Des Autom Embed Syst (2009) 13: 311–332
DOI 10.1007/s10617-009-9047-1

An effective state-based predictive approach for leakage energy management on embedded systems

Minyi Guo · Linfeng Pan · Yanqin Yang · Meng Wang · Zili Shao

Received: 14 January 2009 / Accepted: 20 July 2009 / Published online: 8 August 2009

Abstract Energy optimization is very important for portable and battery-driven embedded systems. With the shrinking of transistor sizes, reducing leakage power becomes a significant issue. In this paper, we propose a novel prediction approach to predict idleness of functional units for leakage energy management. Using a state-based predictor, historical utilization information of functional units (FUs) is exploited to adjust the state of the predictor so as to enhance the accuracy of prediction; based on it, the idleness of the FUs are predicted and utilized for leakage reduction by applying power gating. We design two prediction algorithms, the prediction with fixed threshold (PFT) and the prediction with dynamic threshold (PDT), respectively. We implement our algorithms based on SimpleScalar and conduct experiments with a suite of fourteen benchmarks from Trimaran. The experimental results show that our algorithms achieve better results compared with the previous work.

Keywords Power-gating · Leakage energy management · State-based predictor · Dual thresholds

This version is a revised version. A preliminary version of this work appears in the Proceedings of the 2008 IEEE/IFIP International Conference On Embedded and Ubiquitous Computing (EUC 2008) [29]. The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF POLYU 5260/07E) and HK PolyU 1-ZV5S, National 863 Program of China (Grant No. 2006AA01Z172 and Grant No. 2008AA01Z106), National Natural Science Foundation of China (Grant No. 60533040), and National Science Fund for Distinguished Young Scholars (Grant No. 60725208).

M. Guo · L. Pan · Y. Yang
Department of Computer Science and Engineering, Shanghai Jiao-Tong University, Shanghai 200240, China

M. Wang · Z. Shao (✉)
Department of Computing, Hong Kong Polytechnic University, Hong Kong, Hong Kong
e-mail: cszlshao@comp.polyu.edu.hk

1 Introduction

As portable and battery-powered embedded systems are widely used, energy optimization becomes an important issue. With 0.18 µm or above technology, dynamic power is the biggest concern since it accounts for 90% or more of the total chip power [30]. To reduce dynamic power, the supply voltage is reduced with each processor generation. In order to maintain performance at lower supply voltage, the threshold voltage is decreased accordingly. As the threshold voltage decreases, transistor leakage current increases exponentially. The leakage power constitutes about 54% of the total power with 65 nm technology, and it will be further increased with future technologies [11]. Therefore, reducing leakage power becomes one of the most important research problems.

Power supply gating is an effective leakage management technique by shutting down the power supply of idle functional units. With its effectiveness in controlling leakage power, a lot of power-gating-based techniques have been proposed in recent work [3, 4, 6, 20, 21, 27, 33, 41, 46]. To effectively apply power gating, one of most important problems is how to predict the sufficiently long idleness of functional units. Various prediction approaches have been proposed from both software [7–9, 25, 26, 32, 44, 45] and hardware aspects [15, 43].

In [32], a compiler-based approach is proposed to find out the idle region by analyzing the code. At task level, energy-efficient task scheduling techniques with considerations of leakage power dissipation have been explored in [7–9, 25, 26, 44, 45]. These techniques are dependent on the certain hardware configurations. In [15, 43], the threshold-based approaches are proposed to predict the idleness for leakage savings. With these approaches, however, decisions are made based on a single threshold. So it may not always produce good results.

In this paper we propose a state-based predictor to store the historic utilization information of functional units, and dynamically adjust its states so as to predict possible idleness of FUs for power gating. Different from the previous work, decisions are made based on the state of the predictor that reflects whether or not the previous idle periods are opportunities for power-gating. To the best of our knowledge, this is the first work to introduce a state-based predictor for power gating. Our main contributions are summarized as follows:

- A state-based predictor is designed based on a four-state finite state machine. The states of the predictor are dynamically updated with the information of previous idle periods. Based on the predictor, we can predict the idleness more accurately so as to save more leakage power compared with the existing prediction approaches.
- Based on the predictor, we design two prediction algorithms, the prediction with fixed threshold (PFT) and the prediction with dynamic threshold (PDT), respectively. With the state-based predictor, in our algorithms, we make decisions for applying power gating based on fixed thresholds (PFT) or dynamic thresholds (PDT).
- Our approach is based on simple control logics and independent of architectures. So it can be easily implemented into various microprocessor designs. In particular, it is suitable for DSP processors that are used to process applications with a lot of loops.
- We implement our algorithms into SimpleScalar [17] and conduct experiments with a set of benchmarks from Trimaran [16]. The experimental results show that our algorithms achieve better results compared with the previous work in [15, 43]. In particular, our PDT algorithm allows the fixed-point units to be put into sleep for 54.4% of the idle cycles, and 97.7% for the floating-point units with an average performance loss of 3.2%. Based on our analysis on the dual core UltraSPARC microprocessor, our approach will introduce less than 0.2% energy overhead.

The rest of the paper is organized as follows. Related work is described in Sect. 2. Concepts and models for power gating are introduced in Sect. 3. A simple example is provided to show the drawbacks of other predictive approaches in Sect. 4. Our state-based prediction approach is proposed in Sect. 5. Then the experimental results are exhibited in Sect. 6. Finally, concluding remarks are made in Sect. 7.

2 Related work

Many techniques have been proposed to reduce leakage power, and they can mainly be divided into two categories: static techniques and dynamic techniques [38].

The static leakage control techniques include design optimization methods for leakage current reduction in circuit and architecture levels. The use of dual-V_T transistors in critical paths and non-critical paths is one of the most common static leakage-reduction techniques [19, 24, 37, 39, 40]. Similarly, stack forcing technique is applied in non-critical paths of the circuit to reduce the leakage due to the stack effect [28]. However, as pointed out in [38], with the increase of the number of critical paths in a design, the above techniques cannot efficiently solve the problem. Several techniques are proposed to employ the circuit styles optimized for low leakage power [23]. However, the circuits implemented based on this technique have a relatively low performance; therefore, they can only be used in non-critical paths in a design.

Dynamic techniques identify "idle" or "standby" states while circuits do not need to execute operations. These techniques have been applied at both block and chip levels. Since not all parts of a processor are busy at the same time in practice, block-level techniques can effectively reduce leakage power. In [1, 2, 18, 42], a block-level technique, called input vector control (IVC), is proposed to reduce leakage power using the transistors stacking effect. In this technique, the minimum leakage vector is found for minimizing leakage current. The minimum vector, however, is hard to be obtained since the problem is solved in exponential time [18].

Power supply gating is another effective dynamic leakage management technique. The basic idea of power gating is to shut down the power supply of idle units so as to reduce the leakage power. It is implemented by adding a "sleep transistor" in series with the power supply, which is turned off when the circuit block is in idle mode. Body-bias-based techniques can be combined with a sleep transistor to obtain further leakage power savings [21]. As power gating can effectively control leakage power, a lot of power-gating-based techniques have been proposed in recent work [3, 4, 6, 20, 27, 33, 41, 46].

With power gating technique, one of most important problems is how to predict the sufficiently long idleness for power gating. Various prediction approaches have been proposed. In [32], a compiler-based approach is proposed to identify the region in which functional units are expected to be idle. At task level, energy-efficient task scheduling techniques with considerations of leakage power dissipation have been explored in [7–9, 25, 26, 44, 45]. Luo et al. [25, 26] addressed the problem of variable voltage scheduling of multi-rate periodic task graphs in heterogeneous distributed real-time embedded systems. Kuo et al. [7–9] developed various on-line simulated scheduling strategies and a virtually blocking time strategy for procrastination scheduling to reduce leakage power consumption on a uniprocessor DVS system. Their algorithms derived a feasible schedule for real-time tasks with worst-case guarantees for any input instance. Xu et al. [44, 45] proposed a dynamic programming algorithm for periodic tasks on processors with practical discrete speed levels. Their algorithm determined the lower bound of energy expenditure in pseudo-polynomial time. To

save leakage energy, special instructions are inserted to communicate with hardware, which expands the code size. And the scheduling results produced by these techniques are highly dependent on the configurations of hardware.

The counter-based prediction method has been widely explored in the past [5, 10, 13, 22, 31, 34, 35]. In [5, 10], the idle periods for a component were modeled as stationary discrete-time Markov processes and semi-Markov processes. In [34], Tajana et al. computed optimal decisions in advance for different sets of arrival requests and stored the results in a table that is used for making decisions at runtime. In [22, 31, 35], several genetic algorithms were proposed to predict lengths of future idle periods. In [13], a novel energy-saving method is proposed to use program counters to predict I/O activities in the operating system. These approaches provide effective predictions for power savings for hard disks or other large systems, but they are too complicated to be applied for functional units on a chip.

In [15], a scheme is proposed to shut down functional units after they are idle for a fixed time interval based on a counter. For simplicity, we call this technique FTHP (Fixed Threshold Prediction). In this scheme, since the decisions are based on an aptotic criterion, it may not always accurately predict the idleness for leakage savings. In [43], a scheme is proposed to predict idleness based on a threshold that is adjusted dynamically based on the accuracy of its recent predictions. We call this technique DTHP (Dynamic Threshold Prediction). With this approach, however, decisions are made based on a single threshold. So it may not always produce good results.

3 Basic concepts and models

In this section, we introduce some basic concepts and models that will be used in the later sections.

Power gating uses a suitably sized header or footer transistor as a "sleep transistor". When a sufficiently long idle period of the circuit block is detected, a "sleep" signal is applied to the gate of the "sleep transistor" to turn off the supply voltage of the circuit block. And when the circuit block is requested for use, the voltage is restored to the working voltage. A power gating cycle can be mainly divided into three stages: shutting-down signal generation, voltage decrease and power-on signal generation [15].

Energy overhead is caused by shutting down and powering on. A break-even point is the time point when the overhead energy incurred by switching on and off the device is equal to the leakage power savings from the period when FU is in the shutting-down mode. At the break-even point, the aggregate leakage energy savings compensates the total energy overhead of transition. After break-even point, the leakage power saving becomes the net income. We use $T_{\text{BreakEven}}$ to represent the number of cycles that an FU can reach the break-even point after it is shut down. It is dependent on the configuration (the block size, the decoupling capacitance, etc.) of a circuit. There have been several studies related to this. For example, in [12], the worst-case leakage behavior relative to the dynamic energy is modeled; in [41], it shows that the value of $T_{\text{BreakEven}}$ can be changed from 1 to 128 cycles; in [15], based on a parameterizable model, it shows that the value of $T_{\text{BreakEven}}$ can be as small as 10 cycles.

We can gain energy savings by power gating if and only if the number of consecutive idle cycles is larger than $T_{\text{BreakEven}}$; otherwise, the saved leakage energy can not compensate the energy overhead of transition. In practice, we cannot shut down an FU as soon as it enters idle mode, because it is possible that it will be requested in a very short period. Fortunately, we can approximately predict the idleness of an FU based on a general phenomenon: if the

FU has been idle for some time, it is very possible that it will continue being idle for a while. Therefore, we may gain energy saving if we first wait for $T_{\text{Threshold}}$ cycles after the FU enters the idle mode and then generate a "sleep" signal to shut down the FU. Here $T_{\text{Threshold}}$ is an integer value that can be fixed or dynamically changed based on different schemes.

For simplicity, we introduce a new variable as T_{Balance}:

$$T_{\text{Balance}} = T_{\text{Threshold}} + T_{\text{BreakEven}}$$

T_{Balance} is used to represent the actual break-even point by waiting $T_{\text{Threshold}}$ before entering into the sleep mode, since $T_{\text{Threshold}}$ is the number of cycles to wait from the point that an FU becomes idle to the point that it enters into the sleep mode, and $T_{\text{BreakEven}}$ is the number of cycles that an FU needs to be in the sleep mode for compensating the transition overhead. Based on the definition of T_{Balance}, we define "mistake" and "hit" as follows: A mistake occurs if we shut down an FU when the total length of an idle period is less than T_{Balance}; a hit occurs if we shut down an FU when the total length of an idle period is equal to or greater than T_{Balance}.

Given an idle time period t_{idle}, if power gating is applied, the saved energy for an FU can be calculated as follows:

$$E_{\text{energy\_saved}} = E_{\text{leakage\_saved}} - E_{\text{overhead}} = P_{\text{leakage}} \cdot (t_{\text{idle\_sleep}} - t_{\text{breakeven}}) \tag{2}$$

Here, P_{leakage} is the leakage power of the FU; $t_{\text{idle\_sleep}}$ is the time that the FU is put into the shutting-down mode; $t_{\text{breakeven}}$ is the time that the FU needs in order to reach the break-even point after it is shut down, and it can be obtained by the product of $T_{\text{BreakEven}}$ and the cycle period. P_{leakage} and $t_{\text{breakeven}}$ are dependent on the circuits design of FUs. For a given FU, our target is to make the prediction more accurate in such a way that the idle time period can be correctly utilized so $t_{\text{idle\_sleep}}$ can be larger than $t_{\text{breakeven}}$. In other words, we attempt to fully utilize all hit opportunities (as mentioned the above) for energy saving.

4 Example

In this section, we give an example and show how the two previous techniques, FTHP [15] and DTHP [43], work for managing leakage energy. The example is based on a real program, the "mm" from the Trimaran benchmarks [16], which performs the matrix multiply computation, shown in Fig. 1.

We first run the program on the SimpleScalar simulator [17] with the same configuration as shown in Sect. 6 and obtain all idle periods during its executions. For demonstration purpose, we only show the idle periods of fp-ALU0. The program mainly consists of two loops, and their corresponding idle periods are shown in Segments A and B, respectively, in Fig. 1, in which "CPU Cycles" represents the time when the fp-ALU0 ends the idle period, and "Idle Period" represents the length of the idle period (the number of cycles).

Using FTHP [15], an FU will wait for $T_{\text{Threshold}}$ cycles after the FU enters the idle mode and then generates a "sleep" signal to shut the unit down. Using DTHP [43], a quartet ($T_{\text{StepMistake}}$, T_{StepHit}, $T_{\text{mistake-limit}}$, $T_{\text{hit-limit}}$) is used to adjust the value of $T_{\text{Threshold}}$. In the quartet, the first two parameters are used to increase/decrease the threshold, and the last two parameters are used to decide when to apply an adjustment.

Without loss of generality, we use the parameters in Table 1 for FTHP and DTHP. As shown in Table 1, $T_{\text{Threshold}}$ is 8 cycles that is fixed for FTHP and is the initial value for DTHP, and $T_{\text{BreakEven}}$ is 10 clock cycles. $T_{\text{mistake-limit}}$, $T_{\text{StepMistake}}$, $T_{\text{hit-limit}}$, and T_{StepHit} are

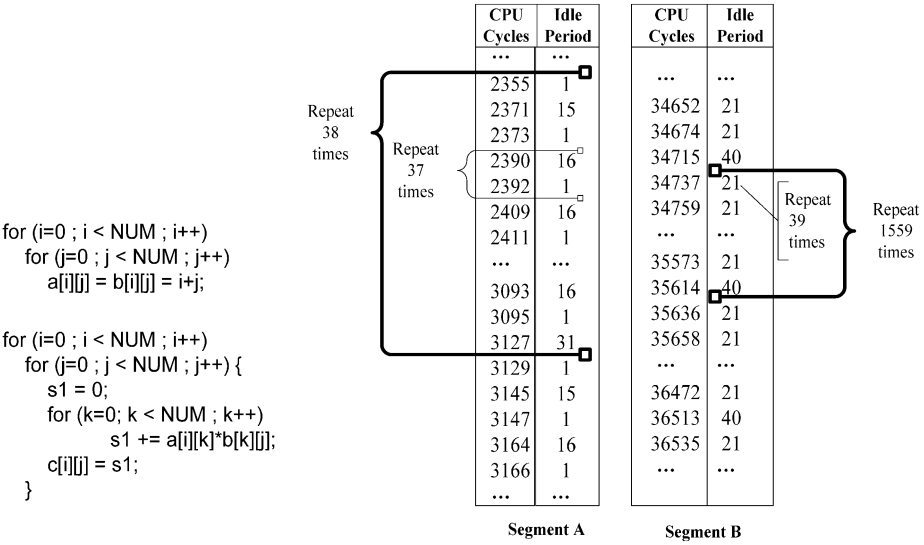


Fig. 1 Two segments with idle periods of fp-ALU0 during the *MM* execution on SimpleScalar

Table 1 Parameters of FTHP and DTHP

$T_{\text{Threshold}}$	$T_{\text{BreakEven}}$	$T_{\text{StepMistake}}$	T_{StepHit}	$T_{\text{mistake-limit}}$	$T_{\text{hit-limit}}$
6	12	1	2	4	20

used to adjust the threshold in DHTP: if consecutive mistakes reach 4 times ($T_{\text{mistake-limit}}$), $T_{\text{Threshold}}$ will be increased by 1 ($T_{\text{StepMistake}}$); if consecutive hits reach 20 times ($T_{\text{hit-limit}}$), $T_{\text{Threshold}}$ will be reduced by 2 (T_{StepHit}). DTHP sets the upper bound of $T_{\text{Threshold}}$ as $T_{\text{BreakEven}}$ minus $T_{\text{StepMistake}}$.

If FTHP is used, for Segment A in Fig. 1, in any idle period, fp-ALU0 will be shut down after it enters idle mode for 6 cycles. Because most of the periods in Segment A are smaller than 18 cycles, it results in large number of mistakes. And for Segment B, there is no mistake. But for most of the idle periods, we can only obtain very small energy savings.

When DTHP is applied, for Segment A, there is no hit except for the period with a length of 31 cycles. When the number of consecutive mistakes increases to 4, $T_{\text{Threshold}}$ is increased by 1 and it will keep increasing until it is 11. As the execution enters Segment B, $T_{\text{Threshold}}$ is 11; because this $T_{\text{Threshold}}$ is too large, DTHP keeps making wrong decisions and has little energy savings in Segment B.

It is important to note that, based on the configuration of Table 1; any value of $T_{\text{IdlePeriod}}$ between 6 and 18 cycles will result in a mistake for both FTHP and DTHP. So the data in Fig. 1 represent a category of applications. On the other hand, if $T_{\text{Threshold}}$ varies from 6 to 11, both FTHP and DTHP may make a lot of mistakes or miss hit opportunities. This example indicates that FTHP and DTHP may make bad mistakes or miss the opportunities in some cases. The main reason is that a single threshold is not sufficient for predicting the length of the idle periods. It motivates us to propose a state-based approach to solve the problem next.

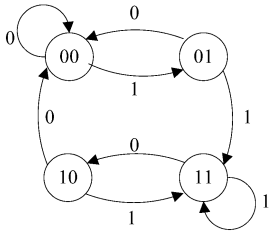


Fig. 2 The state-based predictor

5 State-based prediction approach

In this section, we propose our state-based prediction approach. First, we propose a state-based predictor in Sect. 5.1. Then we design two prediction algorithms, the prediction with fixed threshold (PFT) and the prediction with dynamic threshold (PDT) in Sects. 5.2 and 5.3, respectively.

5.1 State-based predictor

Let's consider such an idleness sequence pattern which contains consecutive short idle periods followed by consecutive larger idle periods. Consecutive long idle periods indicate that the FU may be slightly used currently, so the coming idle period would be a hit opportunity with a high possibility. On the other hand, consecutive short idle periods tell that applying power-gating for the coming idle period may not be an advisable decision. When DTHP meets a serial of mistakes, it keeps increasing threshold (to the upper bound). It not only introduces much overhead due to mistakes, but also reduces the leakage savings for the coming hit opportunities because of a higher threshold. In some bad cases, the high threshold would make mistakes for the idle periods which would be hit opportunities for a smaller threshold. In this paper, we use a state-based predictor to solve such problems. The state machine of the predictor is shown in Fig. 2.

As shown in Fig. 2, the predictor is a finite state machine based on a saturation up-down 2-bit counter [14]. It has four states, "00", "01", "10" and "11". The states "00" ("strongly not taken") and "01" ("weakly not taken") represent that it is very possible ("strongly not taken") or possible ("weakly not taken") that a mistake will occur based on the accuracy of previous decisions. Similarly, states "10" ("weak taken") and "11" ("strong taken") are used to represent that it is possible or very possible that a hit will occur based on the accuracy of previous decisions.

The state transitions of the predictor are updated according to whether or not an idle period is a hit opportunity that is obtained from a counter as shown in Sects. 5.2 and 5.3. If an idle period is a hit opportunity, then the input is "1"; otherwise, the input is "0". With different inputs, the states of the predictor are changed based on the state transition graph in Fig. 2, and the initial state is "00". In this way, we can record the information of the previous idle periods. Based on this predictor, two prediction schemes are proposed in Sects. 5.2 (with fixed threshold) and 5.3 (with dynamic threshold), respectively.

5.2 PFT: prediction with fixed threshold

Based on the predictor, we first propose a prediction algorithm with fixed threshold. The algorithm PFT is shown in Fig. 3.

Input: The state of an FU (1 busy/0 idle); the current state of the predictor; a counter to record the length of an idle period.
Output: Sleep Signal for the FU and the input signal for the predictor.
Algorithm:

```
if FUState == 1          // fu busy
  if Counter < T_Threshold
        Counter=0;
        Return;
  else if Counter>=T_Threshold+T_BreakEven //should be a hit
        Update the predictor with '1';
        Counter = 0;
  else
        Update the predictor with '0'
        Counter=0;
  endif
else              // fu idle
  Counter=Counter+1;
    if Counter == T_Threshold
        if the state of the predictor is "11" or "10"
              Generate sleep signal for the FU;
        endif
  endif
endif
```

Fig. 3 The PFT algorithm

The inputs of the PFT are the state of an FU (‘1’ for busy; ‘0’ for idle), the state of the predictor, and a counter to record the length of an idle period. In our algorithm, we make the decision to shut down the FU as follows:

When an FU is not idle (FUState == 1), there are three possible cases:

(a) The FU is requested again and the idle period is less than $T_{\text{Threshold}}$ cycles. The idle period is so small that there is no chance for leakage reduction. Since we do not make any decision in such an idle period, we just clear the counter and get ready for the next idle period.
(b) The FU has been idle for more than T_{Balance} cycles, which means it is good to apply power gating. Since it is a hit opportunity, we update the state of the predictor with ‘1’ to store this historical information.
(c) The FU has been idle for more than $T_{\text{Threshold}}$ cycles but less than T_{Balance} cycles. To avoid making wrong decisions in such idle periods, we update the state of predictor with ‘0’. In both cases (b) and (c), the counter is cleared.

When the state of the FU is ‘0’, representing that FU is (still) in idle mode, we increase the counter, and make a decision based on the predictor. If the state of the predictor is “11” or “10”, we generate a signal to shut down the FU. Otherwise, the FU is kept active.

Next we give an example to show how PFT works by applying it on the program shown in Sect. 4. We use the same threshold, 8 cycles, as in Sect. 4, and the initial state of the predictor is “00”. For the first idle period of Segment A, when the idle period is accumulated to 8 cycles, the predictor predicts “strongly not taken”, which means that it is not an opportunity for power gating. And the fact proves that it is correct and the state of the predictor keeps as “00”. The situation for the following idle periods in Segment A is exactly the same

as for the first one. When it comes to the idle period with a length as 31 cycles, the state of the predictor is updated to "01". And the state is set back to "00" for the following periods. Though PFT fails to apply power-gating in the idle periods which contain 31 cycles, it successfully avoids making any mistake. Compared to FTHP and DTHP, PFT introduces less power overhead when the FU is heavily used.

All the idle periods in Segment B are hit opportunities. For the first idle period, the predictor predicts "strongly not taken" in the beginning but it is a wrong decision. The state of the predictor is updated to "01". Then for the second idle period, the predictor makes a mistake again but the state is updated to "11". After that, the predictor predicts "strongly taken" for each period till the end of loop B. For consecutive short idle periods, PFT would make two mistakes at most and increase the threshold only once. During the whole execution, the PFT only misses 2 hit opportunities. It obtains remarkably more leakage savings than FTHP and DTHP.

5.3 PDT: prediction with dynamic threshold

Lowering the threshold gains more leakage savings. For an application with many medium-sized idle periods which are a little larger than T_{Balance}, a low threshold may make a notable contribution to reducing leakage. On the other hand, a low threshold may also cause more mistakes so as to introduce more energy overhead. As the patterns of idle periods in applications may vary a lot, we may get better results if we can dynamically adjust the threshold. Therefore, in this section, we propose a variation with dynamic threshold.

From the predictor in Fig. 2, if the state of the predictor is repeatedly updated with "1", it tells us that it is possible that we may gain more energy savings by lowering the threshold. On the other hand, switching the state from "10" to "00" means that the predictor has suffered two consecutive mistakes, which indicates that an FU may be heavily utilized currently. In this case, therefore, we need a higher threshold to reduce future mistakes. Based on this idea, we design our PDT algorithm, a prediction algorithm with dynamic threshold, and it is shown in Fig. 4.

The inputs and outputs of PDT are the same as PFT. Before PDT starts, two variables *bPredicted* and *HitOpp* are initialized. *bPredicted* represents whether a decision for power gating has been made for the current idle period, while *HitOpp* records the number of the previous consecutive hit opportunities.

Similar to the PFT algorithm, the algorithms first check the status of FU. If FUState is 1, which it represents that FU is busy, then there are two cases:

(1) Either the FU is always busy or only has a short idle period so that the counter for the idle cycles is smaller than threshold. In this case, PDT does nothing, but clears up the counter, as shown in Lines 2–4.
(2) When PDT detects that it is an adequately long idle period (Line 5), both the state of predictor and *HitOpp* are updated. If *HitOpp* reaches $T_{\text{hit-limit}}$, the threshold may be reduced. And *HitOpp* is cleared for the coming hit opportunities (Lines 6–11). On the other hand, if there are two consecutive idle periods which are longer than threshold but not hit opportunities, the threshold is increased to avoid possible mistakes. In this case, the state of the predictor is updated from "10" to "00" (Lines 12–16). Since it is no longer in an idle period, both the counter and the flag of prediction (*bPredicted*) are cleared up (Lines 17, 19).

If FUState is 0, it represents that FU is idle. Then we check the number of idle cycles (Lines 22–24). When the number of idle cycles reaches the threshold and the predictor

Input: The state of an FU (1 busy/0 idle); the current state of the predictor; a counter to record the length of an idle period.
Output: Sleep Signal for the FU and the input signal for the predictor.

Algorithm:
Input: FU state(1 busy/0 idle)
Output: Sleep Signal Generation

```
bPredicted=false; //initialize
HitOpp=0;

Function PDT(FUState)
1.    if FUState ==1  // Busy
2.      if Counter < Threshold
3.       Counter=0;
4.       Return;
5.      else if Counter >= Threshold+BreakEven
                  // A hit opportunity
6.        Update the predictor with '1'
7.        HitOpp = HitOpp +1;
8.        if HitOpp == Thitlimit and Threshold> Tstephit
9.          Threshold=Threshold-Tstephit;
10.         HitOpp =0;
11.       endif
12.     else if the state of predictor is "10"
13.         Threshold=Threshold+Tstepmistake;
14.       endif
15.       Update the predictor with '0';
16.     endif
17.       Counter=0;
18.    endif
        //initialized for new power gating cycle
19.    bPredicted=false;
20.  else
21.    Counter=Counter+1;
22.     if Counter == Threshold
23.       if the state of the predictor is "11" or "10"
24.         Generate sleep signal
25.         bPredicted=true;
26.       endif
27.     else if Counter == Threshold2 and bPredicted==false
28.       Generate sleep signal;
29.     endif
30.  endif
```

Fig. 4 The PDT algorithm

predicts ‘taken’, PDT generates a sleep signal for power gating (Lines 22–24). The flag of the prediction is set as ‘true’, representing that a decision for power gating has been made. PDT shuts down the FU for long idle periods when it fails to predict for power gating (Lines 27–29).

Next we give an example to show how PDF works by applying it on the program shown in Sect. 4. When the state of predictor is initialized as "00", PDT works exactly the same as PFT does for the Segment A in the example in Sect. 4. For Segment B, PDT makes two wrong decisions in the beginning, and updates the state of predictor as "11". After that, the predictor always predicts "strongly taken". During the execution, PDT lowers the threshold until the threshold reaches 1. And we can obtain more energy savings with dynamic threshold. Since the predictor has the ability to avoid making consecutive mistakes, we could initialize the primary threshold with a smaller value.

Compared to FTHP, PFT and PDT make more creditable decisions based on the predictor, which stores the information of previous idle periods. For consecutive hit opportunities, PDT lower the threshold in time more leakage savings. Compared to DTHP, the threshold of PDT does not increase rapidly for a sequence of small idle periods.

6 Experiments

We implement our algorithms into the SimpleScalar and conduct experiments with a set of benchmarks from the Trimaran. In the experiments, we compare our PFT and PDT algorithms with FTHP [15] and DTHP [43]. In this section, we first introduce the experimental environment. Then we present the results and discussion. Finally, we analyze the power overhead introduced by our approach.

6.1 Experimental environment

In order to compare these predictive approaches, a modified SimpleScalar is used as our experiment platform. We integrate FTHP, DTHP and our prediction approaches into SimpleScalar (version 3.0d [17]) respectively. The usage of different FUs is traced and the FUs are turned on/off according to the predictions and requests. When an FU is turned off, we record the number of cycles, which can be translated to leakage energy savings. The configuration of SimpleScalar used in the experiments and the information of functional units are shown in Table 2.

To compare the effectiveness of our state-based approaches with the other two approaches, we used a suite of fourteen benchmarks from Trimaran [16], as shown in Table 3. These benchmarks produced various patterns of idle periods. Note that the patterns of idle periods, rather than the sizes of applications, have the biggest impact for the effectiveness of prediction. We calculated the total energy savings that are obtained based on the power model in [15]. For a mistake with negative energy savings, the excess of overhead energy was deducted from total energy savings.

Table 2 Functional units for experiments

Functional unit	Available units	Operational latency (cycles)	Issue latency (cycles)
Integer ALU	4	1	1
Integer multiplier	1	3	1
Integer divider	1	20	19
Floating-point adder	4	2	1
Floating-point multiplier	1	4	1
Floating-point divider	1	12	12

Table 3 Information of benchmarks

Benchmarks	Integer operations	Fp operations	Description
Bmm	√	√	The matrix multiply computation
dag	√	×	Loops with multiple if-conditions
eight	√	×	Loops with multiple if-conditions
fib	√	×	The Fibonacci number computation
fir	√	√	FIR filter code
hyper	√	×	Loops with multiple if-conditions
ifthen	√	×	Loops with multiple if-conditions
mm	√	√	The matrix multiply computation
nested	√	×	The computation on multi-dimensional arrays
sqrt	√	√	Newton Raphson method to find the roots
strcpy	√	×	String copy
switch	√	×	Loops with multiple if-conditions
type	√	×	The computation with various data types
wave	√	√	The wavefront computation

Table 4 Percentage of the idle cycles put to sleep

FU	FTHP(%)	DTHP(%)	PFT(%)	PDT(%)
int-FU	50.4	40.9	48.4	54.4
Fp-FU	91.1	84.5	90.7	97.7
all-FU	77.7	70.1	76.8	83.4

6.2 Results and discussions

The experimental results for different types of functional units with the fixed parameters are presented in Sect. 6.2.1. We then study the trends and influences with different thresholds and break-even points in Sect. 6.2.2. All results are normalized to the corresponding results of DTHP, which are eliminated from the figures. Note that, except the figures with specific benchmark names, all results are reported as the mean of normalized leakage savings for all benchmarks.

6.2.1 Performance comparisons with fixed parameters

To compare various functional units, the parameters in Table 1 are first used. Except for DTHP, other three techniques need a subset of the parameters. For example, PDT is initialized with ($T_{\text{Threshold}}$, $T_{\text{BreakEven}}$, $T_{\text{StepMistake}}$, T_{StepHit}, $T_{\text{hit-limit}}$).

6.2.1.1 Overview of all functional units Table 4 shows the percentage of the idle cycles at which functional units are actually utilized (put into sleep) with the four algorithms. For example, PDT shuts down the floating-point FUs for 97.7% of the idle cycles. Only 2.3% of the idle cycles are not utilized due to the threshold. From the results, we can see that PDT is the best for both int-FUs and fp-FUs. The reason why FTHP shuts down more idle cycles than PFT is that FTHP tries to shut down FU as soon as the threshold is reached. Due to the heavy overhead introduced by wrong decisions, FTHP performs badly for int-ALUs.

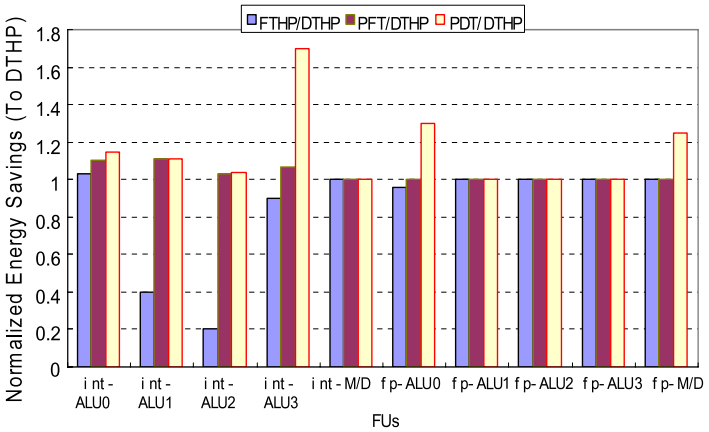


Fig. 5 Energy savings for different FUs (Comparing FTHP, DTHP, PFT and PDT, normalized to DTHP using the parameters in Table 1, and $T_{\text{threshold2}} = 40$)

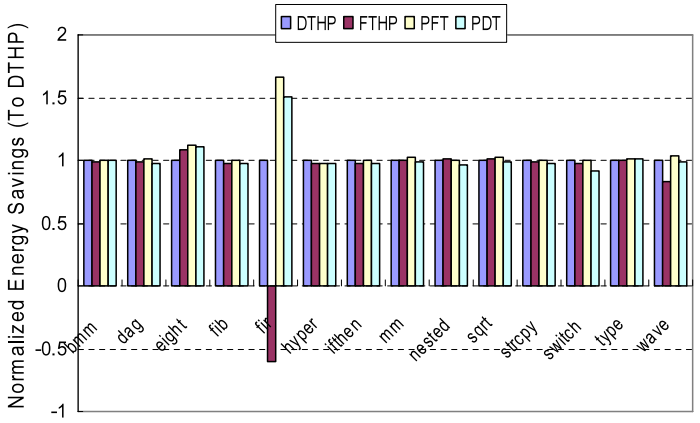
Figure 5 gives an overview of the energy saving for different functional units. In general, both PFT and PDT perform better than previous two approaches. And for int-ALU3, fp-ALU0 and fp-Mul/Div, PDT outperforms the others. On the other hand, for int-Mul/Div and all fp-ALUs except for fp-ALU0, all predictive approaches have the same performance. The detailed analysis is presented in next sections.

6.2.1.2 Heavily utilized functional units Most integer ALUs are heavily utilized in all functional units because they are used by a lot of instructions such as *Integer ADD*, *SUB*, *JUMP*, *BRANCH*, *MOVE* and logic instructions. Most idle periods of integer ALUs are small pieces, averagely smaller than 30 cycles.

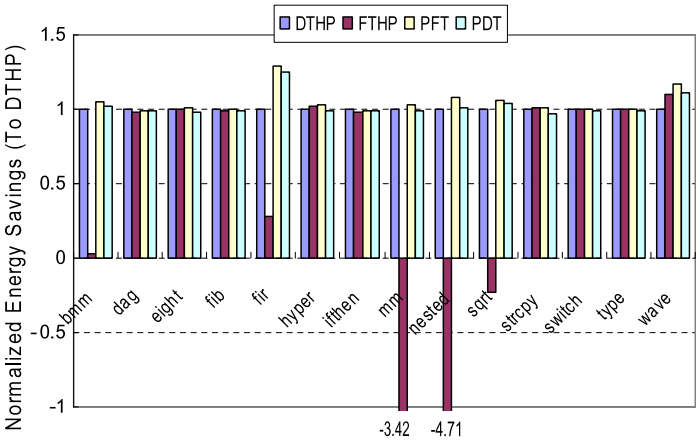
Figure 6 depicts the results of the average energy savings for all integer ALUs. From the results, we can see that our approaches gain more average energy savings compared with the other two approaches. The reason is that using DTHP, after some mistakes, the threshold will be increased, and with a relatively large threshold, a lot of hit opportunities cannot be utilized for energy savings. On the other hand, FTHP keeps making wrong decisions with the fixed threshold by shutting down the functional units in the idle periods which do not last long enough for compensating transition overhead. Therefore, in Fig. 6, FTHP even increases energy consumption for some benchmarks. Both PFT and PDT work well and achieve more energy savings compared with FTHP and DTHP. On average, PFT and PDT achieve 102.1% and 112.6% in energy savings compared to DTHP.

6.2.1.3 Slightly utilized functional units The floating-point ALUs perform the floating-point ADD/SUB operation, integer to floating-point conversion and floating-point comparison. Compared to the integer ALUs, integer multiplier/divider and floating-point functional units are less utilized. Except for the first fp-ALU used by the scheduler, the other three fp-ALUs are idle for most of time during execution.

Figure 7 depicts the results of the total energy savings for these slightly utilized functional units. The benchmarks of bmm, fir, mm, sqrt and wave have float-point operations; the other 9 benchmarks do not, which means all the floating-point FUs are always idle during the execution of these 9 benchmarks. From Fig. 7(b), we can find that DTHP performs badly for benchmark "mm" on fp-ALU0, due to the increased threshold. As there are large amount



(a) int-ALU0



(b) int-ALU1

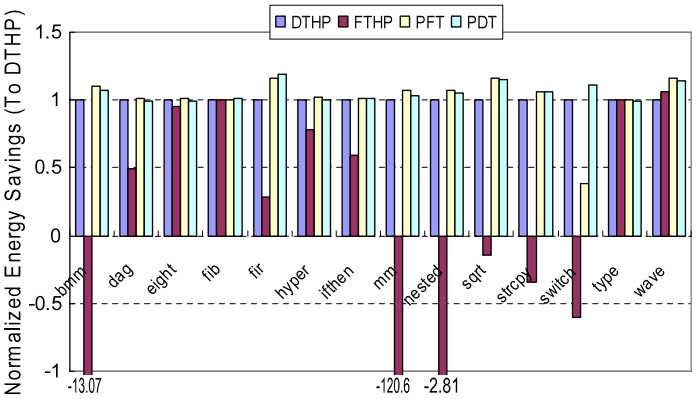
Fig. 6 Leakage energy savings for each int-ALU. (Comparing FTHP, DTHP, PFT and PDT, normalized to DTHP, using parameters in Table 1 and $T_{\text{threshold2}} = 40$)

of idle cycles on fp-ALU1, fp-ALU2 and fp-ALU3 in the experiments, all approaches can utilize most of the opportunities and work well. Still, our two algorithms work better than FTHP and DTHP in general.

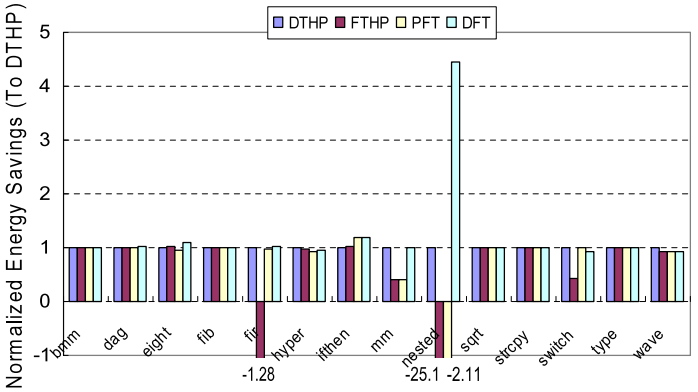
6.2.2 Performance comparisons with various parameters

To show the influences from the threshold and break-even point, we conduct various experiments with different parameters. The experimental results related to floating-point ALU0 are shown in Fig. 8 where all results are normalized to the results of DTHP. Compared to the other FUs, the idle periods of fp-ALU0 provide more power gating opportunities, so the impact of various parameters is highlighted.

In Fig. 8(a), the results (normalized to the leakage energy savings of PFT) are obtained by varying the threshold with the other parameters fixed. Since DTHP sets an upper bound for the threshold, the corresponding $T_{\text{Threshold}}$ has a maximum value as 12. We can see that



(c) int-ALU2



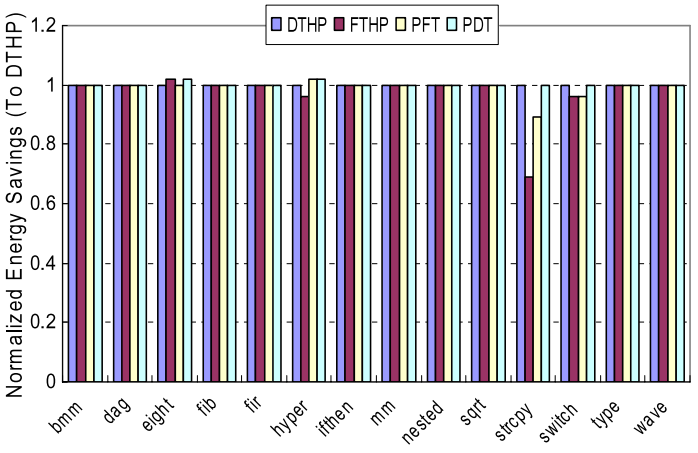
(d) int-ALU3

Fig. 6 (*Continued*)

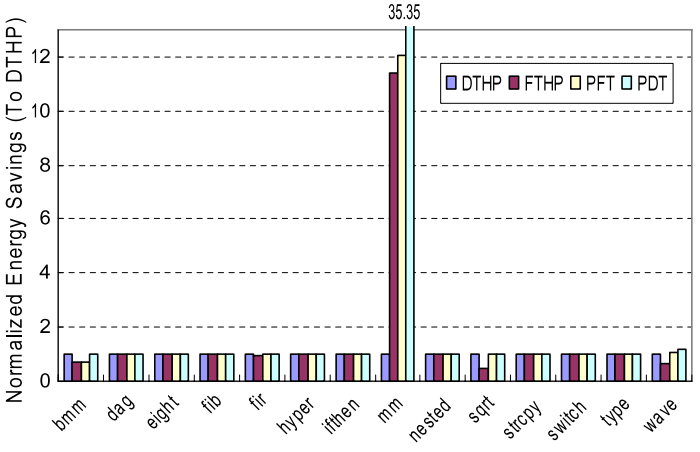
with a small threshold, PDT works much better than PFT and DTHP. And the results get closer from PDT and PFT as the threshold increases. The similar trend can be observed from Figs. 8(b)–(c), in which the break-even points are increased while the other parameters are fixed in Fig. 8(b), and both the threshold and break-even points are increased in Fig. 8(c). It is because there are a large number of medium-sized idle periods, ranging from 15 to 35 cycles in the benchmarks. Therefore, when the threshold or break-even point is more than 35 cycles, these idle periods cannot be optimized by all algorithms. This set of experiment indicates that a small initial threshold may improve the performance of PDT.

6.3 Performance degradation analysis

For each idle period, if power gating is applied to FU, it costs some time for recharging the circuits of the FU when the FU is required again. Thus for both correct predictions and wrong predictions, the waking-up processes prolong the whole execution time. Such time overhead is called as performance loss. In this section, we investigate the performance loss when the proposed approaches are applied. In the experiments, we set the time for waking up the units to 3 cycles as in [15]. The results of performance analysis for PFT and PDT



(a) int-MUL

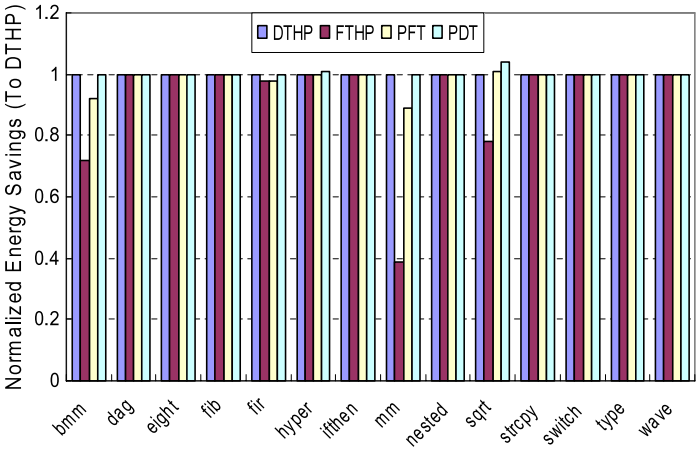


(b) fp-ALU0

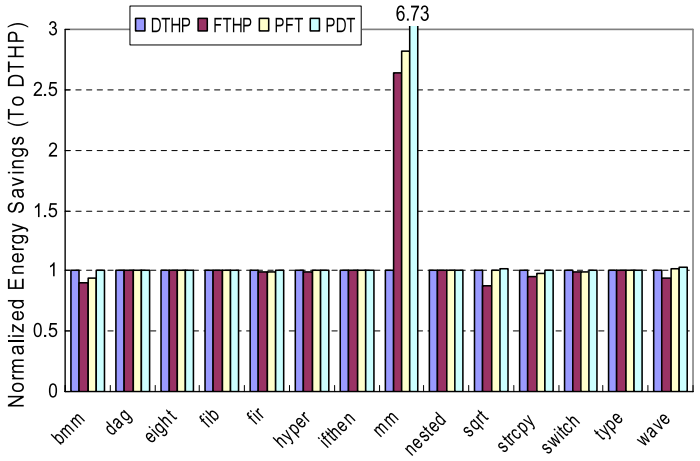
Fig. 7 Leakage energy savings for int-Mul, fp-ALU0, fp-Mul and average results of int-Mul and all fp-FUs. (Comparing FTHP, DTHP, PFT and PDT, normalized to DTHP, using the parameters in Table 1 and $T_{\text{threshold2}} = 40$)

are depicted in Figs. 9(a) and (b), respectively. The percentages of the cycles in sleep mode from all idle cycles are also reported for comparisons.

According to the experimental results, all the functional units are separated into 3 groups. For both PFT and PDT, it is hard to apply power gating for group 1 (int-ALU0, int-ALU1 and int-ALU2), due to large amount of small idle periods. Accordingly, the performance loss is smaller than 0.3%. For other functional units, PFT and PDT shut down more than 88% of idle cycles with a performance loss of 3.2% on average. For group 2 (int-ALU3, fp-ALU0 and fp-Mul/Div), the performance loss is relatively large. It is because these units produce large amount of medium-sized idle periods, which increases the number of power gating actions as mentioned above. And it translates into more cycles for waking-up. For the



(c) fp-MUL



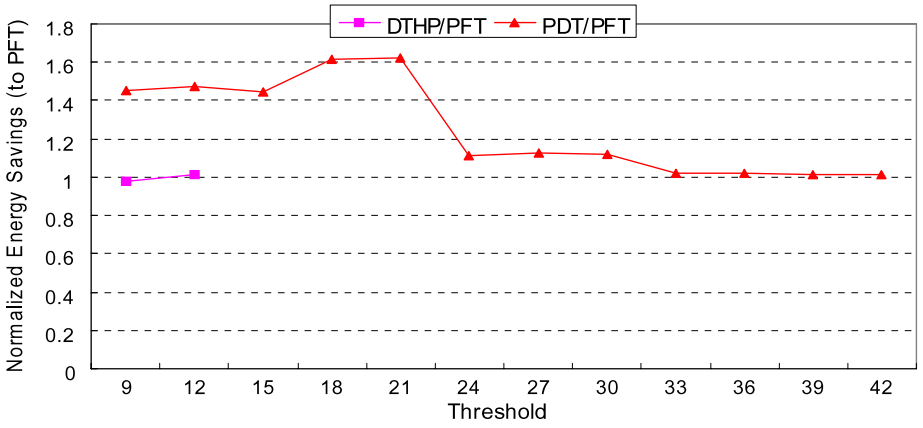
(d) Average

Fig. 7 (*Continued*)

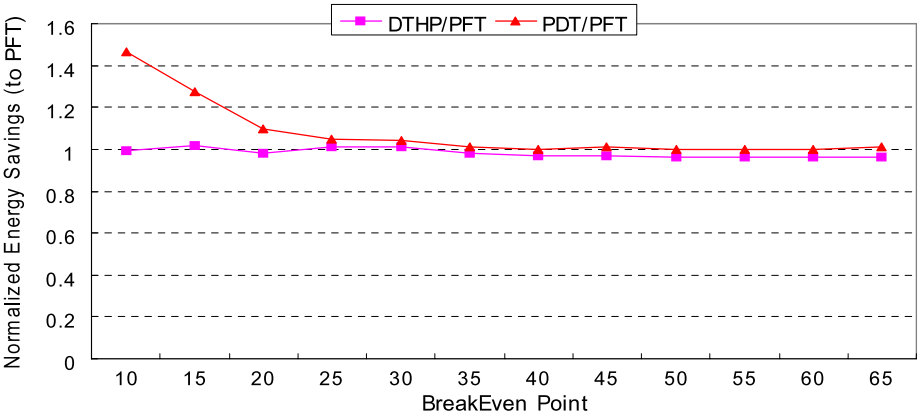
seldom used FUs in group 3 (int-Mul/Div and other 3 fp-ALUs), most of the idle periods are very large and in a very small number. By applying power gating for these units, the proposed approaches save the most leakage energy with the least performance loss. On the other hand, we also find that PDT detects more power gating opportunities than PFT. Therefore, with more energy savings, the overhead increases accordingly.

6.4 Power overhead analysis

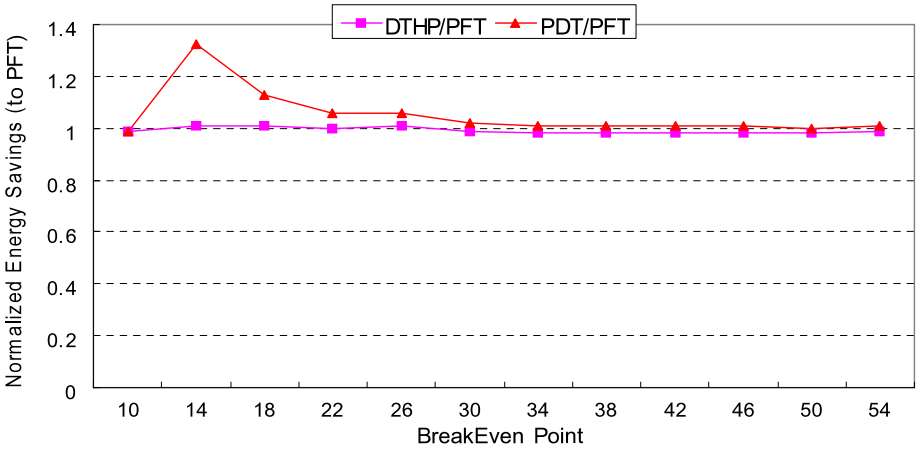
Our approach is based on simple circuit and can be easily implemented into various processors. Figure 10 shows a block-level diagram for the implementation. Adopting a similar design as [43], the circuit to implement PDT is very simple and its power consumption should consume no more than 600 μW. Take the dual core UltraSPARC microprocessor for exam-



(a) $T_{Threshold}$ varies, and $T_{BreakEven}$=14.

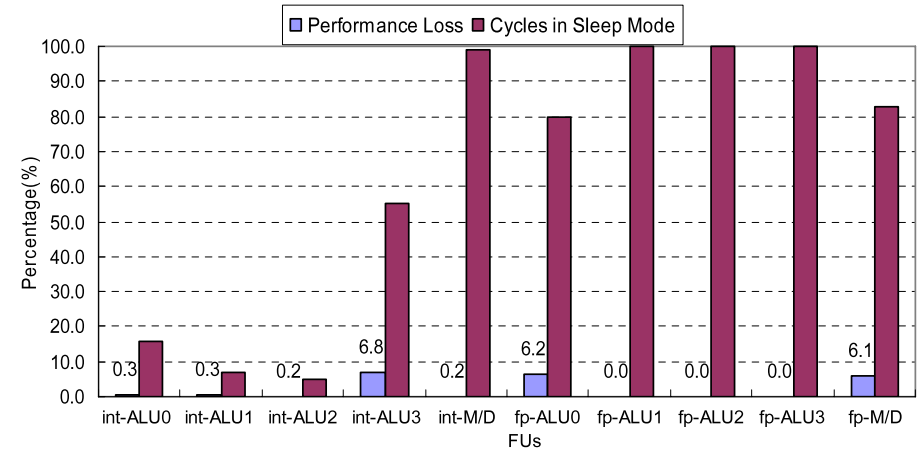


(b) $T_{BreakEven}$ varies, and $T_{Threshold}$=10.

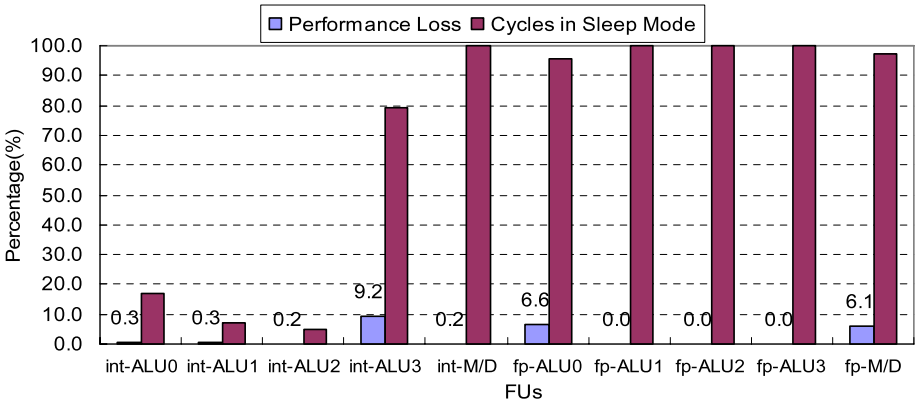


(c) $T_{BreakEven}$ varies, $T_{Threshold}$= $T_{BreakEven}$ / 2.

Fig. 8 Average energy savings for fp-ALU0 (Normalized to PFT. $T_{HitLitmit} = 20$, $T_{StepHit} = 1$, $T_{MistakeLimit} = 4$, $T_{StepMiss} = 2$, and $T_{Threshold2} = 40$)



(a) PFT



(b) PDT

Fig. 9 Performance degradation versus percent of cycles in sleep mode using the parameters in Table 1, $T_{\text{threshold2}} = 40$, and $T_{\text{wakeup}} = 3$

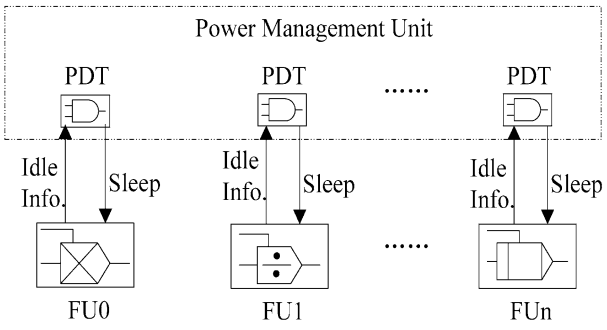


Fig. 10 The implementation of our approach

ple, the power consumption of a modern 64-bit ALU consumes approximate 400 mW [36]. Therefore, our approach has very small impact in the total energy (less than 0.2% on the power budget of a processor). Compared with the energy savings achieved by our approach, the energy overhead introduced by our approach can be ignored.

7 Conclusion and future work

In this paper, we proposed a novel predictive approach for the power leakage reduction. In our approach, we used a state-based predictor to store the historic operational profile information of functional units and to predict the leakage saving opportunities. Based on the predictor, we proposed two prediction algorithms with fixed and dynamic threshold, respectively. We implemented our techniques in SimpleScalar and conducted experiments based on a set of benchmarks from Trimaran. The experimental results show that our algorithms can effectively reduce energy savings compared to the previous work. The predictive techniques in this paper are especially useful for embedded applications which have specific idleness patterns. For a specific application, certain idleness sequences would be generated. For each such sequence, we could vary the parameters for a better result of leakage savings. In embedded systems, it is an important problem to reduce leakage energy caused by on-chip storage, since on-chip memory occupies a big portion of the silicon die. In our future work, we will extend our approach to solve this problem.

References

1. Abdollahi A, Fallah F, Pedram M (2002) Minimizing leakage current in VLSI circuits. Technical Report, Department of Electrical Engineering, University of Southern California, No. 02-08, May 2002
2. Abdollahi A, Fallah F, Pedram M (2002) Runtime mechanisms for leakage current reduction in CMOS VLSI circuits. In: Proc international symposium on low power electronics and design, August 2002
3. Agarwal K, Deogun H, Sylvester D, Nowka K (2006) Power gating with multiple sleep modes. In: 7th international symposium on quality electronic design, 27–29 March, 2006, p 5
4. Babighian P, Benini L, Macii A, Macii E (2004) Post-layout leakage power minimization based on distributed sleep transistor insertion. In: Proceedings of the 2004 international symposium on low power electronics and design, Aug 2004, pp 138–143
5. Benini L, Bogliolo A, Paleologo GA, De Micheli G (1999) Policy optimization for dynamic power management. IEEE Trans Comput-Aided Des Integr Circ Syst 18(6)
6. Chang C, Yang W, Huang C, Chien C (2007) New power gating structure with low voltage fluctuations by bulk controller in transition mode. In: IEEE international symposium on circuits and systems, 27–30 May, 2007, pp 3740–3743
7. Chen J-J, Kuo T-W (2005) Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In: 26th IEEE real-time systems symposium (RTSS)
8. Chen J-J, Kuo T-W (2006) Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scalingprocessor. In: 2006 ACM SIGPLAN/SIGBED conference on language, compilers and tool support for embedded systems, pp 153–162
9. Chen J-J, Kuo T-W (2006) Allocation cost minimization for periodic hard real-time tasks in energy-constrained dvs systems. In: Proceedings of the 2006 IEEE/ACM international conference on computer-aided design, pp 255–260
10. Chung E-Y, Benini L, Bogliolo A, Lu Y-H, De Micheli G (2002) Dynamic power management for nonstationary service requests. IEEE Trans Comput 51(11)
11. Devgan A, Narendra S, Blaauw D, Najm F (2003) Leakage issues in IC design: trends, estimation and avoidance. In: Proceedings of ICCAD
12. Dropsho SG, Kursun V, Albonesi DH, Dwarkadas S, Friedman EG (2002) Managing static leakage energy in microprocessor functional units. In: Micro-annual workshop then annual international symposium, vol 35, pp 321–332

13. Gniady C, Butt AR, Hu YC, Lu Y-H (2006) Program counter-based prediction techniques for dynamic power management. IEEE Trans Comput 55(6):641–658
14. Hennessy JL, Patterson DA (1996) Computer architecture: a quantitative approach, 2nd edn. Morgan Kaufmann, San Mateo
15. Hu Z, Buyuktosunoglu A, Srinivasan V, Zyuban V, Jacobson H, Bose P (2004) Microarchitectural techniques for power gating of execution units. In: Proceedings of the international symposium on low power electronics and design, pp 32–37
16. http://www.trimaran.org/
17. http://www.simplescalar.com/
18. Johnson MC, Somasekhar D, Roy K (1999) Leakage control with efficient use of transistor stacks in single threshold CMOS. In: Design automation conference, Proceedings 36th, 21–25 June 1999, pp 442–445
19. Kao JT, Chandrakasan AP (2000) Dual-threshold voltage techniques for low-power digital circuits. IJSSC 35(7):1009–1018
20. Kao J, Chandrakasan A (2000) Dual-threshold voltage techniques for low-poer digital circuits. IEEE J Solid-State Circ 35:1009–1018
21. Keshavarzi A, Ma S, Narendra S, Bloechel B, Mistry K, Ghani T, Borkar S, De V (2001) Effectiveness of reverse body bias for leakage control in scaled dual-V_t CMOS ICs. In: Proc int symp low power electronics and design. Huntington Beach, CA, pp 207–212
22. Kong F, Tao P, Yang S, Zhao X (2006) Genetic algorithm based idle length prediction scheme for dynamic power management. In: IMACS multiconference on computational engineering in systems applications, Oct 2006, pp 1437–1443
23. Krishnamurthy R, Alvandpour A, De V, Borkar S (2002) High-performance and low-power challenges for sub-70-nm microprocessor circuits. In: Proc custom integrated circuits conf, pp 125–128
24. Kursun V, Friedman EG (2002) Low swing dual threshold voltage domino logic. In: 12th great lakes symposium on VLSI, April 2002
25. Luo J, Jha NK (2007) Power-efficient scheduling for heterogeneous distributed real-time embedded systems. IEEE Trans Comput-Aided Des 26:1161–1170
26. Luo J, Jha NK, Peh L-S (2007) Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. IEEE Trans VLSI Syst 15:427–437
27. Mutoh S, Douseki T, Matsuya Y, Aoki T, Shigematsu S, Yamada J (1995) 1-V power supply high-speed digital circuit technology with multithreshold voltage CMOS. IEEE J Solid-State Circ 30:847–854
28. Narendra S, Borkar S, De V, Antoniadis D, Chandrakasan A (2001) Scaling of stack effect and its application for leakage reduction. In: Proc int symp low power electronic design (ISLPED), pp 195–200
29. Pan L, Yang Y, Wang M, Shao Z (2008) A state-based predictive approach for leakage reduction of functional units. In: IEEE/IFIP international conference on embedded and ubiquitous computing, Dec 2008, pp 52–58
30. Park JC, Mooney VJ III (2006) Sleepy stack leakage reduction. IEEE Trans Very Large Scale Integr (VLSI) Syst 14(11)
31. Raghavan SV, Swaminathan N, Srinivasan J (1999) Predicting behavior patterns using adaptive workload models. In: 7th international symposium on modeling, analysis and simulation of computer and telecommunication systems, 24–28 Oct 1999, pp 226–233
32. Rele S, Pande S, Onder S, Gupta R (2002) In: Optimizing static power dissipation by functional units in superscalar processors. Lecture notes in computer science, vol 2304. Springer, Berlin, pp 261–276
33. Roy K, Mukhopadhyay S, Mahmoodi-Meimand H (2003) Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. Proc IEEE 91(2):305–327
34. Simunic T, Benini L, Glynn P, De Micheli G (2000) Dynamic power management for portable systems. In: International conference of mobile computing and networking. ACM Press, New York
35. Swaminathan N, Srinivasan J, Raghavan SV (1999) Bandwidth-demand prediction in ATM networks using genetic algorithms. Comput Commun 22:1127–1135
36. Takayanagi T, Shin JL, Petrick B, Su JY, Levy H, Pham H, Son J, Moon N, Bistry D, Nair U, Singh M, Mathur V, Leon AS (2005) A dual-core 64-bit UltraSPARC microprocessor for dense server applications. IEEE J Solid-State Circ 40(1):7–18
37. Tschanz J, Ye Y, Wei L, Govindarajulu V, Borkar N, Burns S, Karnik T, Borkar S, De V (2002) Design optimizations of a high-performance microprocessor using combinations of dual-V_t allocation and transistor sizing. In: Symp VLSI circuits dig tech papers, pp 218–219
38. Tschanz JW, Narendra SG, Ye Y, Bloechel BA, Borkar S, De V (2003) Dynamic sleep transistor and body bias for active leakage power control of microprocessors. IEEE J Solid-State Circ 38(11)
39. Wei L, Chen Z, Johnson MC, Roy K, De V (1998) Design and optimization of low voltage high performance dual threshold CMOS circuits. In: ACM/IEEE design automation conf, pp 489–494

40. Wei L, Roy K, Ye Y, De V (1999) Mixed-Vth (MVT) CMOS circuit design methodology for low power applications. In: ACM/IEEE design automation conf, pp 430–435
41. Wei L, Roy K, Vivek KD (2000) Low voltage low power CMOS design techniques for deep submicron ICs. In: Proceedings of the international conference on VLSI design, pp 24–29
42. Ye Y, Borkar S, De V (1998) A new technique for standby leakage reduction in high-performance circuits. In: Symp VLSI circuits dig tech papers, pp 40–41
43. Youssef A, Anis M, Elmasry M (2006) Dynamic standby prediction for leakage tolerant microprocessor functional units. IEEE Computer Society, Washington, pp 371–384
44. Zhong X, Xu CZ (2005) Energy-aware modeling and scheduling of real-time tasks for dynamic voltage scaling. In: Proc of IEEE real-time symposium (RTSS'05), pp 366–375
45. Zhong X, Xu C-Z (2006) System-wide energy minimization for real-time tasks: lower bound and approximation. In: IEEE/ACM international conference on computer-aided design (ICCAD), pp 516–521
46. Zhou D, Hu J, Wang L (2007) Design of adiabatic sequential circuits using power-gating technique. IEEE northeast workshop on circuits and systems, 5–8 Aug 2007, pp 952–955