Temperature-Aware Leakage Minimization Technique for Real-Time Systems [*]

Lin Yuan[1], Sean Leventhal[2] and Gang Qu[2]
[1]Synopsys Inc., Mountain View, CA 94043
[2]ECE Department and UMIACS, University of Maryland, College Park, MD 20742
yuanl@synopsys.com, {sleventhal,gangqu}@eng.umd.edu

Abstract

In this paper, we study the interdependency between leakage energy and chip temperature in real-time systems. We observe that the temperature variation on chip has a large impact on the system's leakage energy. By incorporating the temperature information, we propose an online temperature-aware leakage minimization algorithm for real-time systems. The basic idea is to run tasks when the system is cool and the workload is high, and put the system into sleep when it is hot and the workload is light. This online algorithm has low run-time complexity and improve the leakage energy saving by 34% on average in both real life and artificial benchmarks over traditional DVS approaches. Finally, our algorithm can be combined with existing dynamic voltage scaling methods to further improve the total energy efficiency.

1. INTRODUCTION

As technology scales down to the deep sub-micron (DSM) domain, the rapid increase of power density and leakage power in VLSI circuits have posed the immediate challenge of energy efficiency for real-time system designs, where energy source is often limited. Dynamic voltage scaling (DVS) technique is among the most effective in reducing dynamic energy in the system. To obtain the maximal dynamic energy reduction, DVS method aggressively slows down the execution of the task such that the task completes exactly at its deadline [1]. However, this comes with a longer execution time, which means more leakage energy will be consumed during the period. Due to the steep increase in leakage, several modified DVS techniques have been proposed to trade the dynamic energy reduction for more leakage saving in order to obtain the minimal total energy [4, 14]. In these techniques, a critical supply voltage/speed is defined in the DVS design space, such that if the supply voltage is scaled down below the critical voltage, the reduction in dynamic energy will be surpassed by the increase in leakage energy resulting more total energy. They propose to operate the system at a voltage higher than the critical one and put the system into *sleep* state when the task completes earlier than the deadline. Based on such critical voltage/speed, [4] propose a scheduling algorithm that schedules the tasks running back-to-back and leaves a long idle duration for the system to shutdown. None of these approaches consider the temperature influence in the leakage power. They assume a constant temperature. However, temperature in the system goes up rapidly with the increased power consumption.

High temperature not only affects the performance and reliability of the chip, but also the leakage current. For example, a 10^oC rise in temperature at 35^oC will result in currents going up by 126% due to the leakage [16]. This is because that leakage currents in MOS transistors have an exponential dependency on temperature based on the Berkeley BSIM3 model [15]. Unfortunately, most leakage reduction techniques do not consider this impact, resulting in poor estimation of total leakage power [16]. This interdependency between leakage and temperature also 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 avoid the thermal runaway situation, dynamic temperature management (DTM) techniques have been proposed to control peak temperature in microprocessors and large server systems [7, 8]. A trigger mechanism is designed in [7], where an on-chip sensor is used to measure the temperature; whenever the temperature exceeds a pre-determined threshold, a *throttling* technique is used to reduce the processor speed. Skadron et al. presented a microarchitectural-level thermal model that considers both the temporal and local thermal effects on chip [9]. A feedback control theory based approach was used to tune the system performance at a finer granularity [8]. In addition to general purpose processors, Srinivasan et al.[11] proposed a thermal management technique for multimedia applications. The runtime temperature is predicted based on the starting temperature of each task. However, they did not consider leakage power, which is a significant and temperature sensitive. A system-level leakage reduction technique is proposed in [3], where leakage current is described as a function of temperature.

The DTM techniques are very effective in controlling the peak temperature for general purpose processors; however, they often slow down the system for temperature reduction without considering the urgency of the executing tasks. Therefore, DTM techniques cannot be directly applied to

[*]This work is partially supported by grant CNS0615222 from the National Science Foundation.

real-time systems because they do not guarantee that tasks meet their deadlines, which is very critical for most real-times systems. In addition, how to set the temperature threshold values in DTM has a big impact on the energy consumption in the system, whereas DTM selects such thresholds for the solely purpose of controlling peak temperature, and its solution may not be good from the energy point of view.

The contributions of this paper are the following: we study the temperature and leakage interdependencies in real-time systems; we propose an online temperature-aware leakage minimization algorithm (TALK) that considers the urgency of each task while adjusting the run/sleep mode in the system; in addition, our algorithm also considers the wakeup overhead in time and energy to achieve an accurate energy reduction; finally, it requires little hardware support and has low run-time complexity.

2. PRELIMINARIES

System model: We study a real-time system where the tasks have been scheduled to meet their deadlines. That is, each task has been associated with a starting time, completion time, and worst-case execution time. The system has two operation modes: an *active* mode and a *sleep* mode. During the *active* mode, the tasks are being executed and the system dissipates both dynamic and leakage power; during the *sleep* mode, tasks are halted and and the system only consumes a small amount of power. When the system switches from *sleep* to *active* mode, additional time and energy are incurred, called the “wakeup” overhead.

Thermal model: The temperature variation on chip is modeled using the RC equivalence [8]. As the energy generated by the system is converted into heat, the system temperature rises, and it 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. (We assume a proper heat sink has been employed in the system and the thermal runaway will not occur). During this state, there will be almost no temperature variation and this state is generally called the *thermal equilibrium*. We denote the temperature at this state as K_1. Similarly, when the system cools down, it will also reach a temperature K_2 where it is as cool as the heat sink and no more heat can be removed. Normally, K_2 is equivalent or very close to the ambient temperature. We further define K_3 as the product of R_{th} and C_{th}, where R_{th} and C_{th} are the thermal resistance and thermal capacitance respectively. The rise and fall of temperature can be characterized using equations (1) and (2) below:

$$T_{rise}(t) = K_1 - (K_1 - T_{cur})e^{-\frac{t}{K_3}} \tag{1}$$

$$T_{fall}(t) = K_2 + (T_{cur} - K_2)e^{-\frac{t}{K_3}} \tag{2}$$

where $T_{rise}(t)$ and $T_{fall}(t)$ are functions of temperature over time t during the heating and cooling process respectively; T_{cur} is the transient temperature at the current time.

Energy model: We mainly consider the leakage energy in the system as it is emerging to be the dominant one and temperature dependent. Leakage power can be calculated as:

$$P_{leakage} = N_g \cdot I_{leakage} \cdot V_{dd} \tag{3}$$

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 [3]:

$$I_{leakage} = A \cdot T^2 \cdot e^{\frac{\alpha V_{dd}+\beta}{T}} + B \cdot e^{\gamma V_{dd}+\delta} \tag{4}$$

In this formula, $A, B, \alpha, \beta, \gamma$, and δ are empirical constants that can be found in [3]; 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 [5]. Therefore, we focus on the subthreshold leakage.

3. TEMPERATURE-AWARE LEAKAGE MINIMIZATION (TALK) ALGORITHM

3.1 Problem Formulation

We use a (D, W) pair to denote the life-period (completion time - starting time) and workload (worst-case execution time) of a single scheduled task, which is to be operated at voltage V_{dd}. We define $x(t) \in \{0, 1\}$ to be '1' if at time t the system is at the *active* mode or '0' if it is at the *sleep* mode. Our goal is to determine the function $x(t)$ in the interval $[0, D]$ such that the workload W can be completed and the total leakage energy expressed below is minimized:

$$\int_0^D (P_{active} \cdot x(t) + P_{sleep} \cdot (1 - x(t))dt \tag{5}$$

where P_{active} and P_{sleep} are the leakage power of the system at the active and sleep mode respectively. Note that the second term in equation (5) is a constant $P_{sleep} \cdot (D - W)$ where P_{sleep} is the power consumption when then the system is at the *sleep* mode. Hence we can formulate the temperature-aware leakage minimization problem as following:

Determining $x(t)$ such that $\int_0^D x(t)dt \geq W$ and $\int_0^D P_{leakage}(t) \cdot x(t)dt$ is minimized.

This is a known nonlinear feedback control problem. A fast and practical approach is to partition the interval $[0, D]$ into many sub-intervals $[t_i, t_{i+1}) \quad i = 0, 1, \cdots, N-1$ and $0 = t_0 < t_1 < \cdots < t_N = D$. When the sub-interval $[t_i, t_{i+1})$ is small, we can approximate the leakage power during that time period as invariant. 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, the original problem can be reduced to:

$$\sum_{i=1}^{N-1} x(t_i) \cdot (t_{i+1} - t_i - w_i \cdot t_{wakeup}) \geq W \tag{6}$$

$$\sum_{i=1}^{N-1} (P_{leakage}(t_i) \cdot x(t_i) \cdot (t_{i+1} - t_i) + w_i \cdot E_{wakeup}) \tag{7}$$

3.2 Online TALK Algorithm

We proposed an offline TALK algorithm in [13]. In this paper we will illustrates the online TALK algorithm in Figure 1. Motivated by the leakage dependency on temperature in equation (4), the basic idea of this heuristic is to avoid executing tasks at high temperature, where the leakage current rises sharply. Whenever the workload becomes relatively light, the algorithm postpones the execution and puts the system into the *sleep* mode to lower the temperature for

```
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 1: Pseudo-code of the online temperature-aware leakage minimization algorithm.

future tasks. On the other hand, when the workload is high, it will examine the current temperature and the remaining time, and finds the most energy efficient schedule to finish the task before the deadline.

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

From equations (1) and (2), we have

$$\theta = |\frac{dT_{fall}/dt}{dT_{rise}/dt}| = \frac{T_{cur} - K_2}{K_1 - T_{cur}} \tag{8}$$

If $\eta < \theta$ (step 8), 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 encourages the system to stay at the *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 into 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 algorithm 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 [7, 9]; 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.

4. SIMULATION RESULTS

4.1 Simulation Setup

We simulate the *TALK* algorithms on two types of systems: one with a processor running at a single supply voltage 1.0V, which is the basic model for many small embedded applications; the other with a DVS-enabled processor that can run at voltages from 0.5V to 1.0V in a step of 0.05V. 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 based on the Transmeta processor model [4]. It has a wakeup energy overhead $483\mu J$ and delay overhead $5ms$. The power dissipation at the *sleep* mode is $50\mu W$. The thermal model is from [9]. The ambient temperature is assumed to be 300K; for different supply voltages the maximal temperature at the thermal equilibrium is between 363K and 388K.

Eleven benchmarks are use to evaluate our TALK algorithm. The first benchmark is an MPEG4 media encoding [10]; the second to the fourth benchmarks are taken from the Hartstone suite [2]; the fifth and sixth benchmarks are obtained from the ADSL standard's initialization sequences [12]; the rest five benchmarks are generated artificially to represent different system utilization ratios.

4.2 Leakage Reduction in a Single-Voltage System

In a single-voltage system, we compare TALK with a simple algorithm that runs a task up front and switches to the *sleep* mode upon task completion, which we refer to as the naive approach. We report the leakage energy saving of *TALK* algorithms in Table 1.

Table 1: Leakage saving of the temperature aware algorithms with different interval size.

Benchmark	D(s)	W(s)	leakage naive	int 100ms		int 50ms		int 20ms	
				TALK	#w	TALK	#w	TALK	#w
MPEG4	60	50	1213.2	10%	413	12%	727	14%	2344
CH2	1	0.3	5.4	25%	3	32%	6	35%	15
CO	1	0.15	2.2	16%	2	23%	3	30%	8
airflow	2	0.2	3.2	19%	2	31%	4	39%	8
ADSL1	0.576	0.285	5.1	20%	3	22%	6	25%	15
ADSL2	2.048	0.864	19.0	33%	9	37%	18	39%	43
Bmk1	1	0.4	7.8	26%	4	32%	8	34%	20
Bmk2	1	0.5	10.2	27%	6	28%	11	31%	26
Bmk3	1	0.6	12.6	24%	4	26%	11	27%	29
Bmk4	1	0.7	15.0	19%	5	21%	12	21%	34
Bmk5	1	0.8	17.5	12%	4	15%	11	15%	37
Average				21%	4.2	25%	9	28%	23.5

The first column lists all the benchmarks; the second and third column show the deadline and workload for each application. We assume the execution time of a task is equal to the workload. The leakage energy consumption in systems running with the naive algorithm is shown in the fourth column. In the rest part of this table, we demonstrate the leakage savings of TALK algorithms and the number of times the processor *wakes up* (the #w columns), when the life-period D of each task is partitioned into intervals of length 100ms, 50ms and 20ms respectively.

We observe that as the interval size reduces, the TALK algorithms can achieve larger energy saving. This is because 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 wakeup delay and energy overhead.

Finally, the last row of the table shows the average results over eleven benchmarks. Note that the average number of

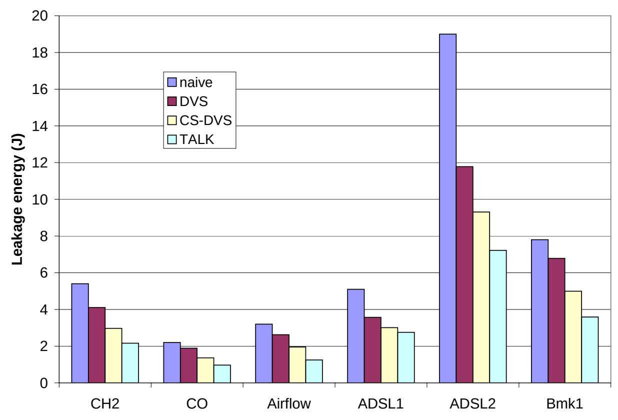

Figure 2: Leakage energy in the systems with naive approach, traditional DVS, CS-DVS and TALK algorithms.

wakeup times does not include that of benchmark MPEG4 because it is substantially larger than the others due to the tasks' much longer period and execution time. Thus, we think it is reasonable to treat this particular benchmark separately.

4.3 Leakage Reduction in DVS-Enabled Systems

Next, we simulate the *TALK* algorithms in a DVS-enabled system. Again, we first report the leakage energy of a naive algorithm and compare the leakage saving of *TALK* with the traditional DVS algorithm (DVS) and the leakage aware DVS algorithm (CS-DVS) proposed in [4]. 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 defined by CS-DVS. In another word, the traditional DVS algorithm that extends the execution over the entire period by having the system run at the lowest possible voltage is still the most energy efficient approach for those five benchmarks. Therefore, in those case, there will be no idle time for the system and *TALK* algorithms will not be beneficial.

We report our simulation results for the rest of six benchmarks in Figure 2. The naive algorithm always let the system run at the highest voltage and hence the run-time leakage power consumed is very high. The traditional DVS algorithms try to set the lowest possible speed to complete tasks. In this case, the supply voltages are between 0.5V to 0.55V. The CS-DVS algorithm favors leakage energy. Therefore, the supply voltages are set at 0.6V, allowing the tasks to finish before the deadline and put the system into the *sleep* mode to save leakage power. As a results, CS-DVS achieves 19% more leakage saving than DVS on average. In our TALK algorithm, we use the same supply voltages as those determined by CS-DVS; instead of having the system run up front and sleep, we choose to selectively puts the system into *sleep* mode based on the chip temperature when executing the tasks. The TALK can achieve on average 11% more leakage saving than CS-DVS algorithm.

Moreover, we also consider the total energy consumption including dynamic energy in the system. We report those results in [13].

5. CONCLUSIONS

In this paper we stress the importance of temperature consideration in designing energy efficient real-time systems. We study the temperature impact on leakage energy and propose a temperature aware leakage minimization algorithms that adjust the processor modes at runtime based on the chip temperature. Our online algorithms can improve the leakage energy saving by 34% over the traditional DVS algorithm and 15% over the leakage aware DVS algorithm.

6. REFERENCES

[1] H. Aydin et al., "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems", in *Proc. of RTSS*, 2001, pp. 95-105.

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

[3] L. He, W. Liao, and M.R. Stan, "System level leakage reduction considering the interdependence of temperature and leakage" in *Proc. of DAC*, 2004, pp. 12-17.

[4] 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.

[5] N.S. Kim et al., "Leakage Current: Moore's Law Meets Static Power", *IEEE Computer*, pp. 68-75, Dec. 2003.

[6] 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.

[7] H. Sanchez et al., "Thermal Management System for High Performance Power PC Microprocessors", in *Proc. of COMPCON*,1997, pp. 325-330.

[8] 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.

[9] K. Skadron et al., "Temperature-Aware Microarchitecture: Modeling and Implementation", *ACM Trans. Architecture and Code Optimization*, Vol. 1, pp. 94-125, Mar. 2004.

[10] 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.

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

[12] P. Yang et al., "Energy-Aware Runtime Scheduling for Embedded Multiprocessor SOCs", *IEEE Design and Test of Computers*, pp. 46-58, 2001.

[13] L. Yuan, S. Leventhal and G. Qu, "Temperature-Aware Leakage Minimization Techniques for Real-Time Systems", *UMIACS Technical Report, University of Maryland*, UMIACS-TR-2006-02, 2006.

[14] B. Zhai et al., "Theoretical and Practical Limits of Dynamic Voltage Scaling", in *Proc. of DAC*, 2004, pp. 868-873.

[15] Berkeley BSIM3 Device Models, *URL: http://www-device.EECS.Berkeley.edu/bsim3/*.

[16] Michael Santarini. "Thermal integrity: a must for low-power-IC digital design", *URL:http://www.edn.com/article/CA6255052.html?industryid=2813*, September 15, 2005.