A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation

CHI-HONG HWANG and ALLEN C.-H. WU
Tsing Hua University

This paper presents a system-level power management technique for energy saving of event-driven applications. We present a new predictive system-shutdown method to exploit sleep mode operations for energy saving. We use an exponential-average approach to predict the upcoming idle period. We introduce two mechanisms, prediction-miss correction and prewake-up, to improve the hit ratio and to reduce the delay overhead. Experiments on four different event-driven applications show that our proposed method achieves high hit ratios in a wide range of delay overheads, which results in a high degree of energy saving with low delay penalties.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Event-driven, predictive, power management, sleep mode, system shutdown

1. INTRODUCTION

With the advent of portable computing and high density VLSI circuits, power dissipation has emerged as a principle design consideration in VLSI designs. In the past few years, a handful of power estimation and minimization methods have been reported for achieving low power designs at circuit, layout, logic, behavioral, and architectural levels. Several excellent reviews of power estimation and minimization techniques are given by Pedram [1996]; Devadas and Malik [1996]; and Najm [1994; 1995].

This research was supported in part by a grant from the National Science Council of the R.O.C. under contracts NSC 86–2221–E–007–034 and 86–2221–E–007–021.
Authors' address: Department of Computer Science, Tsing Hua University, Hsinchu, Taiwan 30043, R.O.C.; email: chunghaw@cs.nthu.edu.tw.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2000 ACM 1084-4309/00/0400–0226 $5.00

Low power VLSI designs can be achieved at various design levels. In this study, we focus on utilizing a predictive system-shutdown technique for energy saving of event-driven applications. Power management techniques have been extensively applied to PC systems. For instance, the design of the PowerPC603 [Gary 1994] applied two types of power management schemes: static and dynamic. In static power management, the system defines several sleep modes with various levels of energy saving and delay overhead which can be controlled externally by software. In dynamic power management, the system will automatically detect the idle periods and disable the clocks on portions of the CPU. Apple's Mac PowerBooks [Farrahi et al. 1995] use a different approach that enters rest mode after 2 seconds of idle time. During rest mode, the processor is powered down but the I/O devices remain on. Typically, conventional shutdown approaches are carried out based on the rule of *go to sleep after the system has been idle for a predefined period of time*. However, this approach poses an obvious drawback—the system continues to consume power in that interval of idle time.

In a recent study, Srivastava et al. [1996] conducted an extensive analysis on different system-shutdown approaches. Their sample traces an event-driven application on an X-server reveals that it spends over 90% of its time in the blocked state, and the average time visiting the blocked state is less than one second. As a result, the conventional approach will fail to effectively achieve power reduction on this type of application. They have proposed a predictive system-shutdown technique for energy saving of event-driven applications. They first collected sample traces of on-off activity on an X-server. Then they proposed two predictive formulas based on the analysis of the sample traces. The first formula was obtained by using a general regression-analysis technique to correlate the length of the upcoming off period to the lengths of previous on and off periods. The second formula was obtained by observation of the on-off activity behavior. They observed that "the idle period following a long running period tends to be short." Based on this observation, they derived a formula that filters out the idle period, fulfilling the above condition; the system enters the sleep state otherwise. Based on the two formulas, they conducted a series of experiments on an X client application. The results demonstrated that predictive shutdown techniques can reduce the power consumption by a large factor compared to the conventional method.

In this paper we present a new predictive system-shutdown method for event-driven applications. We use a well-known exponential-average approach to predict the upcoming idle period. We introduce two mechanisms, prediction-miss correction and prewakeup, to improve the hit ratio and to reduce delay overhead. Experiments on four different event-driven applications are reported to demonstrate the effectiveness of our proposed method.

The paper is organized as follows: Section 2 describes the related work. Section 3 presents the proposed method. Section 4 presents the experimental results. Finally, Section 5 draws concluding remarks.

2. RELATED WORK

In the past several years, many techniques have been proposed for power minimization of IO devices and CPUs. In Douglis et al. [1994], an online algorithm was proposed to reduce power consumption by controlling disk spin-down. The results showed that using the online algorithm with a threshold of 10 seconds can reduce power consumption by 40% compared to the fixed threshold suggested by the disk manufacturers. Various CPU shutdown techniques have been proposed and evaluated for energy saving of CPUs [Weiser et al. 1994; Govil et al. 1995]. Many power minimization techniques at behavioral and architectural levels have also been proposed for low power VLSI and system designs. As summarized in Srivastava et al. [1996], there are three design strategies: (1) supply voltage reduction; (2) switching-activity reduction; and (3) activity-based system shutdown for reducing power consumption. Chandrakasan et al. [1992] first introduced the approach of minimizing the power by increasing the concurrency and lowering the supply voltage of a design. Chandrakasan et al. [1992] also proposed a number of transformation techniques, namely loop unrolling, pipelining, and retiming, to increase the concurrency of a design. Martin and Knight [1995] presented a software tool that is able to perform power minimization during the behavioral synthesis of ASICs using a combination of techniques, including lowering operating voltage, disabling the clock, using mixed supply voltages, and tradingoff design architectures. Raje and Sarrafzadeh [1995] proposed a variable voltage scheduling algorithm for power minimization. The main idea of this approach is to reduce power by lowering the supply voltage of some operation, yet meeting the throughput constraint of the system. Raghunathan and Jha [1994; 1995] proposed heuristics and ILP techniques that consider switching activity of operations for scheduling, clock selection, resource allocation, and binding targeted to low power datapath designs. Chang and Pedram [1995] proposed an optimal, polynomial-time algorithm to solve the register allocation and binding problem for power minimization. In recent studies, Monteiro et al. [1996] presented a scheduling algorithm to maximize the shutdown period of execution units of the design. Farrahi et al. [1995] presented a method to exploit sleep-mode operation on a set of memory segments so that the average power consumption can be reduced. Srivastava et al. [1996] proposed a predictive system shutdown technique that can effectively achieve energy saving for event-driven computation.

3. THE PROPOSED METHOD

In this section we present the proposed method. First, we describe how to use a system-shutdown technique for energy saving. Second, we present a formula for the prediction of idle periods. Third, we introduce a correction method for prediction misses. Fourth, we present a prewake-up method to improve the responsiveness of the system. Finally, we describe the control mechanism of the proposed method.

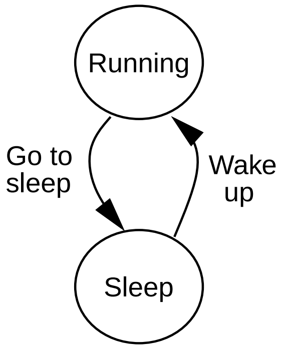

Fig. 1. A simple shutdown approach.

3.1 Energy Saving Using a System-Shutdown Technique

Figure 1 depicts a simple shutdown approach for event-driven applications. When the system detects an idle period, it will determine whether it should stay in the running state or enter the sleep state. If the system decides to enter the sleep state, it first performs a number of housekeeping procedures, such as backing-up data and system status. The system then stays in the energy-saving sleep state until an external wake-up signal occurs. When a wake-up signal occurs, the system will perform recovery procedures, such as data restoring, and then resume the running state. If the decision is not to enter the sleep state, the system stays in the running state as busy waiting. One of the crucial issues of this approach is "*whether or not to shutdown the system, and if so when?*" so that power dissipation of the system can be reduced. This issue is discussed as follows.

Let R be the running period, I the idle time period, E the delay overhead of entering the sleep state from the running state, S the sleeping time period, and W the delay overhead for resuming the running state from the sleep state (the wake-up process). P_R and P_S are the power consumption values of the system in the running and sleep states, respectively. P_{EW} is the average power-dissipation overhead of entering the sleep state from the running state and resuming the running state from the sleep state. Typically, $P_R \geq P_{EW} \geq P_S$. Finally, EG denotes the energy gain of the system.

Assume that the system will enter the sleep state whenever it detects an idle period. Figure 2(a) shows the first scenario in which the idle period is longer than the delay overhead of entering the sleep state from the running state (i.e., $I \geq E$). Since $I \geq E$, the system will successfully enter the sleep state and resume the running state when a wake-up signal occurs. Under this condition, the energy gain EG is $P_R \cdot I - (E + W) \cdot P_{EW} - P_S \cdot S$, while the delay penalty is W. Figure 2(b) shows the second scenario in which the idle period is shorter than the delay overhead of entering the sleep state from the running state (i.e., $I \leq E$). In this case, the system will never enter the sleep state and will suffer a long delay penalty ($delay =$

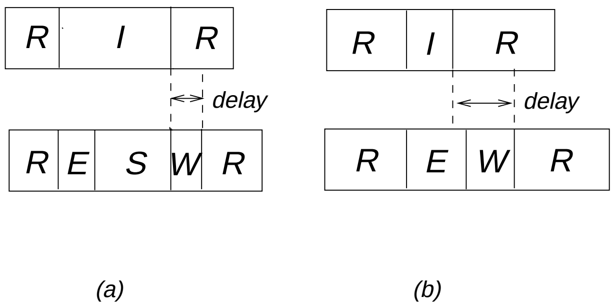

Fig. 2. Two possible scenarios when applying a simple system-shutdown scheme: (a) $I \geq E$, (b) $I \leq E$.

$W + (E - I)$) and a negative energy gain when $P_{EW} > I \cdot P_R - (E + W)$. Hence, in order to achieve energy saving, the idle period must be longer than the delay overhead for entering the sleep state from the running state (i.e., $I \geq E$).

Let us further analyze the minimum required idle period for achieving a positive energy gain, as follows:

$$I \geq E, \tag{1}$$

$$EG = I \cdot P_R - (E + W) \cdot P_{EW} - P_S \cdot S; \tag{2}$$

$$= I \cdot P_R - (E + W) \cdot P_{EW} - P_S \cdot (I - E); \tag{3}$$

$$= I \cdot (P_R - P_S) - E \cdot (P_{EW} - P_S) - W \cdot P_{EW} > 0, \tag{4}$$

$$=> I > \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}, \tag{5}$$

$$=> S_{th} = \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}. \tag{6}$$

The above derivations indicate that Eqs. (1) and (6) are the necessary conditions under which the idle period will achieve energy savings. We define S_{th} as the threshold idle period that results in energy saving.

3.2 Predicting Idle Periods

The analysis in the previous section indicates that predicting the idle period I is vital for an effective power management mechanism. Similar problems have occurred in the CPU scheduling of operating systems. In the CPU scheduling problem, operating systems need to predict the length of the next CPU burst in order to make appropriate process scheduling. The *exponential-average* approach [Peterson and Silberschatz 1986] is one of the commonly used methods for the prediction of the idle period. Using this method, the next CPU burst is predicted as an accumulative average of the measured lengths of previous CPU bursts. Similarly, we can use the *exponential-average* approach to predict the next idle period by the accumulative average of the previous idle periods. The recursive prediction formula is shown below:

$$I_{n+1} = a \cdot i_n + (1 - a) \cdot I_n, \tag{7}$$

where I_{n+1} is the new predicted value, I_n is the last predicted value, i_n is the latest idle period, and a is a constant attenuation factor in the range between 0 to 1. In this formula, I_n is the inertia and i_n is the force to push the predicted idle period toward the actual idle period. We can use Eq. (7) to predict the upcoming idle period, which is a function of the latest idle period and the previous predicted value. The parameter a controls the relative weight of recent and past history in the prediction. If $a = 0$, then $I_{n+1} = I_n$. In other words, the recent history has no effect. On the other hand, if $a = 1$, then $I_{n+1} = i_n$. In this case the prediction only takes into account the most recent idle period, but ignores the previous predictions. In our implementation, a is set to be $1/2$ so that the recent history and past history are equally weighted. We can expand Eq. (3) as below:

$$\begin{aligned} I_{n+1} = {} & a \cdot i_n + a(1 - a)i_{n-1} + a(1 - a)^2 i_{n-2} + \cdots \\ & + a(1 - a)^n i_0 + (1 - a)^{n+1} I_0. \end{aligned} \tag{8}$$

Equation (8) indicates that the predicted idle period is the weighted average of previous idle periods. Early idle periods have less weight, as specified by the exponential attenuation factors.

3.3 Correction of Prediction Misses

The proposed prediction formula (Eq. (7)) can effectively predict idle periods in most cases, except the occurrence of impulse-like idle periods (a sudden, very long idle period $I3$ occurs after continuous, nearly uniform idle periods) during the prediction process, as shown in Figure 3(a). For example, the user works on the system for a while and then goes to answer a telephone, resulting in the system idling for a long period of time. When such an impulse-like idle period occurs, the prediction of the upcoming idle period $I3$ and the following one $I4$ will not be accurate. The reasons are as follows: recall that our proposed prediction formula (Eq. (7)) predicts the upcoming idle period by the accumulative average of the previous idle periods. When a very long idle period occurs after continuous, nearly uniform idle periods, the predicted value of this long idle period is often much lower than the actual idle period ($I3 > I3'$). This underestimation is undesirable for energy saving, especially when the predicted value is lower than S_{th}. In this case the system will stay in the running state instead of

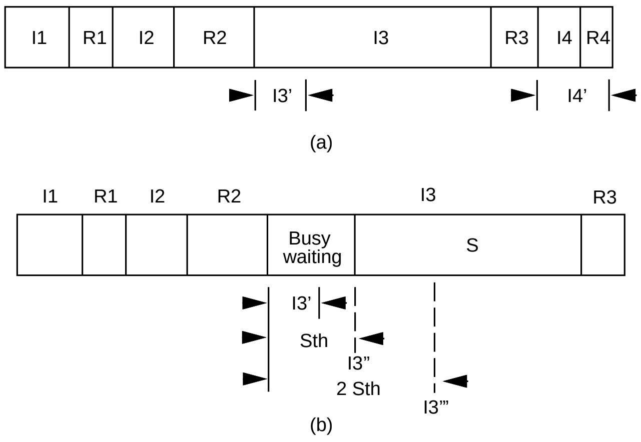

Fig. 3. Correction of prediction miss: (a) an impulse-like idle period; (b) correction using a watchdog scheme.

entering the sleep state, which results in a large amount of unnecessary power consumption. On the other hand, when predicting the idle period $I4$, which followed a long idle period, our proposed formula tends to overestimate the duration of the idle period ($I4 < I4'$). It is also undesirable because the system may falsely enter the sleep state and suffer unnecessary power consumption and delays.

To alleviate the first problem, we use a watchdog scheme to periodically monitor the current idle period. When an idle period occurs, the system predicts the duration of the idle period. If the predicted value is lower than S_{th}, the system stays in the busy waiting state and starts up a timer to trace the actual idle period. The system then performs a new prediction every S_{th} time to determine whether the system should enter the sleep state. For example, in Figure 3(b), a long idle period $I3$ occurs by the end of a running state $R2$. If the predicted idle period $I3'$ is lower than S_{th}, the system stays in the busy waiting state. After an S_{th} time period, the system performs the prediction again, resulting in $I3''$. If the predicted value is larger than S_{th}, then it enters the sleep state. Otherwise, it will perform idle period prediction after another S_{th} time. To resolve the second problem, we add a saturation condition to Eq. (3), as below:

$$\begin{gathered} if\,(ai_n + (1-a)I_n > cI_n) \\ I_{n+1} = cI_n, \end{gathered} \tag{9}$$

where c is a constant. Under the saturation condition, the growing rate of I is limited to c times per update.

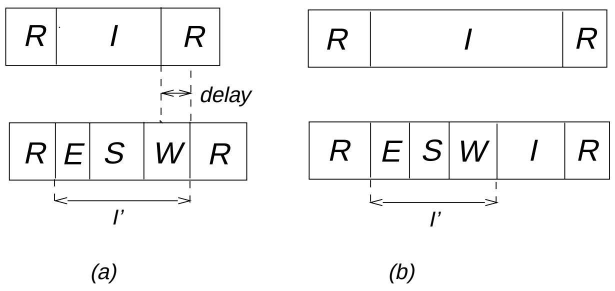

Fig. 4. Examples of prewake-up model: (a) $I' > I$ *and* $D \leq W$, (b) $I' < I$.

3.4 Prewake-up

As described in the previous section, when the system resumes the running state from the sleep state (i.e., system wake-up), the system needs to perform some recovery procedures. In other words, the system suffers a delay penalty W by restoring the system status. This delay penalty may have a great impact on system responsiveness in some cases, especially when W is too long to be neglected. One way to resolve this problem is to prewake-up the system before the arrival of the next wake-up signal. This can be accomplished by predicting the occurrence of the next wake-up signal.

Let I be the actual idle period, I' the predicted idle period, and $D = |I' - I|$ the error of the prediction. Figure 3 depicts two possible scenarios when applying the prewake-up scheme. In the first scenario, we overestimate the predicted idle period by D (i.e., $I' > I$) and $D \leq W$. In this case the system will wake up $W - D$ time ahead of the next wake-up signal, as shown in Figure 3(a). Thus, the delay penalty is D, which is shorter than the original delay penalty W. In addition, the energy gain is $EG = I \cdot (P_R - P_S) - (E + W) \cdot (P_{EW} - P_S) - D \cdot P_S$. On the other hand, if $I' > I$ and $D \geq W$, the system will be woken up by the original wake-up signal. In this case, the prewake-up has no effect on the reduction of the delay penalty. Figure 3(b) shows the second scenario in which $I' < I$. In this case the delay penalty is zero, but the energy gain is reduced to $EG = (I - D) \cdot (P_R - P_S) - (E + W) \cdot (P_{EW} - P_S)$. In other words, when the predicted idle period is less than the actual idle period, there will be no responsiveness delay. However, the energy saving will not be as effective as the original one.

3.5 Control Mechanism of the Proposed Method

Figure 5 shows the finite state machine of the proposed method, which consists of three states: *running*, *correct*, and *sleep*. Initially, the system is in the running state. When an idle period occurs, the system predicts the duration I' of the upcoming idle period. If the predicted value is lower than the threshold value S_{th}, then the system resets the timer c and stays in the

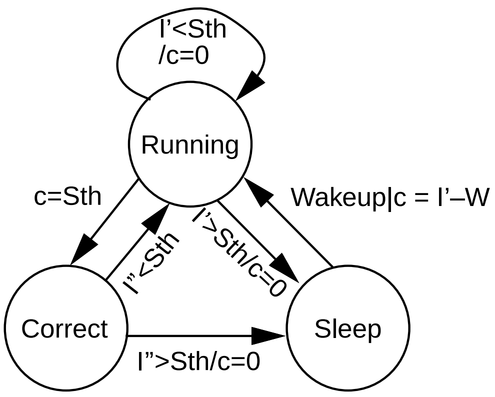

Fig. 5. Finite state machine of the proposed method.

running state as busy waiting. Otherwise, the system enters the sleep state. During the busy waiting, the system monitors the timer and enters the correct state every S_{th} of time. In the correct state, the system repredicts the duration of the idle period. If the predicted value I'' is lower than S_{th}, then the system resumes the running state as busy waiting. Otherwise, it enters the sleep state. In the sleep state, the system will return to the running state when a wake-up signal arrives. If the pre-wake-up scheme is applied to the system, then the system will return to the running state either when a wake-up signal arrives or the system has been shutdown for a predicted idle period ($c = I'$).

4. EXPERIMENTS

We conducted experiments on four event-driven applications: X-server, Netscape, Telnet, and Tin. The experiments were conducted on a SPARC 10 workstation running the SunOS 4.1.3 operating system. Figures 6, 7, 8, and 9 show the sample traces of the applications of X-server, Netscape, Telnet, and Tin, respectively. The black impulses represent the running state and the blanks represent the idle states. In our implementation, we set the constants $a = 0.5$ (Eq. (7)) and $c = 2$ (Eq. (9)). We assume $E = W = 1/2T_{cost}$, where T_{cost} is the delay overhead. First, we determine the threshold idle period S_{th} (Eq. (6)). Recall that

$$S_{th} \geq \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}. \tag{10}$$

We assume that the average power dissipation of entering and resuming from the sleep state P_{EW} is the same as power consumed in the running state; i.e., $P_{EW} = P_R$. In addition, according to the data sheet of the PowerPC [Gary 1994], we define the power dissipation in the sleep state to be approximately 5% of the power dissipation in the running state; i.e., $P_S = 0.05 \cdot P_R$. Hence,

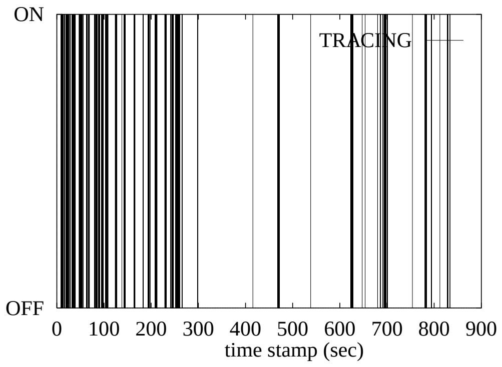

Fig. 6. Sample traces of the X-server.

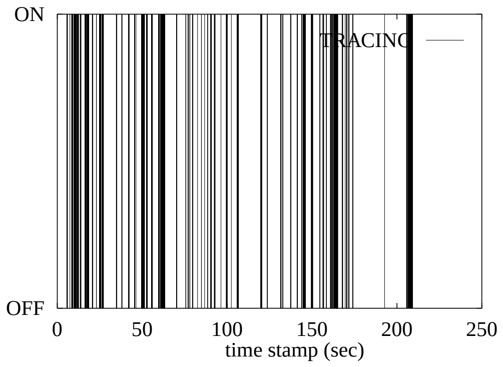

Fig. 7. Sample traces of Netscape.

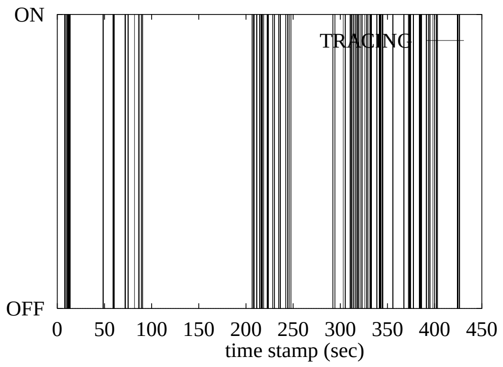

Fig. 8. Sample traces of Tin.

$$S_{th} \approx \frac{1/2 \cdot T_{cost} \cdot (P_R - 0.05 \cdot P_R) + 1/2 \cdot T_{cost} \cdot P_R}{(P_R - 0.05 \cdot P_R)}, \tag{11}$$

$$= \frac{0.5*1.95}{0.95} \cdot T_{cost} = 1.026 \cdot T_{cost}, \tag{12}$$

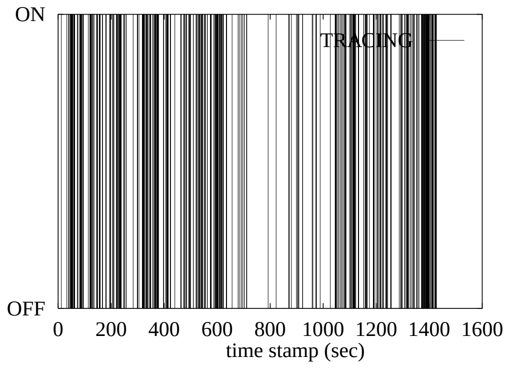

Fig. 9. Sample traces of Telnet.

A reasonable value of S_{th} is larger than $1.026 \cdot T_{cost}$. In our implementation we set $S_{th} = 1.5 \cdot T_{cost}$.

We also define two quality measures: Delay-Overhead (DO) and Energy-Saving (ES). Delay overhead is the ratio between the total elapsed time after applying the system-shutdown scheme (T'_t) and the original elapsed time (T_t) as

$$DO = \frac{T'_t - T_t}{T_t} \times 100\%. \tag{13}$$

Let T_s be the elapsed time of the sleep state. The energy saving is the ratio between the total power dissipation before and after applying the system-shutdown scheme:

$$ES = \frac{P_R \cdot T_t}{P_R \cdot T'_t - T_s \cdot (P_R - P_S)} \tag{14}$$

$$\approx \frac{P_R \cdot T_t}{P_R \cdot T'_t - T_s(P_R - 0.05 \cdot P_R)} \tag{15}$$

$$= \frac{T_t}{T'_t - 0.95T_s}. \tag{16}$$

In the first experiment we tested the hit ratio of our proposed method. We defined a *hit* in two ways: the system enters the sleep state and results in energy saving, or the system does not enter the sleep state if it will not result in energy saving. Figure 10 (a–d) shows the hit ratios of the four applications, i.e., X-server, Netscape, Tin, and Telnet, respectively. T_{cost} is the delay penalty for shutting down the system. We have compared the hit ratios using three predictive methods: (1) *on−threshold*(∞)*based* [Srivastava et al. 1996]; (2) our proposed method without miss correction; and (3)

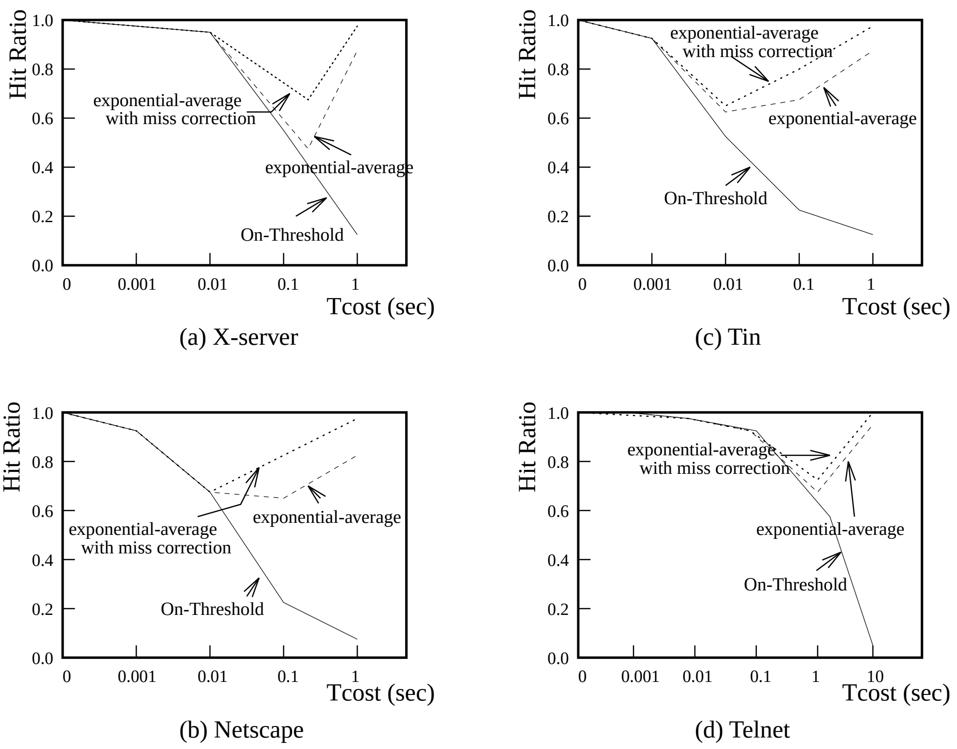

Fig. 10. Comparisons of hit ratios.

the proposed method with miss correction. The results show that when the T_{cost} is low (e.g., 0.001 second for Tin, 0.01 second for the X-server and Netscape, and 0.1 second for Telnet), both the $on-threshold(\infty)based$ and our method produced the same hit ratios. When the T_{cost} increases, the hit ratio of the $on-threshold(\infty)based$ method decreases rapidly. In contrast, our proposed method consistently gives high hit ratios. The results also show that the correction method improves the hit ratio by an average of 20%, compared to that without applying the correction method.

In the second experiment we compared the energy saving and delay overhead using five system-shutdown methods: (1) PM_{2sec}, (2) PM_{ThX}, (3) $PM_{Th\infty}$, (4) PM_{I}, and (5) PM_{II}. The first is a method used in the Apple PC system [Apple Computer Inc. 1992], in which the system will enter the sleep state whenever a 2-second idle period is detected. The second and third methods were proposed by Srivastava et. al. [1996]. To determine a reasonable threshold value for the second method, we first analyzed the relationship between on-off periods of the sample traces. The results show that we obtained the similar L-shaped on-off periods relationship observed in Srivastava et al. [1996] for the four applications. From the observations, we set the $T_{threshold}$ value to 50ms for T_{cost} in the range of 0-0.1 sec and 2ms for T_{cost} of 1 sec, 100ms for T_{cost} = 0-0.1 sec, and 1ms for T_{cost} = 1 sec, 200ms for T_{cost} = 0-0.1 sec, and 1ms for $T_{cost} = 1$ sec, and 40ms for T_{cost} = 0-0.1 sec, and 4ms for $T_{cost} = 1$ sec, for the X-server, Netscape, Tin, and Telnet, respectively. The fourth method is our proposed method without applying the prewake-up scheme. The final method is our proposed method, including the prewake-up scheme.

Table I shows the results, from which the following observations can be made. First, the PM_{2sec} method performs well when $T_{cost} = 1s$, but poorly when T_{cost} is in the range of 0 to 0.1 sec. Second, when T_{cost} is in the range of 0 to 0.1 sec, the $PM_{Th\infty}$ achieves the best energy saving. However, when $T_{cost} = 1$, energy saving drops sharply and delay overhead increases rapidly. For example, for the X-server application, it obtains 1.53 times energy saving with 48% delay overhead. For the Netscape and Tin applications, it actually consumes more power (60% and 20%, respectively) and the delay also increases by two to three times. This is due to the power and delay overhead caused by the large number of hit misses. Similarly, the PM_{Thx} method performs well when T_{cost} is in the range of 0 to 0.1 sec, but poorly when $T_{cost} = 1s$. On the other hand, our proposed method PM_I performs well in the entire range of $T_{cost} = 0$ to 1 sec. Finally, when applying the prewake-up technique, the delay overhead can be reduced by sacrificing some energy saving (PM_{II} v.s. PM_I). This is important for some time-critical applications. In our approach, we are able to tradeoff delay overhead and energy saving by enabling and disabling the prewake-up function.

5. CONCLUSIONS

In this paper we have presented a predictive shutdown method for event-driven computation. We have conducted experiments on four event-driven applications and compared our approach to two other system shutdown methods. The results show that when the shutdown overhead is high (e.g., $T_{cost} \geq 1$ sec), the conventional method, such as that used by the Apple PowerPC, can achieve a reasonable energy saving. However, this method does not achieve an effective energy saving when the shutdown overhead is low (e.g., $T_{cost} < 1$ sec). On the other hand, when the shutdown overhead is low (e.g., $T_{cost} < 1$ sec), a more aggressive predictive shutdown method [Srivastava et al. 1996] can achieve a higher degree of energy saving compared to the conventional method. However, this method does not perform well when the shutdown overhead is high. In addition, because the predictive formula of this approach has to be derived directly from the sample traces of the target application, we have to analyze the sample traces for each application in order to determine the threshold value. On the other hand, our proposed method depends solely on the history of the previous idle periods and the power dissipation of the target system (e.g., the average power consumption in the sleep and running states), which is independent of the target applications. Furthermore, the results show that our proposed method achieves high hit ratios for a wide range of shutdown

Table I. Comparisons of Various System Shutdown Methods

T_{cost}	PM_{2sec}	PM_{ThX}	PM_{ThInf}	PM_{I}	PM_{II}
		X-server(Delay-Overhead(%)/Energy-Saving)			
0	0/3.75	0/16.8	0/18.73	0/18.70	0/18.70
1ms	0/3.75	.03/16.63	0.3/18.53	0.3/18.50	0.3/17.99
10ms	.02/3.75	.3/15.29	.3/16.87	.03/16.80	.029/13.70
0.1s	.24/3.68	3.28/8.45	3.3/8.89	2.18/8.38	1.7/5.14
1s	2.58/3.18	17/1.14	48/1.53	8.3/3.29	1.8/1.92
		Netscape(Delay-Overhead(%)/Energy-Saving)			
0	0/1.60	0/4.77	0/8.92	0/8.82	0/8.82
1ms	0/1.60	.11/4.72	.12/8.73	.12/8.63	.11/8.13
10ms	.07/1.60	1.33/4.29	1.38/7.34	1.14/7.18	.97/5.28
0.1s	.73/1.56	16.8/2.26	17.4/2.82	2.63/3.63	1.8/2.10
1s	8.11/1.30	13.34/0.88	321/0.39	10.44/1.37	.87/1.06
		Tin(Delay-Overhead(%)/Energy-Saving)			
0	0/2.86	0/16.32	0/16.33	0/16.33	0/16.33
1ms	0/2.86	.06/16.02	.06/16.03	.06/16.02	.05/15.04
10ms	.03/2.85	.73/13.70	.73/13.71	.47/13.66	.38/9.92
0.1s	.35/2.80	9.48/5.59	9.5/5.58	1.71/7.43	1.32/3.74
1s	3.99/2.38	3.56/0.97	207/0.80	5.77/2.71	.92/1.54
		Telnet(Delay-Overhead(%)/Energy-Saving)			
0	0/2.63	0/18.18	0/19.19	0/15.15	0/15.15
1ms	0/2.63	.01/18.08	.01/19.09	.01/15.08	.01/14.96
10ms	.05/2.62	.13/17.32	.13/17.32	.13/18.23	.13/13.49
0.1s	.57/2.55	1.43/12.16	1.43/12.60	1.42/10.72	1.34/7.11
1s	6.04/2.04	7.02/1.17	15.99/3.07	11.31/2.46	8.54/1.79

overheads, which results in a high degree of energy saving with low delay penalties.

In this study we have shown that the proposed method is effective for energy saving of event-driven computation. Our proposed method can be applied to manage the shutdown of CPUs as well as peripherals, such as disks. Using the proposed method, one problem is how to select the value of the constant attenuation factor a (Eq. (7)), so that high-accuracy predictions can be achieved. Consider that if a small value of a is selected the predicator is not very agile, and if a large value of a is selected the predicator will be very sensitive to a minor spike. Both cases will cause the predicator to either over-estimate or under-estimate the upcoming idle period. It will result in either the system falsely entering the sleep state or preventing the system from entering the sleep state. Both cases will not be effective in energy saving. We set $a = 0.5$ so that the recent history and past history are equally weighted. When $a = 0.5$, the predicator performs well on uniformly distributed idle-period occurrences. We then applied the correction of the prediction miss strategy to improve the prediction accuracy for spike-like idle periods. The prediction accuracy may be improved if a dynamic a value can be incorporated in the proposed method, so that a different idle period prediction can be performed during spikes. In addition, our current proposed method focuses on a single sleep mode. Future work will extend our method to accommodate multilevel sleep modes.

REFERENCES

Apple Computer, Inc. 1992. *Technical Introduction to the Macintosh Family*. Addison-Wesley, Reading, MA.

Chandrakasan, A. P., Potkonjak, M., Rabaey, J., and Brodersen, R. W. 1992. HYPER-LP: A system for power minimization using architectural transformations. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '92, Santa Clara, CA, Nov. 8–12), L. Trevillyan, Ed. IEEE Computer Society Press, Los Alamitos, CA, 300–303.

Chandrakasan, A. P., Sheng, S., and Brodersen, R. W. 1992. Low-power CMOS digital design. *IEEE J. Solid-State Circuits 27*, 4 (Apr. 1992), 473–484.

Chang, J. -M. and Pedram, M. 1995. Lower power register allocation and binding. In *Proceedings of the 32nd Conference on Design Automation*, 29–35.

Devadas, S. and Malik, S. 1995. A survey of optimization techniques targeting low power VLSI circuits. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16), B. T. Preas, Ed. ACM Press, New York, NY, 242–247.

Douglis, F., Krishnan, P., and Marsh, B. 1994. Thwarting the power-hungry disk. In *Proceedings of the 1994 USENIX Winter Technical Conference on USENIX*, USENIX Assoc., Berkeley, CA.

Farrahi, A. H., Téllez, G. E., and Sarrafzadeh, M. 1995. Memory segmentation to exploit sleep mode operation. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16), B. T. Preas, Ed. ACM Press, New York, NY, 36–41. http:www.getridofme.com

Gary, S. 1994. PowerPC: A microprocessor for portable computers. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 10-12), IEEE Computer Society Press, Los Alamitos, CA, 14–23.

Govil, K., Chan, E., and Wasserman, H. 1995. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking* (MOBICOM '95, Berkeley, CA, Nov. 13–15), B. Awerbuch and D. Duchamp, Eds. ACM Press, New York, NY, 13–25.

Martin, R. S. and Knight, J. P. 1995. Power-profiler: Optimizing ASICs power consumption at the behavioral level. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16), B. T. Preas, Ed. ACM Press, New York, NY, 42–47.

Monteiro, J., Devadas, S., Ashar, P., and Mauskar, A. 1996. Scheduling techniques to enable power management. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 349–352.

Najm, F. N. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE Trans. Very Large Scale Integr. Syst. 2*, 4 (Dec. 1994), 446–455.

Najm, F. N. 1995. Power estimation techniques for integrated circuits. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 492–499.

Pedram, M. 1996. Power minimization in IC design: Principles and applications. *ACM Trans. Des. Autom. Electron. Syst. 1*, 1, 3–56.

Peterson, J. L. and Silberschatz, A. 1985. *Operating System Concepts*. 2nd ed. Addison-Wesley Series in Computer Science. Addison-Wesley Longman Publ. Co., Inc., Reading, MA.

Raghunathan, A. and Jha, N. K. 1994. Behavioral synthesis for low power. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 10-12), IEEE Computer Society Press, Los Alamitos, CA, 318–322.

Raghunathan, A. and Jha, N. K. 1995. An iterative improvement algorithm for low power data path synthesis. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 597–602.

RAJE, S. AND SARRAFZADEH, M. 1995. Variable voltage scheduling. In *Proceedings of the 1995 International Symposium on Low Power Design* (ISLPD-95, Dana Point, CA, Apr. 23–26), M. Pedram, R. Brodersen, and K. Keutzer, Eds. ACM Press, New York, NY, 9–14.

SRIVASTAVA, M. B., CHANDRAKASAN, A. P., AND BRODERSEN, R. W. 1996. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. Very Large Scale Integr. Syst. 4*, 1, 42–55.

WEISER, M., DEMERS, A., WELCH, B., AND SHENKER, S. 1994. Scheduling for reduced CPU energy. In *Proceedings of the (OSDI) Conference on Operating System Design and Implementation* (Nov.)

Received: August 1997; revised: December 1997; accepted: August 1998