uniprocessor Benchmark in a Real-Time System Course", *IEEE Real-Time Systems Education Workshop*, pp. 77-84, 1998.

[5] A.S. Grove, "Changing vectors of Moore's Law", *Keynote Speech, International Electron Devices Meeting*, Dec 2002.

[6] I. Hong, G. Qu, M. Potknojak, and M.B. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor", in *Proc. of RTSS*, 1998, pp. 178-187.

[7] M. Huang, J. Renau, S.M. Yoo, and T. Torellas, "A Framework for Dynamic Energy Efficiency and Temperature Management", *International Symposium on Microarchitecture*, pp. 202-213, 2000.

[8] R. Jejurikar, C. Pereira and R. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems", in *Proc. of DAC*, 2004, pp. 275-280.

[9] N.S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan, "Leakage Current: Moore's Law Meets Static Power", *IEEE Computer*, pp. 68-75, Dec. 2003.

[10] C.M. Krishna and Y.H. Lee, "Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems", in *Proc. of RTAS*, 2000, pp.156-165.

[11] Y-H. Lee, K.P. Reddy, and C.M. Krishna, "Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems", *Euromicro Conference on Real-Time Systems*, pp. 140-148, 2003.

[12] W. Liao, L. He, and K. Lepak, "Temperature-Aware Performance and Power Modeling", *Technical report UCLA Engineering. 04-250*, 2004.

[13] G. Quan and X. Hu, "Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors", in *Proc. of DATE*, 2002, pp. 782-787.

[14] H. Sanchez, B.Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip and J. Alvarez, "Thermal Management System for High Performance Power PC Micorprocessors", in *Proc. of COMPCON*,1997, pp. 325-330.

[15] K. Skadron, T. Abdelzaher, and M.R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management", in *Proc. of HPCA*, 2002, pp. 17-28.

[16] K. Skadron, M.R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation", *ACM Trans. Architecture and Code Optimization*, Vol. 1, pp. 94-125, Mar. 2004.

[17] D. Shin, J.Kim and S. Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications", *IEEE Design and Test of Computers*, pp. 20-30, Mar. 2001.

[18] J. Srinivasan and S.V. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications", *Int'l Conf. on Supercomputing*, 2003.

[19] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-Aware Runtime Scheduling for Embedded Multiprocessor SOCs", *IEEE Design and Test of Computers*, pp. 46-58, 2001.

[20] L. Yan, J. Luo and N.K. Jha, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems", in *Proc. of ICCAD*, 2003, pp. 30-38.

[21] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and Practical Limits of Dynamic Voltage Scaling", in *Proc. of DAC*, 20004, pp. 868-873.

[22] U.C.Berkeley BSIM3v3.1 SPICE MOS Device Models, 1997. http://www-device.EECS.Berkeley.edu/bsim3/.

[23] The International Technology Roadmap for Semiconductors, 2004. http://public.itrs.net.