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

Chi-Hong Hwang and Allen C.-H. Wu
Department of Computer Science, Tsing Hua University
Hsinchu, Taiwan, 300, Republic of China
{chunghaw@cs.nthu.edu.tw}

Abstract

This paper presents a system-level power management technique for power saving of event-driven applications. We present a new predictive system-shutdown method to exploit sleep mode operations for power saving. We use an exponential-average approach to predict the upcoming idle period. We introduce two mechanisms, prediction-miss correction and pre-wakeup, 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 power saving with low delay penalties.

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 [1], Devadas and Malik [2], and Najm [3, 4].

Low power VLSI designs can be achieved at various design levels. In this study, we focus on utilizing a predictive system-shutdown technique for power saving of event-driven applications. Power management techniques have been extensively applied to PC systems. For instance, the design of the PowerPC603 [5] applied two types of power management schemes: static and dynamic. In static power management, the system defines several sleep modes with various levels of power 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 [6] use a different approach which 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.

†This work was supported in part by the National Science Council of R.O.C. under Grant NSC 85-2221-E-007-034 and NSC 86-2221-E-007-021.

In a recent study, Srivastava et al. [7] conducted an extensive analysis on different system-shutdown approaches. Their sample traces on an X-server, an event-driven application, showed 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 the previous on and off periods. The second formula was obtained by the 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 which filters out the idle period fulfilling the above condition and 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. However, one drawback of this approach is that the predictive formula is directly derived from the sample traces of a specific application. For different applications, different predictive formulas are needed in order to make accurate predictions.

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 pre-wakeup, to improve the hit ratio and to reduce the delay overhead. Experiments on four different event-driven applications are reported to demonstrate the effectiveness of our proposed method.

2 The proposed method

2.1 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

0-89791-993-9/97 $10.00 © 1997 IEEE

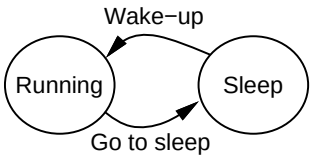

Figure 1: A simple shutdown approach.

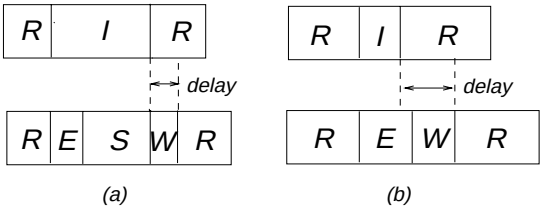

Figure 2: Two possible scenarios when applying a simple system-shutdown scheme:(a) $I \geq E$,(b) $I \leq E$.

decides to enter the sleep state, the system 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 wakeup signal occurs. When a wakeup 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 will be discussed as follows.

Let I be 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 ($delay = W + (E - I)$) and a negative energy gain ($EG = I \cdot P_R - (E + W) \cdot P_{EW}$). 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.

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

$$\begin{aligned} EG &= I \cdot P_R - (E+W) \cdot P_{EW} - P_S \cdot S; && (2)\\ &= I \cdot P_R - (E+W) \cdot P_{EW} - P_S \cdot (I - E); && (3)\\ &= I \cdot (P_R - P_S) - E \cdot (P_{EW} - P_S) - W \cdot P_{EW}; && (4)\\ &> 0, && (5)\\ \Rightarrow \quad I &> \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}, && (6)\\ \Rightarrow \quad S_{th} &= \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}. && (7) \end{aligned}$$

The above derivations indicate that Eqns. 1 and 7 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 of the system.

2.2 Prediction of idle periods

The analysis in the previous section indicates that predicting the idle period I is vital for an effective power management mechanism. However, the distribution and variation of I may be strongly dependent on the user behavior, target applications, working frequency, and operating system.

In our approach, we adapt the *exponential-average* approach [8] used in the CPU scheduling problem for the prediction of the idle period. In the CPU scheduling problem, operating systems need to predict the length of the next CPU burst in order to make appropriate process scheduling. In general, the next CPU burst is predicted as an accumulative average of the measured lengths of previous CPU bursts. Similarly, we can predict the next idle period by the accumulative average of the previous idle periods. The recursive prediction formula is shown below.

$$I_{n+1} = a \cdot i_n + (1-a) \cdot I_n, \tag{8}$$

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 Eqn. 8 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 Eqn. 8 as below.

$$I_{n+1} = a \cdot i_n + a(1-a) i_{n-1} + \cdots + a(1-a)^n i_0 + (1-a)^{n+1} I_0. \tag{9}$$

Eqn. 9 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.

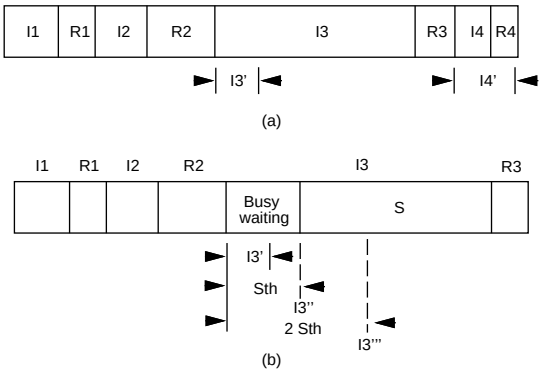

Figure 3: Correction of prediction miss: (a) an impulse-like idle period, (b) correction using a watchdog scheme.

2.3 Correction of prediction misses

The proposed prediction formula (Eqn. 8) 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 discussed as follows. Recall that our proposed prediction formula (Eqn. 8) 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 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 is 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 suffers 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), by the end of running state of $R2$, a long idle period $I3$ occurs at the end of a running state T. 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.

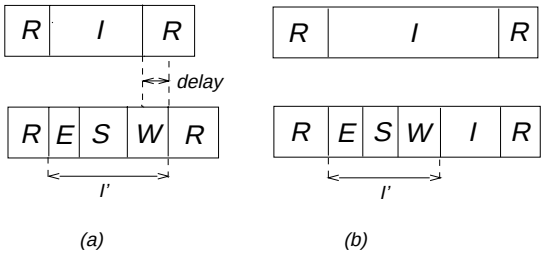

Figure 4: Examples of pre-wakeup model:(a) $I' > I$ *and* $D \leq W$, (b) $I' < I$.

To resolve the second problem, we add a saturation condition to Eqn. 8 as below.

$$if\ (ai_n + (1-a)I_n > cI_n)$$
$$I_{n+1} = cI_n, \tag{10}$$

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

2.4 Pre-wakeup

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 of 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 pre-wakeup the system before the arrival of the next wakeup signal. This can be accomplished by predicting the occurrence of the next wakeup signal.

Let I be the actual idle period, I' the predicted idle period, and $D = |I' - I|$ the error of the prediction. Figure 4 depicts two possible scenarios when applying the pre-wakeup 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 wakeup signal, as shown in Figure 4(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 wakeup signal. In this case, the pre-wakeup has no effect on the reduction of the delay penalty. Figure 4(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.

2.5 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 running

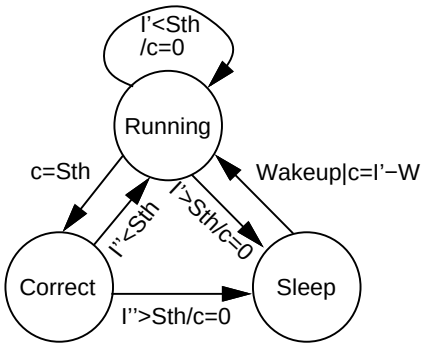

Figure 5: Finite state machine of the proposed method.

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 re-predicts 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 wakeup signal arrives. If the pre-wakeup scheme is applied to the system, then the system will return to the running state either when a wakeup signal arrives or the system has been shutdown for a predicted idle period ($c = I'$).

3 Experiments

We have conducted experiments on four event-driven applications: (1) X-server, (2) Netscape, (3) telnet, and (4) tin. The experiments were conducted on a SPARC 10 workstation running the SunOS 4.1.3 operating system.

In our implementation, we set the constants $a = 0.5$ (Eqn. 8) and $c = 2$ (Eqn. 10). We assume $E = W = \frac{1}{2}T_{cost}$, where T_{cost} is the delay overhead. First, we determine the threshold idle period S_{th} (Eqn. 7). Recall that

$$S_{th} \geq \frac{E \cdot (P_{EW} - P_S) + W \cdot P_{EW}}{(P_R - P_S)}. \tag{11}$$

We assume that the average power dissipation of entering and resuming from the sleep state P_{EW} is same as the power consumed in the running state; i.e., $P_{EW} = P_R$. In addition, according to the data sheet of the PowerPC [5], 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,

$$S_{th} \approx \frac{1/2 \cdot T_{cost} \cdot (P_R - 0.05 \cdot P_R) + 1/2 \cdot T_{cost} \cdot P_R}{(P_R - 0.05 \cdot P_R)} \tag{12}$$

$$= \frac{0.5 * 1.95}{0.95} \cdot T_{cost} = 1.026 \cdot T_{cost}. \tag{13}$$

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

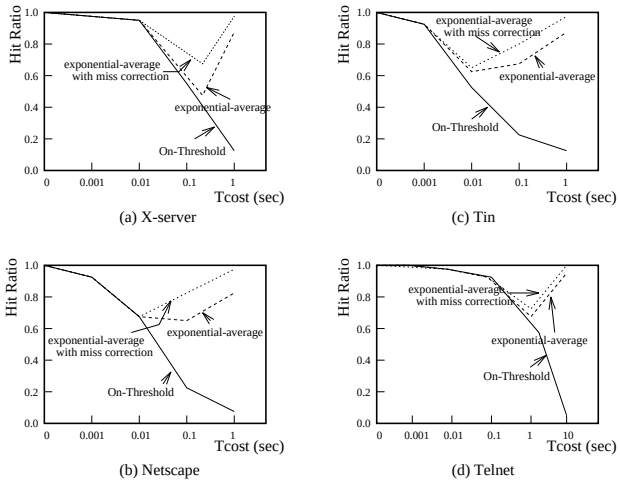

Figure 6: Comparisons of hit ratios.

$$DO = \frac{T_t' - T_t}{T_t} \times 100\%. \tag{14}$$

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 as

$$ES = \frac{P_R \cdot T_t}{P_R \cdot T_t' - T_s \cdot (P_R - P_S)}, \tag{15}$$

$$\approx \frac{P_R \cdot T_t}{P_R \cdot T_t' - T_s(P_R - 0.05 \cdot P_R)}, \tag{16}$$

$$= \frac{T_t}{T_t' - 0.95T_s}. \tag{17}$$

In the first experiment, we tested the hit ratio of our proposed method. We define a *hit* in two ways: (1) the system enters the sleep state and results in power saving or (2) the system does not enter the sleep state if it will not result in power saving. Figures 6(a), (b), (c), and (d) show the hit ratios of the four applications 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) $on-threshold(\infty)$ *based* [7], (2) our proposed method without miss correction, and (3) 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 ratio. 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 one is a method used in the Apple PC system [6] in which the system will enter

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	.03/18.53	.03/18.50	.03/17.99
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	.13/18.23	.13/14.54	.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

Table 1: Comparisons of various system shutdown methods.

the sleep state whenever a 2-second idle period is detected. The second and third methods were proposed by Srivastava et. al. [7]. 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 as observed in [7] 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 second and 2ms for T_{cost} of 1 second, 100ms for T_{cost} = 0-0.1 second and 1ms for T_{cost} = 1 second, 200ms for T_{cost} = 0-0.1 second and 1ms for T_{cost} = 1 second, and 40ms for T_{cost} = 0-0.1 second and 4ms for T_{cost} = 1 second, for the X-server, Netscape, Tin, and Telnet, respectively. The fourth method is our proposed method without applying the pre-wakeup scheme. The final method is our proposed method including the pre-wakeup scheme.

Table 1 shows the results. From the results, 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 second. Second, when T_{cost} is in the range of 0 to 0.1 second, the $PM_{Th\infty}$ achieves the best power saving. However, when T_{cost} = 1, power saving drops sharply and delay overhead increases rapidly. For example, for the X-server application, it obtains 1.53 times power 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 2 to 3 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 second 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 second. Finally, when applying the pre-wakeup technique, the delay overhead can be reduced by sacrificing some power saving (PM_{II} v.s. PM_I). This is important for some timing critical applications. In our approach, we are able to tradeoff delay overhead and power saving by enabling and disabling the pre-wakeup function.

4 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 have shown that when the shutdown overhead is high (e.g., $T_{cost} \geq 1$ second), the conventional method, such as that used by the Apple PowerPC, can achieve a reasonable power saving. However, this method does not achieve an effective power saving when the shutdown overhead is low (e.g., $T_{cost} <$ 1 second). On the other hand, when the shutdown overhead is low (e.g., $T_{cost} < 1$ second), a more aggressive predictive shutdown method [7] can achieve a higher degree of power 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 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 have shown that our proposed method achieves high hit ratios for a wide range of shutdown overheads, which results in a high degree of power saving with low delay penalties.

Currently, our proposed method focuses on a single sleep mode. However, many processors provide a variety of sleep modes with different levels of power saving and shutdown overhead. Future work will extend our method to accomodate multi-level sleep modes.

References

[1] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, no. 1, pp.3-56, January, 1996.

[2] S. Devadas and S. Malik, "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits," in *Proc. of the 32nd Design Automation Conf.*, pp. 242-247, 1995.

[3] F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits," *IEEE Trans. on VLSI Systems*, vol. 2, no. 4, pp.446-455, December 1994.

[4] F. N. Najm, "Power Estimation Techniques for Integrated Circuits," in *Proc. of the ICCAD*, pp. 492-499, 1995.

[5] S. Gary, "PowerPC: A Microprocessor for Portable Computers," *IEEE Design of Computers*, pp. 14-23, Winter 1994.

[6] Apple Computer Inc., "Power Manager IC and Reduced Power Modes," in *Technical Introduction to the Macintosh Family*, Reading, MA: Addison-Wesley, Oct. 1992.

[7] M. B. Srivastava, A. P. Chandrakasan, and R. W. Broderson, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Trans. on VLSI Systems*, pp. 42-54, vol. 4, no. 1, March 1996.

[8] J. L. Peterson and A. Silberschatz, *Operating System Concepts*, 2nd Ed., pp. 118-120, Addison-Wesley Publishing Co. Inc.