Maintaining Performance on Power Gating of Microprocessor Functional Units by Using a Predictive Pre-Wakeup Strategy

CHANG-CHING YEH, National Chung Cheng University
KUEI-CHUNG CHANG, Feng Chia University
TIEN-FU CHEN, National Chiao Tung University
CHINGWEI YEH, National Chung Cheng University

Power gating is an effective technique for reducing leakage power in deep submicron CMOS technology. Microarchitectural techniques for power gating of functional units have been developed by detecting suitable idle regions and turning them off to reduce leakage energy consumption; however, wakeup of functional units is needed when instructions are ready for execution such that wakeup overhead is naturally incurred. This study presents time-based power gating with reference pre-wakeup (PGRP), a novel predictive strategy that detects suitable idle periods for power gating and then enables pre-wakeup of needed functional units for avoiding wakeup overhead. The key insight is that most wakeups are repeated due to program locality. Thus, the pre-wakeup predictor learns the wakeup events and selects which prior branch instruction can provide early wakeup (wakeup patterns are visible); these information are then used to adequately prepare available functional units for instruction execution. Simulation results with benchmarks from SPEC2000 applications show that substantial leakage energy reduction with negligible performance degradation (0.38% on average) is worthwhile.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Pipeline processors*

General Terms: Design, Performance

Additional Key Words and Phrases: Low power, leakage-power reduction, power gating

ACM Reference Format:
Yeh, C.-C., Chang, K.-C., Chen, T.-F., and Yeh, C. 2011. Maintaining performance on power gating of microprocessor functional units by using a predictive pre-wakeup strategy. ACM Trans. Architec. Code Optim. 8, 3, Article 16 (October 2011), 27 pages.
DOI = 10.1145/2019608.2019615 http://doi.acm.org/10.1145/2019608.2019615

1. INTRODUCTION

In deep-submicron process, leakage power accounts for a considerable fraction of chip power consumption and is a key consideration in chip design. Power gating is an effective method for detecting idle functional units and switching them to a low-leakage state. However, a practical design concern is the significant time required to wake up functional units since sleep transistors must be switched on to initially discharge

Authors' addresses: C.-C. Yeh, Department of Electrical Engineering, National Chung Cheng University, Min-Hsiung Chia-Yi, Taiwan; email: yeh.changching@gmail.com; K.-C. Chang, Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan; email: changkc@fcu.edu.tw; T.-F. Chen, Department of Computer Science, National Chiao Tung University, Taiwan; email: tfchen@cs.nctu.edu.tw; C. Yeh, Department of Electrical Engineering, National Chung Cheng University, Taiwan; email: ieecwy@ccu.edu.tw.
Additional emails: eye2001@dragon.ccut.edu.tw; rainer_mcu@hotmail.com.

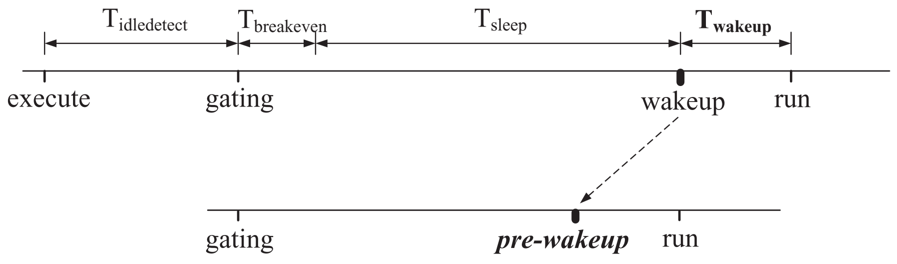


Fig. 1. Pre-wake up functional unit to avoid wakeup overhead.

virtual ground capacitance [Kao et al. 1997; Bhunia et al. 2005]. Meanwhile, ground bounce [Pant et al. 1999] arising from power mode transitions in power gating structures greatly affects the reliability of surrounding circuit blocks. Shortening wakeup time therefore complicates the suppression of ground bounce [Abdollahi et al. 2007; Lee et al. 2008]. In addition to process technology, most runtime reduction techniques target circuit-level optimizations and incur performance overhead penalties in order to reduce leakage power. Since most leakage reduction techniques focus only on parts of microprocessors (e.g., functional units or caches), performance degradation may be compounded when multiple processor components reduce leakage current simultaneously. To mitigate such penalties, a fine-grained control technique is needed for efficient exploitation of idle intervals in order to reduce leakage power without degrading performance [Chung and Skadron 2008]. To avoid the performance penalty incurred by wakeup latency, this work presents a method of predicting impending executions and then pre-waking up the required functional units at runtime.

A recent development in dynamic power gating is *time-based power gating*, in which a functional unit with idle time exceeding a predetermined threshold is predicted to have sufficiently long idle time to make power gating worthwhile [Hu et al. 2004]. Idle functional units are typically put into sleep mode periodically, and wakeup of a functional unit is only needed when instructions are ready for execution. Whenever power gating prevents functional units from executing instructions, the processor incurs wakeup overhead that increases execution time. Figure 1 shows the details of power gating in a typical execution example. If a functional unit is not executed for consecutive $T_{idledetect}$ cycles, power gating is enabled to minimize leakage energy. Compensation for the dynamic energy overhead generated by power gating requires $T_{breakeven}$ cycles, and net leakage savings can be achieved in T_{sleep} cycles. Any ready instructions stop power gating and wake up the needed functional units in T_{wakeup} cycles. Because wakeup of a functional unit may also delay processor execution, increased program execution time can increase processor energy consumption. Although leakage savings in $T_{breakeven}$ cycles compensates for dynamic energy overhead generated by power gating, the impact of wakeup latency on other processor design metrics (performance and total energy) must be minimized. Therefore, early wakeup of required functional units is preferable. Practically, compiler-based leakage reduction technique inserts suitable wakeup instructions in advance to prepare for the instruction execution. Similarly, an effective microarchitectural solution is also needed to provide a preventive pre-wakeup mechanism for executing instructions on-demand.

An effective power gating mechanism must have the capability to provide early wakeup without stalling program execution. This study proposes a technique for efficiently predicting which functional units are likely to be used. Recently executed instructions are assumed to be the instructions most likely to be executed in the near future (i.e., temporal locality). Thus, past execution behavior is analyzed to predict future needs. Based on the above premise, the pre-wakeup predictor identifies recurrent wakeup activities during runtime and provides early wakeup to avoid incurring the same wakeup overheads. As Figure 1 shows, power gating does not delay execution as wakeup is early enough before functional units are actually needed.

The proposed pre-wakeup predictor anticipates wakeup activities by using *control-flow events*. Essentially, control flow instructions determine which instructions should subsequently be executed and therefore are suitable to identify recurrent execution of a loop. Hence, an early wakeup could be determined by the prior branch instructions. The proposed heuristic for limiting the control-flow information sent to the pre-wakeup predictor focuses on the most recent branch instructions that were executed leading up to a wakeup overhead. As soon as a new wakeup occurs, the predictor captures the control-flow event by storing the most recent branch address and its direction. Subsequent executions of the same branch instruction cause the predictor to check for wakeup need. This approach requires an extra memory to capture control-flow information. Fortunately, for most idle periods, time-based approach with a suitable detection threshold prevents inefficient power gating and also avoids corresponding wakeup activities whereas the pre-wakeup predictor can service remaining wakeup events by providing a very small memory.

Leakage reduction techniques inevitably incur a performance penalty. Traditional time-based approaches that only have high idle detection thresholds prevent most wakeup activities. The proposed predictor allows the time-based approaches to have low thresholds for efficient power gating without delaying program execution. This work makes the following contributions.

—While time-based solutions are adequate for intermittent power-gating, this study shows that recurrent wakeup overheads can significantly degrade performance. Idle interval distributions of SPEC2000 benchmarks are analyzed to show why time-based power gating schemes should consider recurrent wakeup overheads.
—*Pre-wakeup prediction*. Wakeup can be determined by using a control-flow event (branch address and its direction) and, before an impending execution, wakeup of functional units can be predicted with over 90% accuracy.
—*Time-based Power Gating with Reference Pre-wakeup* (*PGRP*). Time-based power gating scheme can prevent most wakeup activities behind short idle periods; therefore, the proposed pre-wakeup table requires very little memory (32-entry content addressable memory) to store wakeup information.

The remainder of this article is organized as follows. Section 2 discusses related work. Section 3 describes the experimental environment. Section 4 then presents a sample distribution of idle periods and finds inherent wakeup overheads arising from time-based power gating. Next, Section 5 discusses the architecture of the proposed PGRP scheme. Section 6 summarizes the simulation results. Conclusions are finally drawn in Section 7.

2. RELATED WORK

2.1. Ground Bounce Noise and Wakeup Time

Switching current may cause a ground bounce noise due to the substantial parasitic inductance in the power-supply network. Grochowski et al. [2002] implemented a di/dt controller on the chip by using a current-to-voltage computation to estimate supply voltage variations and controlling the microprocessor's activities upon detection of a voltage violation. Typically, ground bounce noise can be significant as sleep transistors are turned on within a very short time to minimize performance overhead. Low supply voltage in CMOS technology scaling decreases noise margin and further limits the ground bounce allowed by the chip.

To reduce the magnitude of voltage glitches in the power distribution network as well as the time required for network stabilization, Kim et al. [2003] proposed two novel power-gating structures, in which the first turns on multiple sleep transistors in a nonuniform stepwise manner (parallel sleep transistors), and the second structure turns a single sleep transistor on gradually (staircase sleep signal). Abdollahi et al. [2007] proposed a wakeup scheduler to partition logic cells into different clusters and schedule intercluster sleep signals to minimize wakeup time while limiting the instantaneous supply/ground current. Calimera et al. [2009] minimizes reactivation time by sizing the parallel sleep transistors according to an optimal algorithm and introducing a row-based physical implementation to minimize layout disruption and routing congestion.

Approaches for tolerating wakeup latency overhead include the power-gating circuit proposed by Agarwal et al. [2006], in which different gate biases are applied to NMOS footer devices. The resulting differences in V_{GND} potentials enable different wakeup latencies. A processor can select the optimum sleep mode based on the number of idle cycles and wakeup latencies. Singh et al. [2007] partitioned the logic datapath into different blocks and exploited multimode power gating based on block position in the datapath flow. Deep logic circuits in a datapath have particularly high wakeup margins and can therefore be gated strongly. Dropsho et al. [2002] provided an analytical energy model for the dual-threshold-voltage dynamic domino logic circuits in the integer functional units of a processor to determine suitable sleep-mode activation policies. Sleep mode may not be advantageous if the integer functional units are dominated by the distribution of short idle periods; hence, Dropsho et al. proposed a gradual sleep policy in which successive portions of the functional unit enter sleep mode when the functional unit is idle.

2.2. Early Wakeup Control Policies

Simultaneous control of ground bounce and wakeup time increases circuit complexity (e.g., a finite-state machine needed for scheduling algorithm). Knowledge of future execution patterns provides accurate prediction of functional units required in the near future, which thereby provides sufficient wakeup time for efficient control of ground bounce.

Compiler-level approaches perform data-flow analysis to identify program regions in which idle functional units can be exploited. Activate/deactivate instructions are inserted into the code to set/reset a sleep signal that controls leakage current from functional units [You et al. 2006]. The benefit is that wakeup overhead is reduced by placing wakeup instructions far in advance of ready instructions whenever possible; hence, wakeup can be early enough according to predictions (preactivated), and performance loss is negligible. Nevertheless, the number of additional power-control instructions in application programs is problematic because rich components are equipped with power-gating controls in system-on-a-chip (SoC) design platforms. You et al. [2007] developed a compact control framework that reduces the number of power-gating instructions by combining several power-gating instructions into a single compound instruction.

Microarchitectural-level approaches have difficulty for predicting the future use of functional units. Mohamood et al. [2006] used a hardware table to quantify current demand after decode stage for clock gating of idle functional units. A simple decay counter detects suitable idle periods for preemptive turn-on of clock-gated ALU through pre-decoding instructions. To maintain performance while controlling leakage current from the instruction cache, Chung and Skadron [2008] proposed an on-demand wakeup prediction policy for selectively pre-waking required instruction cache lines according

Table I. The Baseline Configuration

Fetch Queue	8 entries
Fetch/Decode	8 instructions per cycle
Issue/Commit	8 instructions per cycle
functional units	4 integer alu, 1 integer mult 4 FP alu, 1 FP mult
Branch Predictor	gshare, 16K table, 14-bits history
L1 ICache	64K, 2-way, 32B
L1 DCache	64K, 2-way, 32B
L2 Cache	1MB, 4 way, 6 cycles hit latency
TLB size	ITLB: 16 set 4 way DTLB: 32 set 4 way 4KB page size, 30 cycles penalty
Memory	8 bytes/line, virtual memory 4 KB pages

to branch prediction information. This approach not only achieves excellent leakage energy reduction comparable to that of the optimal policy, it also consistently achieves near-optimal performance (only 0.08% performance overhead on average).

This study analyzed the idle interval distribution of each benchmark to explore the inherent wakeup characteristic. A predictive method is then used to provide early wakeup while also maximizing sleep time.

3. SIMULATION METHODOLOGY

Before addressing the power-performance trade-offs of power gating, the machine architecture, simulation method, and benchmarks used in this study are described in this section. The machine model assumes a multiissue architecture composed of four integer ALUs, one integer MUL, four floating ALUs, and one floating MUL. This study used the *SimpleScalar/Alpha* toolset with *Wattch* power extensions [Brooks et al. 2000]. The CPU is an in-order processor. Unused functional units are power gated for leakage energy reduction. This study applied the activity-sensitive power model with aggressive nonideal conditional clocking (cc3 mode). The assumptions were that leakage power in *Wattch* comprises 10% of total power consumption, and PGRP can still achieve energy reduction even when leakage power consumption is increased in deep submicron technology (as discussed in Section 6.7). During wakeup, the voltage fluctuations in the power network were minimized by assuming that wakeup requires 20 cycles,[1] even if latency is overestimated.

Table I presents experimental parameters. This study simulated SPEC2000 benchmarks. All benchmarks were simulated using reference inputs. All tests were performed by fast-forwarding the first 500M instructions and then performing detailed simulations for the next 500M instructions.

4. POWER-GATING CONSIDERATIONS

The leakage-control mechanism is effective only if it provides sufficient power-gating time without excessively increasing in runtime. To observe wakeup behavior for different application benchmarks, this section profiles the idle interval distribution in each functional unit to estimate when the processor may be stalled. The performance penalty resulting from time-based power gating is discussed next.

[1]For example, wakeup time with 8–10 cycles is used in Singh et al. [2007]. Meanwhile, the propagation time required for a sleep signal should also be considered.

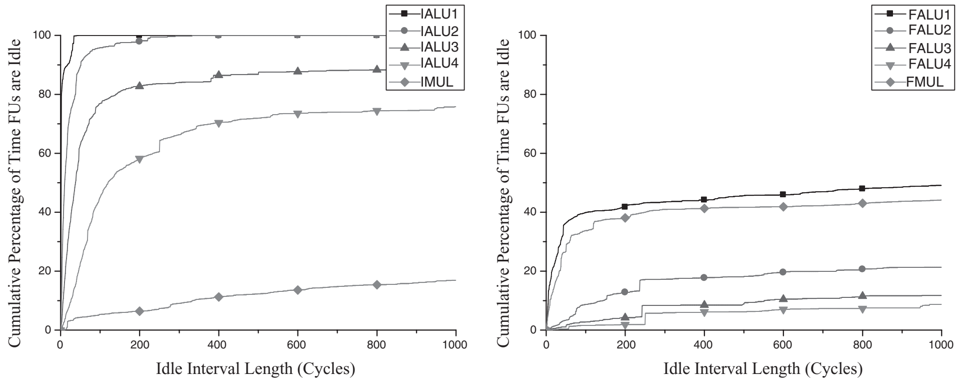


Fig. 2. Average cumulative percentage of idle time of functional units as a function of idle interval length for all SPEC2000 programs.

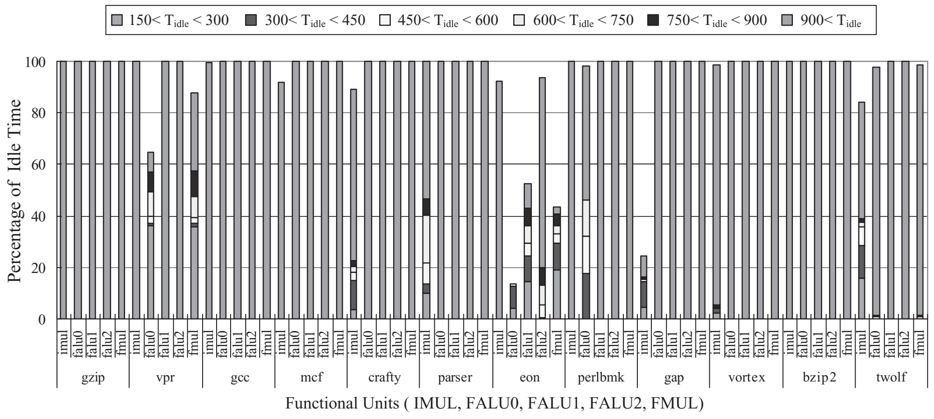
4.1. Idle Interval Distribution of Functional Units

Significant energy savings can be realized if functional units can power down to a low-power sleep state during idle periods between intermittent executions; however, the frequent transitions from sleep to active mode adversely affect the performance. The performance penalty resulting from recurrent wakeup overheads was assessed in this study by evaluating the varying distribution of idle intervals needed to manage power gating in each functional unit. Measuring execution complexity in each functional unit according to idle time is a simple and effective way to evaluate existing power gating techniques.

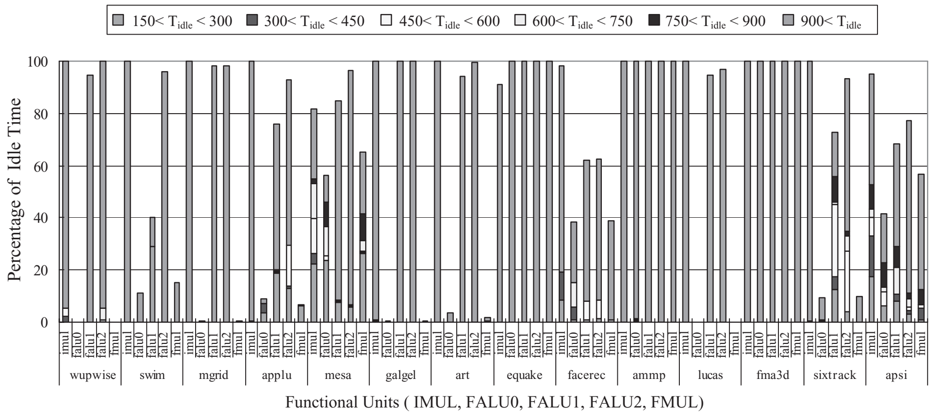
Figure 2 shows the average distribution of idle periods in each functional unit across the benchmarks. The x-axis is the idle period length and the y-axis is the average cumulative percentage of time functional units are idle over the total execution time for all SPEC2000 programs. The left figure shows the percentage of idle time in a series of integer ALUs (IALUs) and an integer MUL (IMUL), and the right figure shows the percentage for a series of floating ALUs (FALUs) and a floating MUL (FMUL). The two figures illustrate how idle periods can be power gated in functional units by scavenging idle time.

In the first integer ALU (IALU1), extremely long idle periods are rare, and relatively short periods are common. The graph shows that IALU1 is idle 85% of the time when idle time is shorter than five cycles; clearly, most instructions are executed by IALU1 and therefore prevent it from entering sleep mode. Wakeup overheads incurred by integer IALUs are relatively smaller than those incurred by other functional units since the instructions look forward to wake up a power-gated IALU can be changed to wait for other idle/busy IALUs [Hu et al. 2004]. IMUL and other floating point units are very likely to have a long idle time, which makes power-gating an effective energy conservation strategy. Finally, the short but sharp step at the idle periods smaller than 150 cycles for the IALUs and FALUs indicates a suitable threshold value ($T_{idledetect} = 150$), that avoids most inefficient power-gating and excessive wakeup activities.

This study determined that wakeup patterns can be classified using the distribution of idle periods to evaluate the potential recurrent wakeup overheads and approximate increases in execution time. Figure 3 presents the distribution of idle periods within each distinct range of idle time length across all SPEC2000 benchmarks. Only representative idle periods T_{idle} (corresponding to at least 150 cycles) are plotted. Idle periods with <150 cycles are omitted because they occur very frequently, and power gating therefore achieves only limited energy savings. The y-axis is the percentage



(a) SPEC2000 INT applications.



(b) SPEC2000 FP applications.

Fig. 3. Distribution of idle periods within each distinct range of idle time length.

of idle time in the range greater than 150 cycles. For brevity, plotted distributions for functional units with low potential wakeup overheads, including all IALUs and FALU3, are omitted.

Clearly, for benchmarks with idle periods shorter than 900 cycles (e.g., *vpr*, *eon*, *twolf*, *mesa*, and *apsi*), the middle periods cannot be controlled by the time-based approach when $T_{idledetect}$ is <150 cycles. Therefore, recurrent wakeup overheads are possible. The number of middle idle periods is insignificant, but heavy wakeup overheads may still be incurred when most functional units are concurrently power gated in the middle periods and the number of idle periods is weighted by the length of wakeup latencies (*e.g., mesa* may have a maximum of 13.53% increased runtime when T_{wakeup} is 20 cycles and $T_{idledetect}$ is 150 cycles). Conversely, although high $T_{idledetect}$ can minimize recurrent wakeup overheads, power gating time is also reduced (e.g., about 96% of idle periods in FALU1, FALU2 and FALU3 for *lucas* is between 150 and 300 cycles, which cannot be power gated when $T_{idledetect}$ is >300 cycles; in Figure 12(b), the leakage energy savings from functional units is also reduced by time-based power gating from 25.7% to 2.41%).

These observations indicate that a pre-wakeup predictor can prevent recurrent wakeup overheads. The time-based approach simply excludes power gating during short idle periods [Hu et al. 2004]. If the periods are somewhere in the middle, PGRP provides efficient power gating without incurring wakeup overheads.

4.2. Limits of Time-Based Power Gating

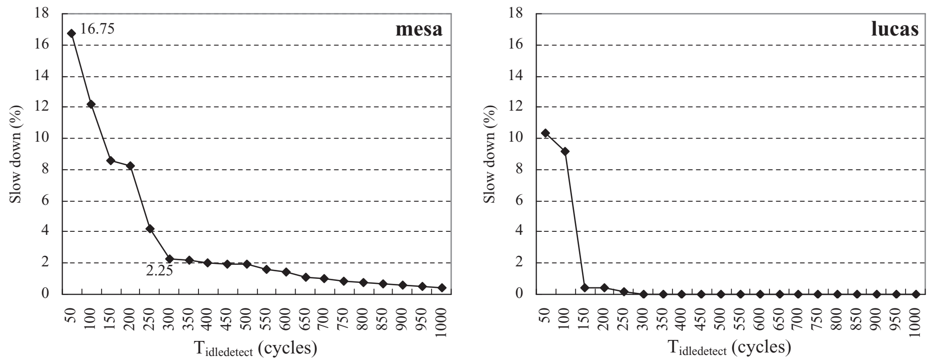
Detecting and exploiting middle idle periods is extremely challenging. In the time-based power gating scheme proposed in Hu et al. [2004], the functional unit is not put to sleep until a certain number of idle cycles (i.e., $T_{idledetect}$) has elapsed since the end of the last execution and is not awakened until it must execute the next instruction. A nonnegligible wakeup time must be taken before functional units begin to execute instructions. Wakeup overhead is incurred if the same functional unit is accessed twice within a time interval exceeding the idle detection threshold. Excessive wakeup overhead can substantially degrade performance. Hence, modern designs provide a conservative detection threshold by increasing $T_{idledetect}$ to avoid most wakeup operations; however, high $T_{idledetect}$ reduces power gating duration, which then impairs the effectiveness of the leakage energy reduction scheme.

Notably, the efficiency of time-based power-gating is greatly decreased by various idle interval distributions of benchmarks. Thus, $T_{idledetect}$ should be sufficiently high to avoid recurrent wakeup overheads but also sufficiently low to detect enough idle periods for power gating. However, a single $T_{idledetect}$ can hardly satisfy the power gating requirements of each application and different processor configurations without wakeup overheads.

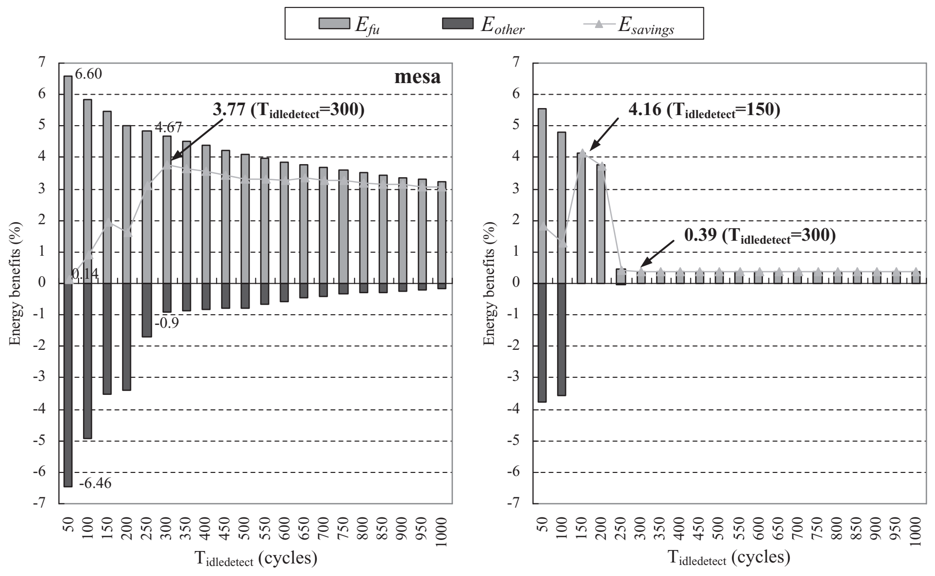
Case studies reveal interesting observations concerning the performance and energy penalties of recurrent wakeup overheads. Consider, for example, a sensitivity analysis based on different $T_{idledetect}$ values for *mesa* and *lucas*. Figure 4 presents the energy benefits of the time-based approach when integer arithmetic logic unit (ALU), integer multiplier, floating-point ALU, and float-point multiplier are considered for power gating. The baseline model used for comparison is that without power-gating control. Energy benefits can be classified into three categories: E_{fu} is the energy savings obtained by power gating in functional units; E_{other} represents the increased leakage energy incurred by all other processor components (excluding functional units); and $E_{savings}$ is the total energy benefits of the processor.

The smallest processor energy benefit of *mesa* ($E_{savings} = 0.14\%$) is obtained when $T_{idledetect}$ is 50 cycles (Figure 4). Notably, energy benefits are calculated for the *entire* processor; that is, processor energy benefits are reduced, and this additional overhead is the overall leakage energy ($E_{other} = -6.46\%$) expended by the processor due to increased runtime (16.75%). Although maximum energy benefits ($E_{fu} = 6.60\%$) are obtained from functional units, the processor cannot realize these benefits. In fact, the energy reduction is even smaller due to increased runtime. Hence, performance cost must be carefully controlled to ensure that the power gating consistently provides energy benefits. $T_{idledetect}$ is assumed to be 50 cycles, which is clearly unsuitable for use in time-based power-gating. By increasing $T_{idledetect}$, performance cost is reduced to roughly 2.25% when $T_{idledetect}$ is 300 cycles. Overall energy benefits are only increased to 3.77% because energy benefits obtained in functional units have decreased to 4.67%.

In the second example, *lucas* obtains maximal energy benefits ($E_{savings} = 4.16\%$) when $T_{idledetect}$ is 150 cycles. Most of the middle idle periods are between 150 and 300 cycles in FALU1, FALU2 (Figure 3(b)), FALU3 and IALU3; therefore, when $T_{idledetect}$ is >200 cycles, these middle idle periods cannot be power gated, and $E_{savings}$ is reduced to 0.39%. Consequently, $T_{idledetect}$ must be low enough to efficiently reduce leakage energy. In the *mesa* example, however, $T_{idledetect}$ should be at least 300 cycles to avoid wakeup overheads. Hence, due to varying idle interval distributions between *mesa* and *lucas*, a major difficulty is choosing a suitable $T_{idledetect}$ that meets the demands of negligible performance overhead while providing high energy benefits for



(a) Slow down due to wakeup latency overhead.



(b) Energy benefits obtained by time-based power gating.

Fig. 4. Evaluation of the time-based power-gating for the benchmarks *mesa* and *lucas*, T_{wakeup} and $T_{breakeven}$ are fixed at 20 cycles. Higher is better.

each benchmark (even for a single benchmark, idle interval distributions may vary within each program section, as discussed in Section 6.1) and for different processor architectures. Thus, an efficient scheme for avoiding wakeup overheads is needed to obtain stable energy benefits.

5. POWER GATING WITH REFERENCE PRE-WAKEUP (PGRP) METHOD

5.1. Motivation

The distribution of wakeup activities may differ as workload phase changes. The sampling window period determines the finest granularity at which phase changes can be resolved. Generally, sampling period must be sufficiently short to capture minute phase changes in workload behavior; however, memory size must also be large enough to store significant wakeup information. Moreover, the time between pre-wakeup and incoming issued instructions varies and is therefore difficult to match to wakeup time (i.e., the pre-wakeup table notices the wakeup changes at any time). Instead, a long sampling window provides much earlier wakeup to increase wakeup space but cannot apply aggressive power gating for even a few idle cycles. The longer sampling window also provides a safe wakeup margin at very low hardware cost. Since branch instructions have strong temporal locality, a branch is likely to be refetched within the last *n* branch instructions in which *n* varies at 8–64 [Baniasadi and Moshovos 2002; Baniasadi 2005]. Hence, in this study, captured wakeup activity is stored in the pre-wakeup table indexed by the prior branch address.[2]

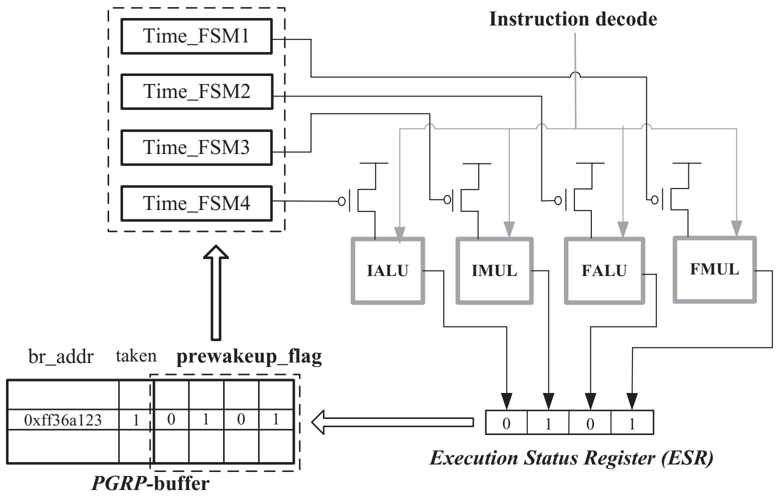
5.2. The Power Gating with Reference Pre-Wakeup Scheme

Power gating is applied by providing one or more header transistors (local power gating switches) for each functional unit (Figure 5(a)). Transistors are turned off in sleep mode to cut off the leakage path. The time-based approach is used for gating control to determine whether power gating is needed. The PGRP-buffer, a very small memory structure used for short-term storage of execution history, is a pre-wakeup table for confirming gating decisions and identifying suitable pre-wakeup time. Execution status in each functional unit is monitored, and the activity information is written into the *Execution Status Register* (*ESR*). Each entry in PGRP-buffer has three fields: a tag or branch address (br_ addr), branch taken flag (taken), and pre-wakeup flags (prewakeup_ flag). Each bit in the pre-wakeup flags records the wakeup decision of a functional unit. The branch taken flag determines which path the current branch takes. The instruction address and its direction can index PGRP-buffer once a branch is executed. If an entry is found, pre-wakeup flags are examined to determine whether any functional unit requires wakeup.

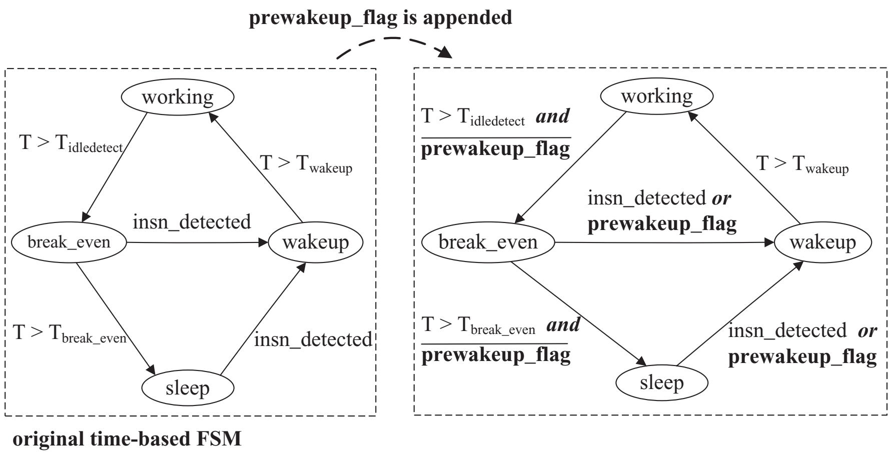
The time-based finite state machine is logically associated with each functional unit [Hu et al. 2004]. Functional units are normally in the working state (Figure 5(a)). When the number of consecutive idle cycles reaches a predetermined threshold ($T_{idledetect}$), power gating is enabled, and the functional unit enters a break_even state and then a sleep state; the leakage savings obtained by power gating in the break_even state compensates for the inherent energy cost of turning the sleep transistor on and off during power mode transition, and net leakage savings can be achieved in sleep state. Any ready instructions assert insn_detected signal to wake up the functional units in wakeup state. In PGRP scheme, the time-based finite state machine is inserted with a prewakeup_flag signal to provide early wakeup control. The predictor collects a history of wakeup activities and uses these information to notify the processor to execute instructions in the near future by asserting a prewakeup_flag signal.

The PGRP-buffer lookup is performed when a branch is being executed. Branch direction is combined with the address of the last-executed branch to form a control event that must be matched with the event in PGRP-buffer. As soon as the predictor observes a control event (i.e., a hit in PGRP-buffer), PGRP wakes up the needed functional unit. The execution activities within each functional unit are monitored by ESR. Once wakeup overhead is incurred, the ESR value is entered into a 32-entry PGRP-buffer (Section 6.4 discusses the impact of varying table size). For each branch execution, PGRP-buffer is read first to determine whether any functional unit requires wakeup. The activity information in the *ESR* is then written to the PGRP-buffer only when wakeup overhead is incurred by a functional unit. An example of the pre-wakeup prediction is illustrated in Figure 6. The initial pre-wakeup prediction operates in five steps described as following:

[2]According to actual simulation result in Figure 12, leakage energy reduced by PGRP and time-based approach is almost the same. PGRP has little impact on the power gating time even though pre-wakeup is controlled by branch instruction.



(a) Power gating with reference pre-wakeup architecture.



(b) Time-based FSM added with pre-wakeup operation.

Fig. 5. Power gating with reference pre-wakeup scheme.

Step1. Both *brancha* address and direction are stored in the branch register.
Step2. If a wakeup overhead is incurred, wakeup information is stored in the ESR register.
Step3. PGRP-buffer Update. When *branchb* is executed, both wakeup information (ESR register) and *brancha* (branch register) are stored in the PGRP-buffer.
Step4. PGRP-buffer Lookup. During the next iteration loop, pre-wakeup information is obtained when *brancha* is executed again.
Step5. Pre-wakeup of needed functional units.

The first three steps capture a wakeup information into PGRP-buffer when a wakeup overhead is incurred. Steps 4 and 5 provide pre-wakeup for required functional units. In this example, *brancha* is executed leading up to a wakeup overhead and therefore is the particular branch associated with pre-wakeup prediction. During next iteration, executing *brancha* provides early wakeup in sequence.

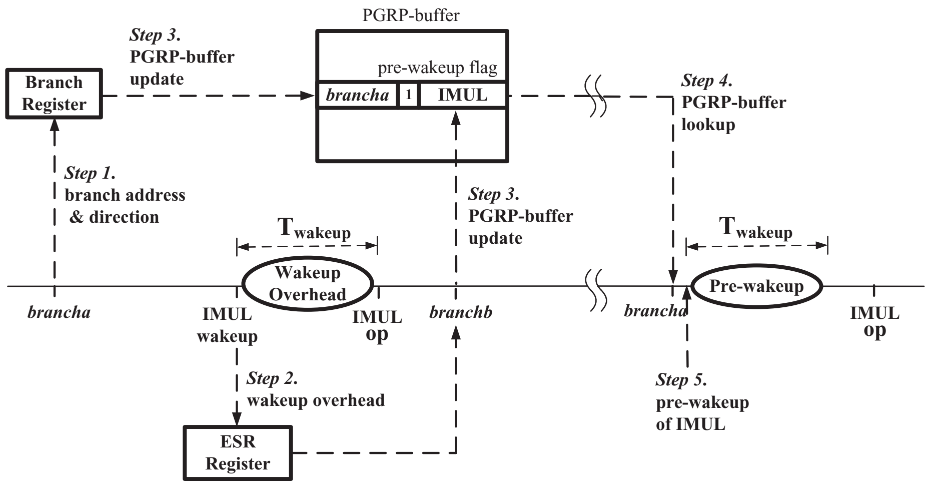


Fig. 6. Example of pre-wakeup prediction.

A large PGRP-buffer has additional area and energy consumption, but reduced table size may not capture sufficient information to predict wakeup. Increasing $T_{idledetect}$ can reduce table size by avoiding most wakeup activities; however, power gating time is reduced. To capture enough pre-wakeup information in a small table while keeping $T_{idledetect}$ sufficiently low to reduce leakage energy efficiently but without impairing performance, the wakeup information is entered into the PGRP-buffer only when wakeup overhead substantially affects processor performance.

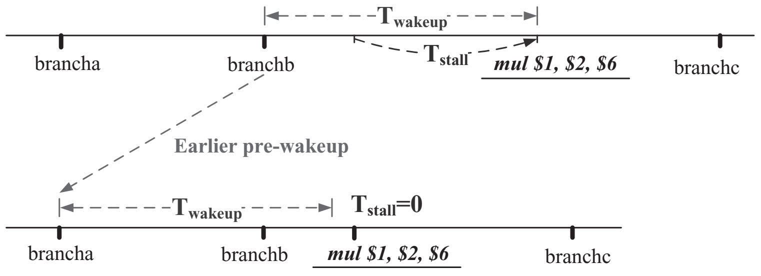
Finally, considering the branch predictor, the most recent branch prediction history is maintained in a prediction history table, which also records the execution history of functional units. However, the branch predictor does not capture the history of two possible prediction paths concurrently. Augmenting branch predictor with an additional execution history field for each functional unit may require significant additional hardware.

5.3. Optimizations

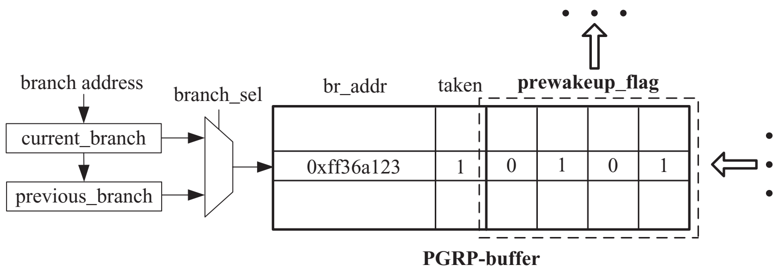
Two issues arise when using the given base predictors to perform wakeup. The first problem is determining which prior branch provides sufficient lead time for prewakeup when a wakeup overhead is first incurred. The second problem is how to handle a low accuracy pre-wakeup (i.e., predicted incoming instructions are not actually executed).

5.3.1. Lead Time Control. Pre-wakeup prediction should provide sufficient lead time. Wakeup latencies depend on the circuit parameter, wakeup overheads may be incurred by functional units that have already been pre-waked up since wakeup is too late to make functional units available in time.

The example of wakeup overhead in Figure 7 depicts a pre-wakeup condition. Initially, a multiplier instruction is executed several cycles after *branchb*. If the multiplier is already power gated (Figure 7(a)), the pre-wakeup determined by *branchb* provides the multiplier for execution. Clearly, the multiplier instruction is stalled by T_{stall} cycles as the pre-wakeup is too late to catch up with the multiplier instruction. The nonzero wakeup overhead reminds the predictor that *branchb* could not provide enough wakeup time and enables a change of pre-wakeup by selecting an earlier branch. Hence, the pre-wakeup strategy further pre-wakes the multiplier in an earlier branch (*brancha*),



(a) Earlier pre-wakeup example. Pre-wakeup at branchb has stalled cycles T_{stall}, earlier pre-wakeup at brancha avoids performance overhead.



(b) Integrating further pre-wakeup with PGRP.

Fig. 7. Earlier pre-wakeup scheme.

which incurs only negligible wakeup overhead (Figure 7(a)); thus, the multiplier instruction (e.g., *mul* $1, $2, $6) is not stalled.

Accordingly, each wakeup overhead is checked to determine whether a new entry must be indexed by an earlier branch and inserted into the PGRP-buffer. The predictor tracks the last several branches and corresponding pre-wakeup information in a shift register by updating it whenever a branch is executed. Figure 7(b) shows the PGRP architecture extended with the early wakeup mechanism. The current_branch and previous_branch registers are implemented as a sequential table, which holds multiple branch addresses from the branch address register. These branch addresses are selected based on the pre-wakeup information captured in the PGRP-buffer. In some functional units, when wakeup overhead is incurred, the predictor first checks for pre-wakeup activity in corresponding functional units. Initially, the prior branch is selected as the pre-wakeup branch; otherwise, the branch that is earlier than the prior pre-wakeup branch is used.

Simulation of the first 2000M instructions from the SPEC2000 suit revealed that, when T_{wakeup} is 20 cycles, <1% performance degradation is achieved with four levels of earlier branch addresses. Of course, the predictor can further advance pre-wakeup by adding a relatively earlier branch when overhead is still incurred.

5.3.2. Pre-Wakeup Check Function. Recurrent wakeups of functional units that have been allowed for power gating are usually anticipated by the instructions in a loop many times so that recurrent wakeup patterns appear predictable; however, a number of branches may behave in a bipolar manner and the simulations showed that the predictor often mispredicts pre-wakeup events in some benchmarks, which significantly degrade prediction accuracy and reduce power gating time.

In this study, an unnecessary wakeup caused by incorrect prediction can be prevented by inserting a pre-wakeup check register to find out whether predicted instructions are

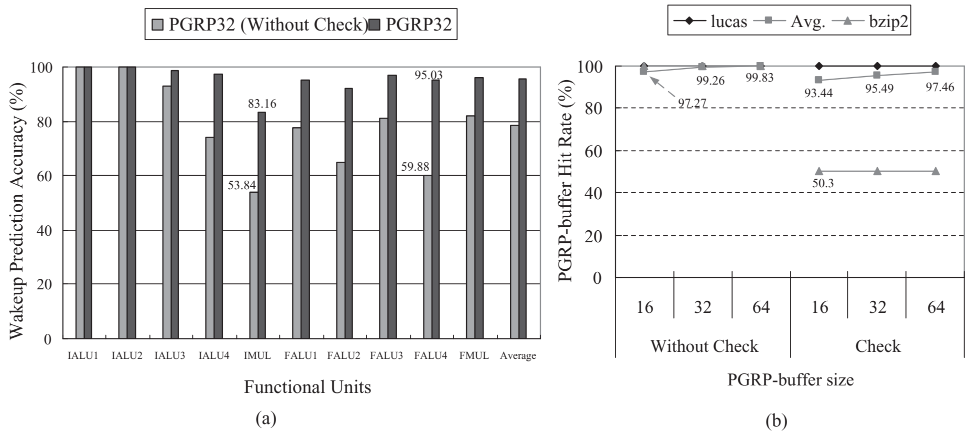


Fig. 8. Profiling of pre-wakeup accuracy and hit rate with and without pre-wakeup check function (assuming a PGRP-buffer with thirty-two entries). (a) Average accuracy of all SPEC2000 applications for each functional unit. (b) Hit rates of PGRP-buffer with different sizes. *lucas* and *bzip*2 show the highest and lowest hit rate, respectively.

actually executed upon (behind) each pre-wakup and to prevent most prediction errors by correcting (removing) low accuracy data. Typically, recurrent wakeups within a short time interval can result in serious performance impact. Moreover, since the penalty for incorrect pre-wakeup is relatively small (it cannot continue leakage reduction for $T_{idledetect}$ cycles); on the rare occasions when anticipated instructions are not actually executed behind the pre-wakeups (incorrect prediction), pre-wakeup is still worthwhile. Instead, frequent mispredictions (e.g., two consecutive misses used by this study) indicate that the functional unit is usually not required behind the pre-wakeup and has long idle time, so pre-wakeup check function corrects (removes) the relative wakeup information.

Prediction accuracy is defined as the fraction of predicted pre-wakeup in which instructions for the functional unit have actually been executed (as shown in Figure 8(a)). Investigations on the check ability in order to improve prediction accuracy are also shown. The average accuracy of pre-wakeup prediction by the original PGRP is only 78.65%. The pre-wakeup check function removes incorrect pre-wakeup predictions, so accuracy improves to 95.5%. The first two functional units, IALU1 and IALU2, achieve near 100% accuracy because they constantly execute instructions. Most functional units have accuracy as high as 92% with the exception of IMUL, for which accuracy was only 83.16%.

Hit rate. Figure 8(b) shows the hit rates, which are calculated as the ratio of the hit number in PGRP-buffer versus fraction of branch instructions that should be associated with pre-wakeup prediction (i.e., pre-wakeup branch), for varying PGRP-buffer size from 16 to 64. Pre-wakeup behavior is retained by PGRP with only a very small table size. Hit rate can be improved by increasing PGRP-buffer size to provide additional pre-wakeups and maintain performance. Notably, the size of the PGRP-buffer should be carefully chosen to keep sufficient pre-wakeup information and to maintain a high hit rate. However, the PGRP-buffer must also be small enough to satisfy the access latency, area, and hardware cost constraints. A PGRP-buffer with thirty-two entries obtains an average hit rate of 99.26%. In contrast, pre-wakeup check function removes incorrect pre-wakeup information, which decreases hit rate to 95.49%.

The high hit rate means that most pre-wakeup branches are found in the PGRP-buffer; thus, PGRP can provide early wakeups sufficiently without incurring recurrent wakeup overheads. Note that low hit rate does not certainly indicate that pre-wakeup is unsuccessful (e.g., 50.3% for *bzip*2) since wakeup events are too rare to require sufficient pre-wakeups (e.g., Table II shows that there are only 0.03% of branches in *bzip*2 needed for pre-wakeup prediction). Thus, evaluation of pre-wakeup efficiency should also take pre-wakeup branch ratio into account. The branch instructions are split into two groups: the pre-wakeup branch instructions, which are predicted for pre-wakeup of functional units, and the other dynamically predicted functional units that do not require wakeup. The size of the pre-wakeup branch depends on the specific application, so smaller or larger fractions of branches can be used for pre-wakeup prediction.

To explore the underlying principles of the proposed wakeup prediction method, the following scenario from benchmark *gcc* demonstrates the capture of a wakeup event, which is the underlying pre-wakeup mechanism in PGRP.

5.4. Recurrent Wakeup Activities

Repeatedly executing power-gated functional units cause recurrent wakeup activities, which result in predictable activities, thereby predicting the execution demands of functional units in the near future.

To clarify how recurrent wakeup overhead patterns are prevented by the proposed pre-wakeup scheme, the wakeup activities of integer multiplier (IMUL) for benchmark *gcc* with 12000 instructions over 18000 cycles were profiled (Figure 9(a)), and the processor executed the nested loop behind 3390 cycles (Figure 9(b)). Figure 9(a) illustrates power gating and pre-wakeup activities using a vertical bar in the `Power_gating` and `Pre-wakeup` subgraph. The symbols "bra" and "brb" under the vertical bars in the `Pre-wakeup` graph correspond to the branch instructions in Figure 9(b) leading to the IMUL instruction. The stall cycles at IMUL for time-based approach and the pre-wakeup prediction are presented for the same time frame. The address distribution of executed branch is also shown.

Effective pre-wakeups in functional units for sustained periods incurs little or no performance degradation. Accurate prediction is essential and depends on the application program characteristics. Recurrent wakeup overheads incurred by time-based power gating as shown in Figure 9(a) clearly illustrates this point (see top of figure). Initially, power gating is efficiently enabled with no stalling cycles. However, after 3800 cycles, executing IMUL every 300–500 cycles causes 20-cycle stall episodes, which cannot be prevented by time-based approach with $T_{idledetect}$ of 150 cycles.

The top subgraph shows the address trace of branch instructions. The wakeup overheads were profiled as explained above, and the relevant address of branch instructions were plotted across the two distinct execution phases. The starting address was 4833.536M. As the distribution shows, the subgraph shows two bands with approximate address of 4832M and 4833M and with strong temporal locality in their address patterns. Analysis of the overall address patterns reveals two distinct phases, A and B, and recurrent wakeup overheads incurred at phase B. Lastly, Figure 9(a) (bottom of figure) shows that recurrent wakeup overheads are successfully prevented by the pre-wakeups at branches (brb and bra) in front of IMUL instructions after 5630 cycles.

Determine Pre-wakeup time. Instruction execution behind long idle periods is uniquely identifiable by tracking control flow instructions in order of occurrence. As discussed in Section 5.3.1, pre-wakeup time is easily determined by analyzing several leading overheads. As Figure 9(a) shows, wakeup overhead incurred at 3777 cycles causes 20 stalled cycles and provides notification of pre-wakeup. Hence, in Figure 9(b), information of branch *brb* in front of the IMUL instruction is stored in the PGRP-buffer. When branch *brb* is executed again, pre-wakeup of IMUL is determined at 5119 cycles. Nevertheless, eight stall cycles remain because pre-wakeup at *brb* is too late to prevent

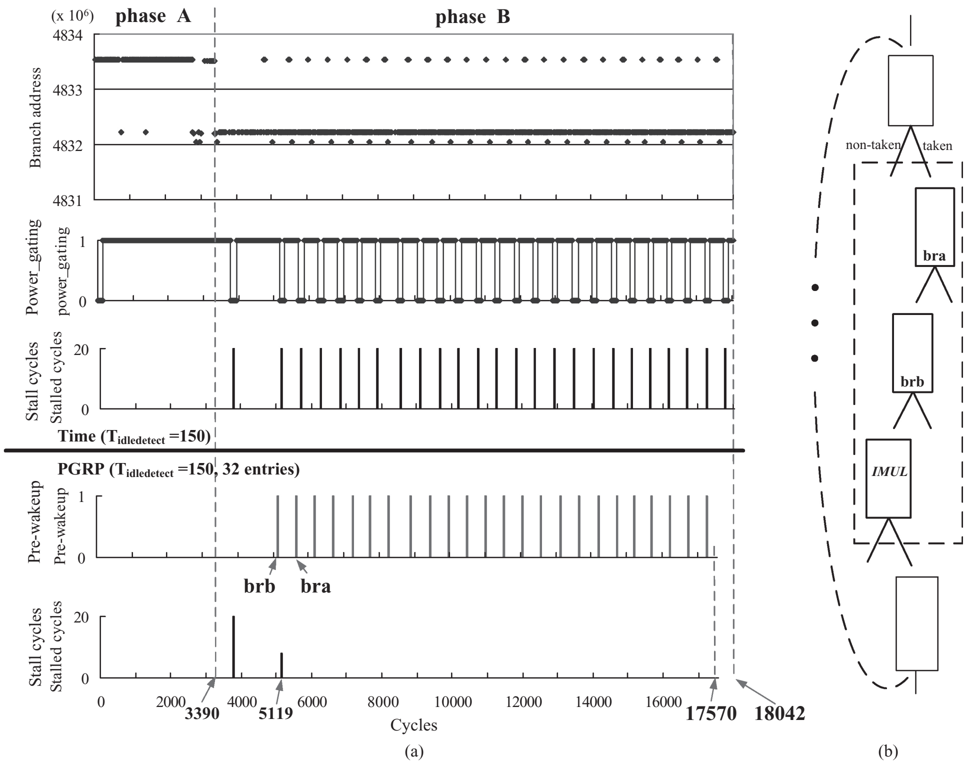


Fig. 9. Recurrent wakeup overheads in IMUL (Integer Multiplier) are associated with neighboring branch instructions. (a) Branch addresses, wakeup overheads, and pre-wakeups for IMUL as a function of time (in terms of thousands of clock cycles). The vertical bars in the Pre-wakeup subgraph show the pre-wakeup time in IMUL when branches *bra* and *brb* are executed. (b) A nested-loop of benchmark *gcc* contains an IMUL instruction.

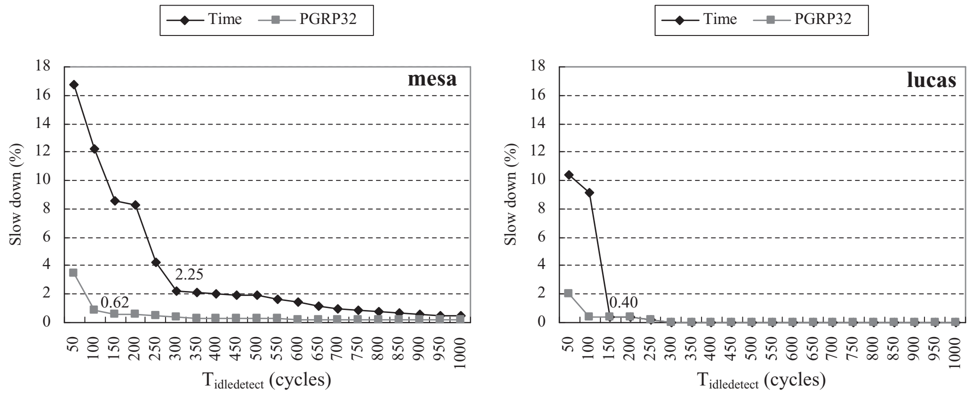
wakeup overhead successfully. Clearly, the earlier branch *bra* before *brb* is a better pre-wakeup candidate. Hence, at 5630 cycles, further pre-wakeup of IMUL provides a successful on-demand execution without wakeup overhead.

5.5. Performance/Energy Trade-Off

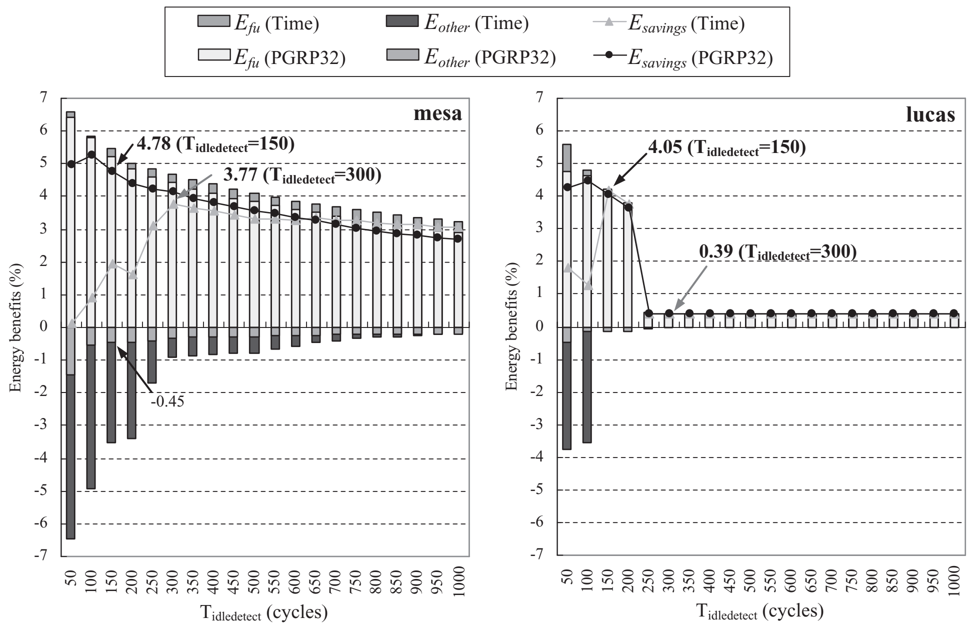
The same simulation is conducted with thirty-two entries in PGRP-buffer (Figure 10)(as in the experiment described in Section 4.2). Simulation results for time-based power gating (Figure 4) are shown for comparison purpose. Performance impact decreases linearly as $T_{idledetect}$ increases, indicating that applying the time-based approach in PGRP still prevents ineffective power gating during short idle periods. Energy overhead of *mesa* is negligible ($E_{other} = -0.45\%$) due to a slightly longer run-time (0.62%) when $T_{idledetect}$ is 150 cycles. Hence, overall energy savings in the processor is 4.78%. Moreover, *lucas* can also use the same low $T_{idledetect}$ and obtains 4.05% energy reduction. In contrast, time-based power gating can only use high $T_{idledetect}$ of 300 cycles due to the significantly longer run-time for *mesa* when $T_{idledetect}$ is shorter than 300 cycles; hence, the energy savings obtained by *lucas* is only 0.39%.

6. EXPERIMENTS

To compare time-based and PGRP policies to the theoretically best policy, this study applied *Ideal policy* [Hu et al. 2004] during simulations. This policy consumed the



(a) Slow down due to wakeup latency overhead.



(b) Energy benefits obtained by power gating.

Fig. 10. Evaluation of PGRP for the benchmarks *mesa* and *lucas*, T_{wakeup} and $T_{breakeven}$ are fixed at 20 cycles. Higher is better.

least energy but performed as well as the base model because the policy was assumed to contain knowledge of functional units required in the future. Notably, wakeup time T_{wakeup} is used inside the idle period, and the unit is ready for execution at the start of the busy period.

As the organization of PGRP-buffer is identical to that of a cache, the *CACTI* [Tarjan et al. 2006] tool with 0.18μm technology can estimate energy consumption by the PGRP-buffer.[3] Energy consumption was 0.093nJ, 0.16nJ, and 0.37nJ for 16, 32, and

[3]Only lower 10 bits of branch address is used to find the pre-wakeup, instead of its full 64-bit address. Throughout all simulation scenarios, using fewer bits address does not affect the prediction accuracy. But CACTI still use 64 bits to estimate PGRP-buffer energy consumption as this is the minimum word size

64 entries, respectively. Although PGRP-buffer consumed additional energy, the access number was trivial since it is only read for each branch execution and is written when nontrivial wakeup overhead is incurred.

In addition to the augmented PGRP-buffer, hardware overhead included ten 8-bit idle threshold counters (i.e., $T_{idledetect}$ could reach maximum 256 cycles), one 21-bit pre-wakeup check register and one 10-bit register (i.e., the ESR). The Hspice simulation of a typical 8-bit counter, 10-bit register and 21-bit register consumed 0.0003nJ, 0.00051nJ, and 0.00152nJ, respectively. As discussed in Section 5, the ESR is accessed by PGRP-buffer only when a branch instruction is executed and is updated when any functional unit is used. The 8-bit counter only operates when the corresponding functional unit is not power gated. Throughout the simulations, total energy overhead comprised <0.3% on the processor.

6.1. Idle Interval Distribution and Power-Gating Schemes

Each program phase in an application program has different idle interval distributions in functional units. In the simulation, for example, the percentage of idle time longer than 150 cycles in IMUL[4] is plotted for the first 2000M instructions from the *mesa* benchmark (Figure 11(a)). Clearly, few IMUL instructions with idle periods shorter than 750 cycles are executed behind 410M cycles. Considering that energy consumption of functional units (Figure 11(b)), the original case does not use a power-gating scheme; $T_{idledetect}$ is 150 cycles in the case of Time150 as well as PGRP32. The PGRP-buffer in PGRP32 has thirty-two entries. An additional case, Time750, is simulated using high detection threshold to prevent power gating of short idle periods to minimize wakeup overhead, but the energy benefit is also reduced. The ideal case would exploit any beneficial idle opportunity for power gating so that the processor can obtain the best energy benefits without performance degradation. The energy traces of the three different simulations represent similar waveform fluctuations.

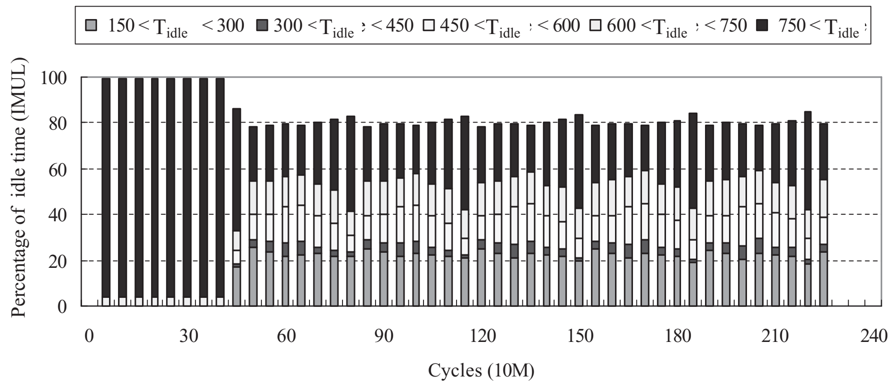
First, energy is reduced from 20.9mJ to 9.89mJ by Time150 and PGRP32, in which two policies have the same energy reduction as most idle periods are > 750 cycles, and no wakeup overhead is identified by PGRP32 for pre-wakeup operations. However, Time750 can only be reduced to 11.02mJ because the power-gating time is decreased due to high threshold of 750 cycles. Considering the time behind 410M cycles, power-gating time is decreased as pre-wakeup is used by PGRP32. Hence, PGRP32 has higher power dissipation than Time150 does; however, execution time for Time150 is increased, which accounts for 7.11% of the increase in execution time and which reduces overall processor energy by only 3.05%. Although energy benefits for functional units obtained by Time150 (38.96%) were greater than those achieved by PGRP32 (35.45%), PGRP32 achieved a 5.69% processor energy reduction with 0.54% performance overhead, which is closest to the 8.17% obtained by the Ideal policy. Conversely, Time750 skips the idle periods of <750 cycles to avoid wakeup overhead; thus, Time750 incurs a smaller increase in run-time (0.81%) than Time150 does. However, energy benefits by Time750 are decreased to only 28.14%. Time150 would assumedly have a high $T_{idledetect}$ to avoid performance overhead. However, as $T_{idledetect}$ (e.g., Time750) increases, the reduction in leakage energy diminishes.

6.2. Energy Savings in Functional Units

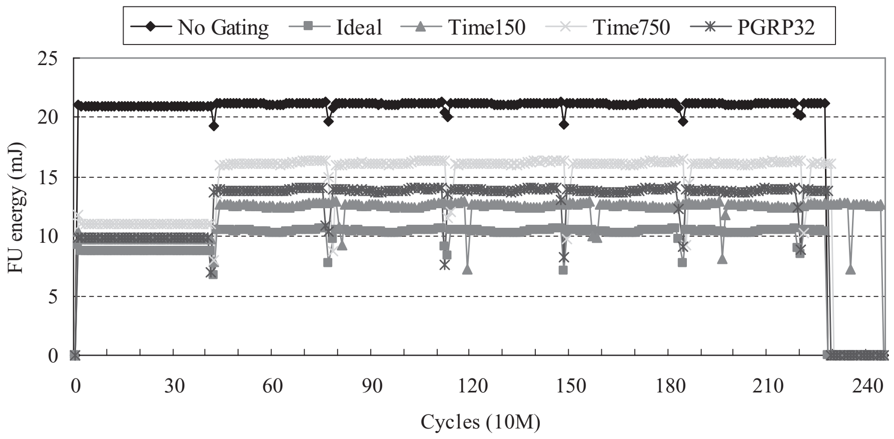
As mentioned in Section 4.1, most short idle periods in each benchmark are <150 cycles. Dynamic power gating maintains performance only when $T_{idledetect}$ is high enough to

provided by CACTI. Hence, the storage overhead is only 84 bytes, which is less than 0.26% of 2K-entry BTB size.

[4]FALU1 and FMUL have also similar percentage of idle periods greater than 150 cycles.



(a) Percentage ofidle time distribution in IMUL.



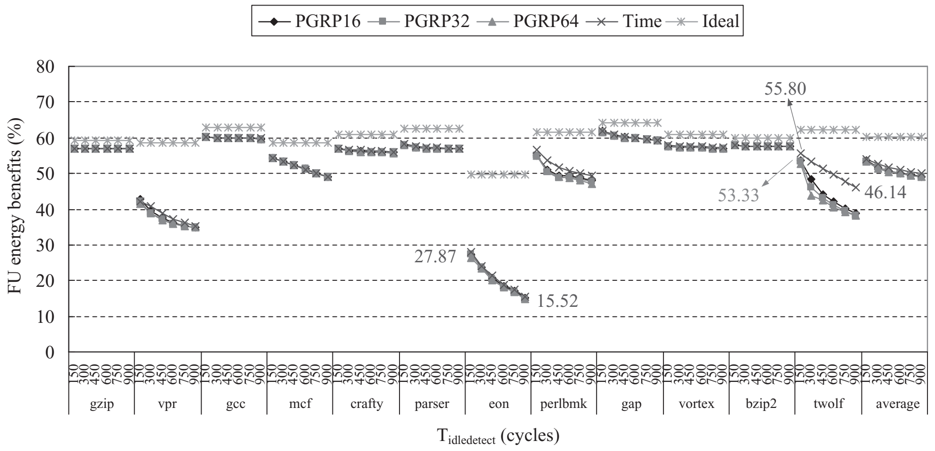
(b) Snapshot of energy profile in functional units. Each data point represents the energy consumption within the sample period of10M cycles.

Fig. 11. Simulation for the first 2000M instructions of mesa.

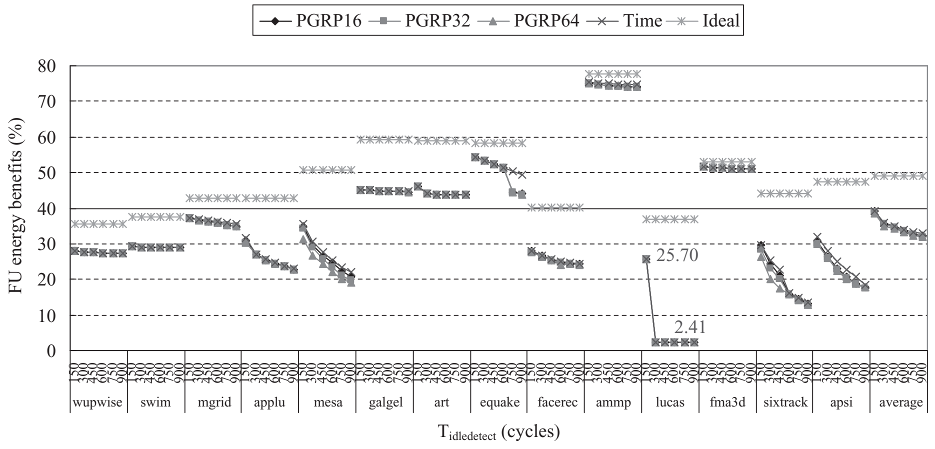
skip frequent transitions from sleep to active mode. Thus, this study only considers $T_{idledetect}$ with >150 cycles.

Figure 12 presents the percentage of energy reduction in functional units. Simulation results are reported as normalized metric where normalization is given with respect to the functional units of the baseline processor without power gating. Time-based power gating uses six different $T_{idledetect}$. The PGRP is simulated using different table sizes for PGRP-buffer, which demonstrates its pre-wakeup effect during power gating. For example, the pre-wakeup table for PGRP16 contains sixteen entries.

Most benchmarks achieve close to 60% energy reduction in functional units when $T_{idledetect}$ was 150 cycles. Percentage of energy reduced for INT applications exceeded that for FP applications as idle time in FALUs and FMULT for INT applications usually exceeded 900 cycles (Figure 3). Additionally, the graphs for Time and PGRP schemes almost overlap, which indicates that the aggressive pre-wakeup policy of PGRP pre-wakes the functional units in which power-gating time is reduced; however, pre-wakeup had little impact on leakage energy reduction.



(a) SPEC INT applications.



(b) SPEC FP applications.

Fig. 12. Percentage of energy reduction in functional units, Higher is better.

Average energy reduction achieved by the Ideal policy was 54.11%, and the difference between the Ideal policy and other policies was only 8.53% when $T_{idledetect}$ was 150 cycles. Hence, although time-based and PGRP schemes prevent power-gating within short idle intervals, significant reduction of leakage energy in functional units is still possible. Finally, as $T_{idledetect}$ increases, the energy savings by the Time and PGRP schemes decrease (when $T_{idledetect}$ is >300 cycles, the energy benefit of *lucas* is reduced from 25.7% to 2.41% because *lucas* has many middle idle periods as discussed in Section 4.1 and Figure 3(b).). Thus, in any power-gating control policy, $T_{idledetect}$ should be minimized.

6.3. Pre-Wakeup Prediction Accuracy

Figure 13 shows the average pre-wakeup prediction accuracy (the number of pre-wakeups with actual instruction executions/the number of pre-wakeups) in the functional units.

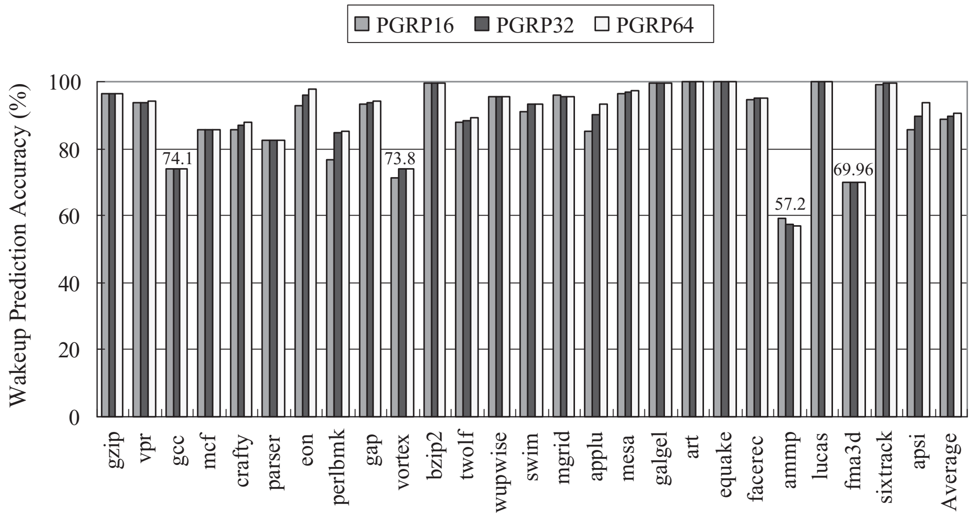


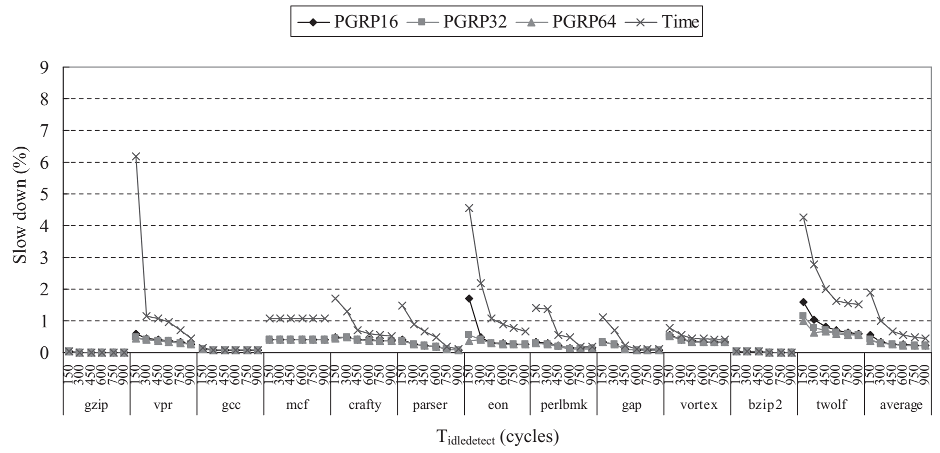
Fig. 13. Average prediction accuracy of functional units (except IALU1 and IALU2) for PGRP32.

Table II. Hit Rate, Pre-Wakeup Branch Ratio, and Branch Instruction Ratio

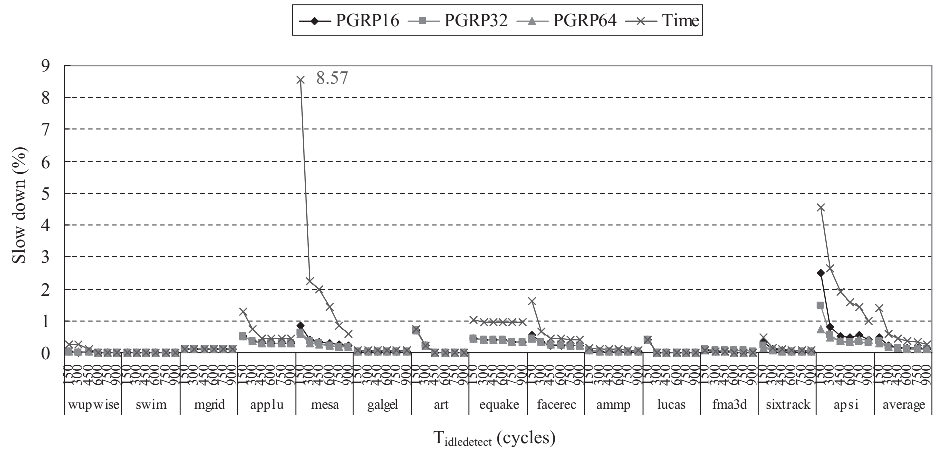
INT Suite	Hit Rate	Pre-wakeup Ratio	Branch Ratio	FP Suite	Hit Rate	Pre-wakeup Ratio	Branch Ratio
gzip	99.02	0.05	8.41	wupwise	99.99	1.34	3.99
vpr	98.65	21.12	8.33	swim	97.88	0.03	1.77
gcc	83.51	0.15	14.76	mgrid	99.88	21.37	0.95
mcf	99.99	1.99	20.07	applu	98.35	15.55	3.25
crafty	96.29	7.07	9.47	mesa	97.81	18.10	7.76
parser	97.47	7.43	14.52	galgel	99.96	12.63	6.32
eon	99.57	12.83	10.06	art	99.99	7.45	10.79
perlbmk	96.36	5.45	12.63	equake	99.99	1.76	19.78
gap	99.48	24.50	5.48	facerec	99.60	12.58	5.69
vortex	90.15	1.45	15.76	ammp	95.35	0.86	23.16
bzip2	50.30	0.03	9.58	lucas	99.99	11.11	3.72
twolf	98.97	24.85	12.09	fma3d	97.42	0.02	17.89
				sixtrack	99.78	33.46	7.79
				apsi	86.84	12.99	6.19
INT avg	92.48	8.91	11.76	FP avg	98.06	10.66	8.51

Table II shows the hit rate (number of branch instructions found in PGRP-buffer /number of branch instructions that should be considered for pre-wakeup prediction, i.e., pre-wakeup branch instructions), the pre-wakeup branch ratio (number of pre-wakeup branch instructions/ number of branch instructions), and branch instruction ratio (number of branch instructions/number of total instructions). Average hit rate is 95.49%, the pre-wakeup branch ratio is 9.85%, and the branch instruction ratio is 10.01%. Obviously, most benchmarks achieve high prediction accuracy and high hit rate concurrently when a significant number of pre-wakeups (pre-wakeup branch ratio) are required for avoiding wakeup overheads.

Recall that pre-wakeup is needed only when a branch address and its direction are found in PGRP-buffer. As the hit rate, pre-wakeup branch ratio, and the number of branch instructions decrease, the impact of prediction accuracy on power gating time also decreases due to the fewer pre-wakeup activities. For example, the accuracies of *ammp* and *fma3d* are as low as 57.22% and 69.96%, respectively, which indicate that most pre-wakeups do not have actual executions. The percentage of pre-wakeups for



(a) SPEC INT applications.



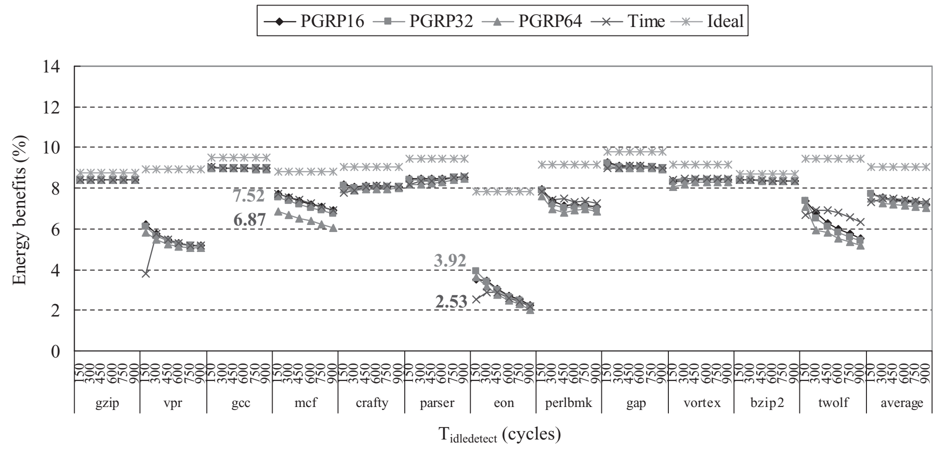
(b) SPEC FP applications.

Fig. 14. Performance degradation. Lower is better.

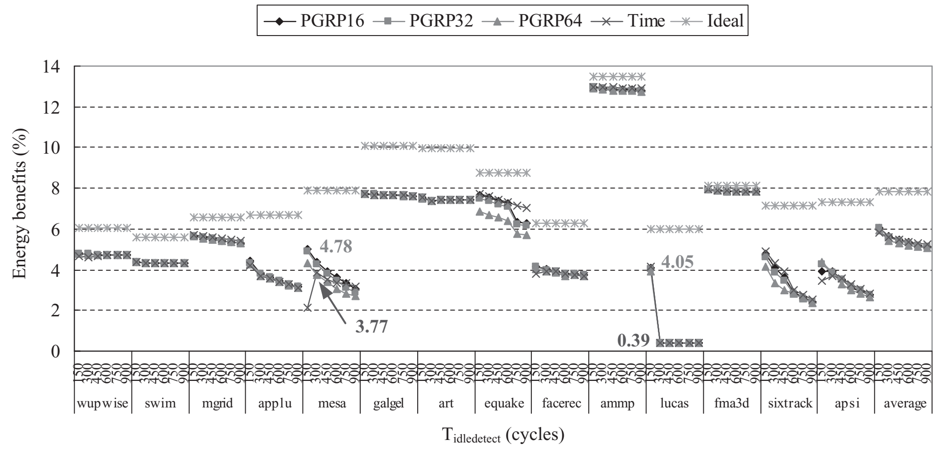
all instructions is only 0.19% and 0.003%, respectively (hit rate * pre-wakeup branch ratio * branch instruction ratio); hence, leakage energy reduction of PGRP32 is almost the same with that of Time scheme (Figure 12). Instead, with the higher prediction accuracy of 88.14% for *twolf*, PGRP32 obtains a slightly smaller benefit (53.33%) than Time (55.8%) does since the percentage of pre-wakeups reaches as high as 2.47%. Of course, performance impact of Time scheme ($T_{idledetect}$ is 150 cycles) is reduced by PGRP from 4.27% to 1.16% (Figure 14(a)).

6.4. Performance Impact

As Figure 14 shows, Time150 incurs a severe performance degradation since, as noted above, benchmarks *vpr*, *eon*, *mesa*, and *apsi* have idle periods > 200 cycles (Figure 3). Meanwhile, PGRP has a much smaller increase in run-time compared to the Time scheme. Naturally, performance overhead for the Time and PGRP schemes decreases as $T_{idledetect}$ increases, i.e., $T_{idledetect}$ must be high enough to avoid serious performance cost.



(a) SPEC INT applications.



(b) SPEC FP applications.

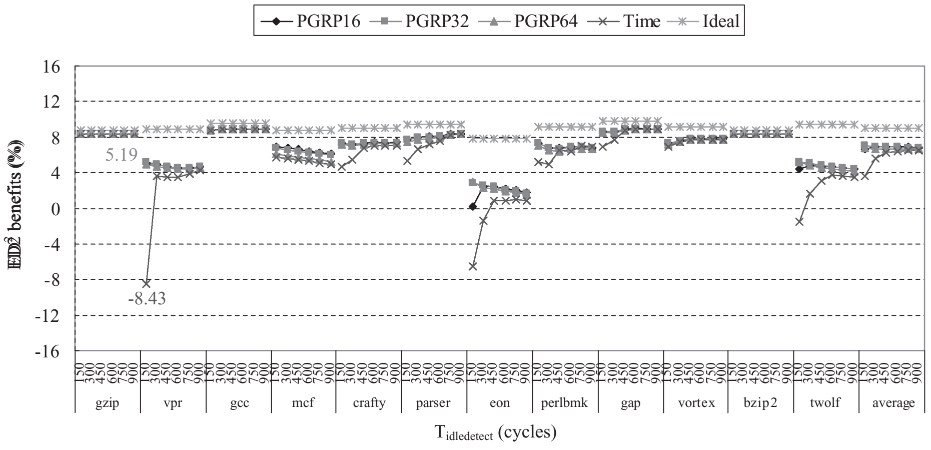
Fig. 15. Percentage of processor energy reduction. Higher is better.

For benchmarks *eon* and *apsi*, when $T_{idledetect}$ is 150 cycles, a significant performance improvement is achieved when PGRP-buffer size increases from 16 entries to 32 entries.

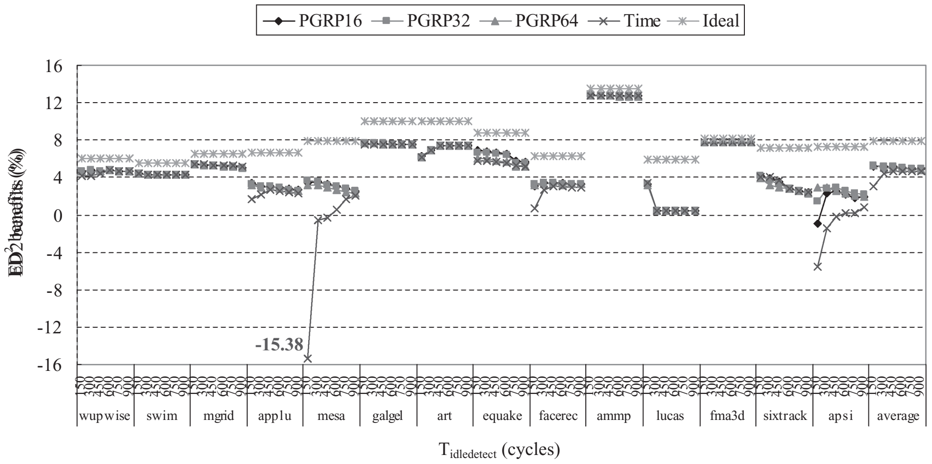
Consequently, increasing the table size in PGRP-buffer prevents additional wakeup overheads. When $T_{idledetect}$ is 150 cycles, the increases in average runtime are 1.62%, 0.51%, 0.38%, and 0.33% for Time, PGRP16, PGRP32, and PGRP64 schemes, respectively.

6.5. Total Processor Energy Benefits

The Time and PGRP policies obtained the same energy reduction in functional units (Figure 12). Hence, most benchmarks obtained comparable processor energy benefits (Figure 15). When $T_{idledetect}$ is 150 cycles, the Time policy for benchmarks *vpr*, *eon*, and *mesa* obtains a smaller energy benefit than PGRP does due to the impact of increased runtime on overall processor energy. Since Time policy does not allow low $T_{idledetect}$ to prevent wakeup overheads, a high $T_{idledetect}$ reduces energy benefits (e.g., Time policy for *lucas* decreases energy benefits from 4.06% to 0.39% when $T_{idledetect}$ is



(a) SPEC INT applications.



(b) SPEC FP applications.

Fig. 16. Percentage of processor ED^2 reduction. Higher is better.

>300 cycles), and PGRP32 achieves the greatest (on average) energy benefits when $T_{idledetect}$ is 150 cycles.

6.6. Energy∗Delay^2 (ED^2)

Figure 16 shows the Energy∗Delay^2(ED^2) for SPEC2000 applications. PGRP can achieve a stable ED^2 improvement for each benchmark because PGRP has negligible increase in run-time. Considering benchmarks *apsi*, *eon*, and *mesa*, the Time policy represents a negative ED^2 due to the significantly longer runtime. The PGRP obtains an ED^2 improvement comparable to that obtained by the *Ideal* policy.

In summary, the given evaluation of the Time and PGRP schemes indicates that the best results are obtained by PGRP32, which has a history table with thirty-two entries when $T_{idledetect}$ is 150 cycles. This power-gating policy obtains average energy savings of 6.82% (up to 12.94% for *ammp*) and an average ED^2 improvement of 6.11% (up to 12.84% for the *ammp*), which is better than that obtained by the Time scheme (6.50%

energy benefits, with an ED^2 improvement of 3.34% when $T_{idledetect}$ is 150 cycles.) (e.g., benchmarks *vpr*, *apsi*, *eon*, and *mesa* have negative ED^2). This configuration is a good choice for minimizing performance cost and maximizes energy savings. Notably, when $T_{idledetect}$ is 150 cycles, PGRP consistently achieves significant energy reduction, which confirms the effectiveness of the aggressive pre-wakeup scheme. Although the Time scheme achieves similar energy benefits for most benchmarks, its limitation is the need for a high $T_{idledetect}$ to minimize performance cost. Hence, compared with the Time scheme, PGRP not only obtains better benefits, it does so with lower performance overhead. Finally, for all benchmarks other than $twolf$ =1.16% and *apsi* =1.46%, increased run-time was maintained <1%, which confirms the effectiveness of the PGRP scheme.

6.7. Sensitivity Study of Leakage Power

Static power consumption is a growing concern in advanced technologies. According to the 2001 International Technology Roadmap for Semiconductors (ITRS), leakage power in 70nm technologies is likely to approach 50% of total power dissipation and to become the main consideration in circuit power over the next several processor generations. This study showed the extent of the leakage power problem, which should be addressed by future technologies.

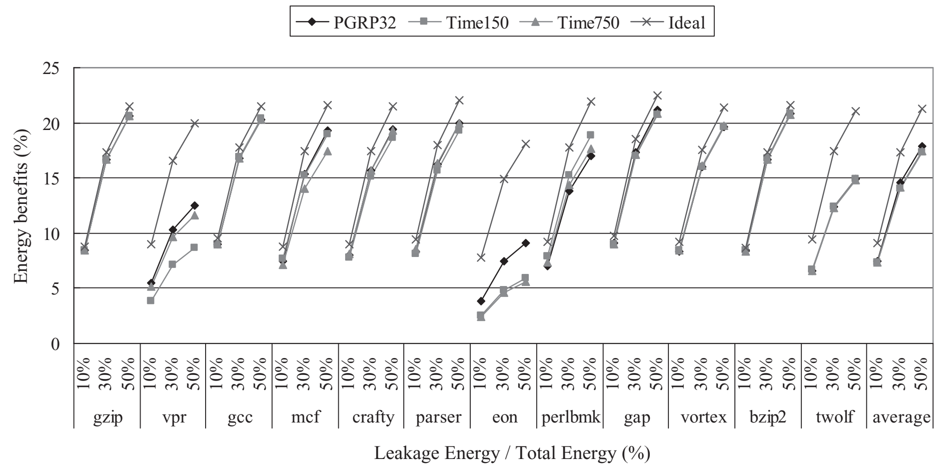
Figure 17(a) shows that, on average, PGRP32 reduces processor energy by 6.63%, 12.56%, and 15.33% when leakage power accounts for 10%, 30%, and 50% of total power, respectively. Time750 and PGRP32 show similar energy reduction when (leakage power)/(total power) is 10%. However, if leakage power increases as in the cases of *eon*, *applu*, *mesa*, *lucas*, *sixtrack*, and *apsi*, the differences become larger. The low energy benefits obtained by Time750 scheme result from the long threshold time (750 cycles). Although PGRP32 provides energy benefits similar to those obtained by TIME150, PGRP32 obtains higher benefits due to the significantly increased runtime of *vpr* (6.17%) and *mesa* (8.57%) incurred by Time150.

7. CONCLUSION

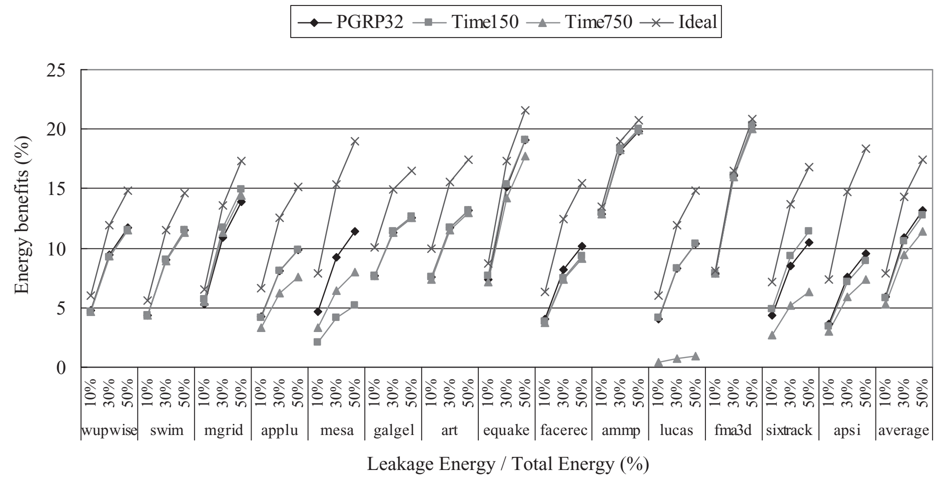
Subthreshold leakage power is a continuing problem in CMOS circuit design. Due to the increased number of functional units in SoC design platforms, an aggressive power-gating scheme is needed to reduce leakage energy. Fluctuating supply voltage arising from power mode transitions require that wakeup (sleep to active mode transition) time be set pessimistically to limit the maximum transient currents. Hence, efficient power gating usually requires further effort to maintain performance (e.g., minimizing wakeup time while limiting ground bounce [Abdollahi et al. 2007] and multiple sleep modes [Singh et al. 2007]).

Time-based power-gating cannot avoid recurrent wakeup overheads as they rely on information about the length of current temporary idle time to predict power-gating opportunities; no information is shared across different power-gating activities. Our pre-wakeup predictor works in concert with time-based approach to power gate functional units after their idle time has exceeded the threshold, and restart far in advance of corresponding instructions. The rationale for the predictor is that the past execution behavior of loop-closing branches is usually a good guide to predict future execution needs. Hence, with knowledge of the future execution pattern known, PGRP has a chance to provide successful pre-wakeups.

The simulations in this study revealed that, when PGRP-buffer had thirty-two entries and $T_{idledetect}$ was 150 cycles, average performance impairment was 0.38%, and 1.46% in the worst case. Leakage energy in functional units was efficiently reduced by 45.58%, which approaches the ideal policy (54.11%). Average processor energy



(a) SPEC INT applications.



(b) SPEC FP applications.

Fig. 17. Percentage of processor energy reduction when leakage power account for 10%, 30% and 50% of the total power. Higher is better.

savings were 6.83%. Thus, PGRP strives to predict when to perform early wakeup for functional units, and a low $T_{idledetect}$ consistently provides significant energy benefits without adversely impacting performance.

REFERENCES

Abdollahi, A., Fallah, F., and Pedram, M. 2007. A robust power gating structure and power mode transition strategy for mtcmos design. *IEEE Trans. VLSI Syst. 15,* 1, 80–89.

Agarwal, K., Deogun, H., Sylvester, D., and Nowka, K. 2006. Power gating with multiple sleep modes. In *Proceedings of the International Symposium on Quality Electronic Design*. IEEE Computer Society, Los Alamitos, CA, 633–637.

Baniasadi, A. 9 Sept. 2005. Power-aware branch predictor update. *IEE Proc. Compute. Digit. Tech. 152,* 5, 585–595.

BANIASADI, A. AND MOSHOVOS, A. 2002. Branch predictor prediction: A power-aware branch predictor for high-performance processors. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*. IEEE Computer Society, Los Alamitos, CA, 458.

BHUNIA, S., BANERJEE, N., CHEN, Q., MAHMOODI, H., AND ROY, K. 2005. A novel synthesis approach for active leakage power reduction using dynamic supply gating. In *Proceedings of the 42nd Annual Design Automation Conference (DAC'05)*. ACM, New York, NY, 479–484.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*. 83–94.

CALIMERA, A., BENINI, L., MACII, A., MACII, E., AND PONCINO, M. 2009. Design of a flexible reactivation cell for safe power-mode transition in power-gated circuits. *Trans. Cir. Sys. Part I 56,* 9, 1979–1993.

CHUNG, S. W. AND SKADRON, K. 2008. On-demand solution to minimize I-Cache leakage energy with maintaining performance. *IEEE Trans. Comput. 57,* 1, 7–24.

DROPSHO, S., KURSUN, V., ALBONESI, D. H., DWARKADAS, S., AND FRIEDMAN, E. G. 2002. Managing static leakage energy in microprocessor functional units. In *Proceedings of the International Symposium on Microarchitecture*. IEEE Computer Society Press, Los Alamitos, CA, 321–332.

GROCHOWSKI, E., AYERS, D., AND TIWARI, V. 2002. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA'02)*. IEEE Computer Society, Los Alamitos, CA, 7.

HU, Z., BUYUKTOSUNOGLU, A., SRINIVASAN, V., ZYUBAN, V., JACOBSON, H., AND BOSE, P. 2004. Microarchitectural techniques for power gating of execution units. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 32–37.

ITRS. International technology roadmap for semiconductor. http://public.itrs.net.

KAO, J., CHANDRAKASAN, A., AND ANTONIADIS, D. 1997. Transistor sizing issues and tool for multi-threshold cmos technology. In *Proceedings of the 34th Annual Design Automation Conference (DAC'97)*. ACM, New York, NY, 409–414.

KIM, S., KOSONOCKY, S. V., AND KNEBEL, D. R. 2003. Understanding and minimizing ground bounce during mode transition of power gating structures. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 22–25.

LEE, Y., JEONG, D.-K., AND KIM, T. 2008. Simultaneous control of power/ground current, wakeup time and transistor overhead in power gated circuits. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08)*. IEEE Press, Los Alamitos, CA, 169–172.

MOHAMOOD, F., HEALY, M. B., LIM, S. K., AND LEE, H.-H. S. 2006. A floorplan-aware dynamic inductive noise controller for reliable processor design. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'39)*. IEEE Computer Society, Los Alamitos, CA, 3–14.

PANT, M. D., PANT, P., WILLS, D. S., AND TIWARI, V. 1999. An architectural solution for the inductive noise problem due to clock-gating. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'99)*. ACM, New York, NY, 255–257.

SINGH, H., AGARWAL, K., SYLVESTER, D., AND NOWKA, K. J. Nov. 2007. Enhanced leakage reduction techniques using intermediate strength power gating. *IEEE Trans. VLSI Syst. 15,* 11, 1215–1224.

TARJAN, D., THOZIYOOR, S., AND JOUPPI, N. P. 2006. CACTI 4.0. Tech. rep. hpl-2006-86, HP Laboratories.

YOU, Y.-P., HUANG, C.-W., AND LEE, J. K. 2007. Compilation for compact power-gating controls. *ACM Trans. Des. Autom. Electron. Syst. 12,* 4, 51.

YOU, Y.-P., LEE, C., AND LEE, J. K. 2006. Compilers for leakage power reduction. *ACM Trans. Des. Autom. Electron. Syst. 11,* 1, 147–164.

Received March 2008; revised January 2010, February 2011; accepted July 2011