RC24948 (W1002-049) February 18, 2010
Computer Science

IBM Research Report

Dynamic Power Gating with Quality Guarantees

Anita Lungu
Duke University

Pradip Bose, Alper Buyuktosunoglu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Daniel J. Sorin
Duke University

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Dynamic Power Gating with Quality Guarantees

ANITA LUNGU

Department of Computer Science, Duke University

PRADIP BOSE

IBM T. J. Watson Research Center

ALPER BUYUKTOSUNOGLU

IBM T. J. Watson Research Center

DANIEL J. SORIN

Department of Electrical and Computer Engineering, Duke University

Power gating has emerged as a promising solution for reducing leakage energy consumption. However, power gating is usually driven by a predictive control and frequent mispredictions can counter-productively lead to a large increase in energy consumption. This energy vulnerability of the system could be exploited by malicious applications such as a power virus, or it may be exposed by regular applications containing repetitive mispredictions patterns. The possibility of these power overruns decreases the confidence in the overall design's robustness, potentially leading to the power gating feature not being implemented in real processors, despite its large energy saving benefits in the common case.

In this work we document this vulnerability for a system implementing power gating of processor functional units. We propose to counteract this vulnerability by including a guard mechanism to ensure that the overall power gating control hardware is "safe" in the sense that power-overruns are prevented. Because power gating, with or without our guard mechanism, might be viewed as adding too much burden to processor verification, we demonstrate that functional correctness can be ensured for a system implementing power gating of processor functional units, with low supplementary verification effort.

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

General Terms
Design, Performance, Experimentation

Keywords
power gating, power management, execution units, low power, microarchitecture

1. INTRODUCTION

One of the most promising mechanisms for reducing leakage energy is power gating, whereby leakage energy is saved by cutting the supply voltage to idle circuits. In prior

This TODAES submission is an extension of the following paper:

Anita Lungu, Pradip Bose, Alper Buyuktosunoglu and Daniel J. Sorin. "Dynamic Power Gating with Quality Guarantees." *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2009.

work, Hu et. al [3] demonstrated how to effectively use microarchitectural information to predict when to power gate a microprocessor's units, such as its functional units. Recently, Intel has announced the adoption of power-gating in Nehalem [6], although the focus is core-level gating, and the exact dynamic algorithms have not been published.

Power gating can potentially save a significant amount of leakage energy but, because it is predictive, *it can also lead to significant energy penalties when mispredictions are frequent*. Every time a component is power gated off, some energy is needed for turning the component off and then on again. If the power gated unit needs to be brought back in use soon after being turned off, that energy cost is not offset by the leakage energy saved during power gating.

We show in this paper that the impact of power gating depends on the workload running on the processor, and we observe that both big wins *and big losses* can occur. Across the SPEC2006 benchmarks, we observe a large variability in the leakage energy savings of a well-known power gating algorithm [3] applied to functional units. On a given benchmark, power gating can save as much as 99% of a functional unit's leakage energy during some time frames, whereas during other time frames it can consume 70% *more* energy than a system without power gating.

When adding a feature such as power gating to a processor, we would like for it to both benefit the common case and do no harm in less common cases. To assess the common case benefit (e.g., average energy savings), it is generally sufficient to simulate the system across a wide range of benchmarks. To assess the potential for harm (e.g., consuming 70% extra energy), this simulation approach is often insufficient, because *a harmful situation may be unacceptable even if it is unlikely.* Consider, for example, a power virus application designed to attack the system by exploiting this vulnerability and increasing its energy consumption. When deciding whether to add a feature to a real design, we must consider the feature's behavior for applications that have not been simulated. Such an unsimulated behavior could be, for example, a scientific application running in a loop that consistently wastes a significant amount of energy through the addition of the power gating scheme. This consistent waste of energy is a design vulnerability of the power gating scheme that might prevent its adoption in a real processor.

When a proposed feature such as power gating can have both a large benefit and a large penalty, we would like to augment it with two mechanisms. First, we want a mechanism that enables the feature when that is likely to translate into a win and disables it when the result is likely to be a loss. Second, we want a mechanism that provides *a guaranteed bound on the worst-case behavior* of the feature. We propose adding both of these mechanisms. Specifically:

- We augment current dynamic power gating schemes with a Success Monitor that dynamically estimates whether a particular power gating scheme is successful at saving energy or not.
- Based on the Success Monitor, we add a feedback control mechanism, called the Success Monitor Switch, that enables/disables power gating depending on its success. For benchmarks initially exhibiting energy loss, the Success Monitor Switch saves energy and decreases performance penalties.
- We implement a Token Counting Guard mechanism that provably bounds the worst-case energy penalty to an acceptable threshold (2% of leakage energy) for a specified time interval.

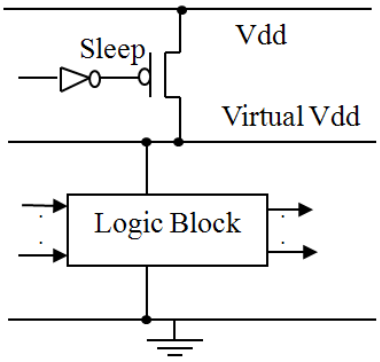

Fig. 1. Power gated circuit

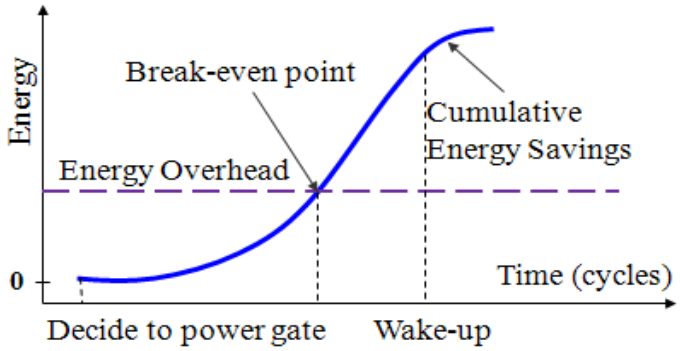

Fig. 2. Break-even point

- We combine the Success Monitor Switch with the Token Counting Guard to bound the energy loss while slightly increasing the average energy savings (by 2%) and decreasing the performance penalty by 63%.
- We demonstrate that functional correctness at microarchitectural level can be ensured for a system implementing power gating of processor functional units, with or without our mechanisms, with low extra verification effort.

The rest of the paper is organized as follows. In Section 2, we discuss background on power gating, as well as its potential and pitfalls. Section 3 details our proposed mechanisms (the Success Monitor, the Success Monitor Switch and the Token Counting Guard). Section 4 and Section 5 present our evaluation methodology and results. We continue in Section 6 with the implications of these mechanisms on the processor's functional correctness and verification. Section 7 presents related work in power gating, and in Section 8 we conclude.

2. POWER GATING: POTENTIAL AND PITFALLS

Power gating requires, for each circuit that can be turned off, the presence of a header (or footer) "sleep" transistor that can set the supply voltage of the circuit to ground level (or V_{DD} level for footer) during idle times. Figure 1 illustrates power gating using a header transistor. Power gating also requires control logic to predict when would be a good time to power gate the circuit.

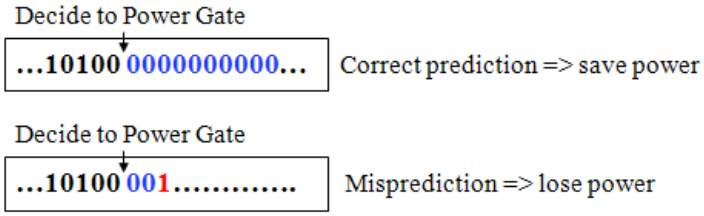

Fig. 3. Deciding when to power gate

Every time the control logic decides to power gate the circuit, an Energy Overhead cost is incurred. This energy overhead is due to 1) distributing the sleep signal to the header transistor before the circuit is actually turned off and 2) turning off the sleep signal and driving the Virtual V_{DD} when the circuit is powered-on again. The energy savings resulting from power gating is a non-linear function of time. Figure 2 depicts a schematic of the cumulative energy savings achieved during power gating. On the left side of the figure, the control algorithm decides to power gate the circuit and on the right the unit is turned on again. For more details on how such a curve can be derived, the interested reader is referred to Hu et al. [3]. The break-even point on the figure represents the point in time where the cumulative leakage energy savings equals to the energy overhead incurred by power gating. If, after the decision to power gate a unit, the unit stays idle for a time interval that is longer than the break-even point, then power gating saves energy. If, however, the unit needs to be active again before the break-even point is reached, then power gating incurs an energy penalty. Adopting the terminology used in by Hu et al. [3], we call the time between power gating and when the unit has reached the break-even point "uncompensated," and we call the time after the break-even point "compensated."

Figure 3 shows two example functional unit utilization patterns for a break-even point equal to nine. A "zero" in the pattern signifies that the unit has been idle for that cycle while a "one" means that the unit was utilized. The arrow indicates the point at which the decision is made to power gate the circuit. For the upper utilization pattern, the prediction to power gate the unit ends up saving energy because the unit remains idle for longer than the break-even point of 9. However, the lower pattern represents a misprediction for the control algorithm because the unit needs to become active again the third cycle after being turned off, resulting in an overall energy penalty. Power gating can also have an impact on performance due to supplementary cycles spent waiting for functional units to wake up. This impact on performance can be mitigated to a large extent by using information from the decode stage or issue queue to proactively wake up needed units.

2.1 An Example Power Gating Scheme

The baseline power gating scheme that we will augment in this paper is a microarchitectural technique for power gating functional units that was developed by Hu et al. [3]. They evaluate the potential for power gating the functional units and propose an IdleCount algorithm for deciding when to assert the power gate signal. The IdleCount algorithm counts the number of cycles a unit has been idle and decides to shut the unit off when a fixed threshold (called the *idle_detect*) has been reached. We will illustrate two of our proposed mechanisms, the Success Monitor Switch and Token Counting Guard, by adding them on top of the IdleCount algorithm. However, as we discuss in Section 3, our mechanisms are not restricted to the IdleCount algorithm.

2.2 Power Gating Potential

Workloads must have a significant amount of idleness for power gating to be effective. We show that this is indeed the case by exploring the power gating potential present for func-

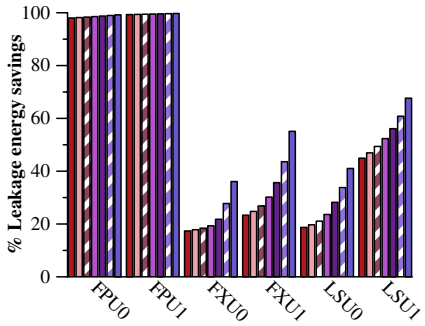

Fig. 4. Best-case potential to power gate (SPECint)

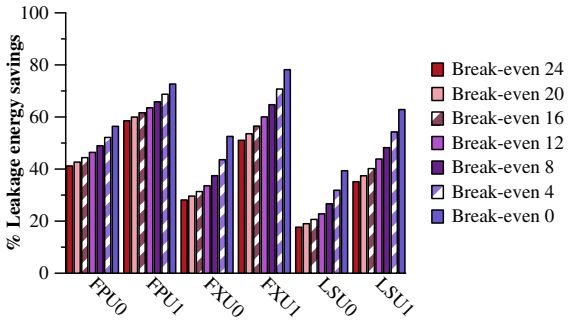

Fig. 5. Best-case potential to power gate (SPECfp)

tional units in the SPEC 2006 benchmarks. The data presented was obtained using the methodology presented in detail in Section 4.

Figure 4 (SPECint) and Figure 5 (SPECfp) show the energy saving potential through power gating of each unit, as a percentage of the total leakage energy of that unit, as a function of the break-even point. Considering the analytical model developed by Hu et al. [3], the break-even point for functional units developed in current technology parameters is between 9 and 24. The energy savings values are calculated for an oracle algorithm that knows the future workload behavior and turns a unit off immediately when that decision saves energy. The oracle also wakes-up a power gated unit when required without incurring any performance penalty at wake-up.

Overall, we observe a large potential for power gating across all units. Furthermore, there is a significant increase in power gating potential associated with a decrease in the break-even point. This result is intuitive since a lower break-even point signifies that more idle intervals can be effectively used for power gating by the oracle algorithm. If circuit-level techniques become available to reduce the energy overhead associated with power gating, then the additional energy savings are significant. The large potential for saving leakage energy by power gating processor units justifies microarchitectural level techniques for harvesting it. However, power gating can have its own pitfalls which we present next.

2.3 Power Gating Pitfalls

Due to the inherently speculative characteristic of power gating algorithms, it is possible for power gating to result in significant energy penalties. This possibility represents a vulnerability of the power gating algorithm that could result in a design decision against implementing the scheme in a real processor design.

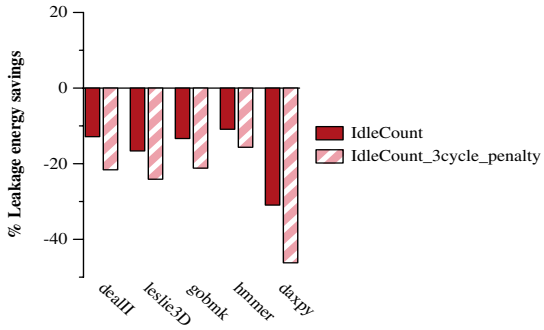

Fig. 6. Benchmarks wasting energy with IdleCount

To investigate this possibility, we implemented the IdleCount algorithm described in Section 2.1. Using the methodology described in Section 4, we ran the IdleCount algorithm on the SPEC2006 benchmarks and computed the energy savings. Several of the SPEC2006 benchmarks (*dealII, leslie3d, gobmk,* and *hmmer*) exhibited an energy penalty due to the addition of the IdleCount power gating feature. We also tested a well-known floating point microbenchmark (loop kernel) called *daxpy.* Dynamically, such a periodic loop execution profile can exhibit a type of behavior that is difficult for the power gating scheme to predict correctly. For these benchmarks, Figure 6 presents the percentage of leakage energy saved (with respect to a system with no power gating) by using IdleCount to power gate the FXU0 unit for a value of the idle_detect threshold of 5. Solid bars represent energy savings assuming zero performance penalty at wake-up from power gating while the striped bars assume a three cycle wake-up penalty. We see that the speculative power gating algorithm can indeed incur significant energy penalties (negative savings).

We conclude that a feature added to save power can at runtime end up costing extra energy. Moreover this type of runtime behavior can be encountered in regular benchmarks. To reduce the worst-case impact of power gating (and predictive schemes, in general), we add two mechanisms to the policy, which we discuss next.

3. MECHANISMS TO IMPROVE COMMON CASE AND BOUND WORST CASE

In this section, we describe two mechanisms that we propose adding to predictive schemes. The first is a Success Monitor that assesses the dynamic benefit of the predictive scheme. We can use the Success Monitor to make better predictions about when to power gate. By not using power gating when it is not saving energy, we decrease the performance penalty associated with power gating. Our second mechanism is a Token Counting Guard that provides a provable worst-case bound on the possible penalty associated with mispredictions. For both mechanisms, we discuss them in general but describe details for their specific application to power gating.

3.1 Improving Common Case: Success Monitor

We propose adding a Success Monitor to predictive mechanisms (e.g., power gating) whose success or loss depends on the dynamic behavior of the application. The goal of the Success Monitor is to estimate whether a particular policy is successful or harmful during a certain time interval that we call a *monitoring interval*. Based on the estimate from the Success Monitor, the control logic can better predict when to power gate (discussed in Section 3.1.2).

3.1.1 Success Monitor Structure and Behavior

The Success Monitor requires four values to dynamically estimate the success or loss of a policy, the first two of which are obtained at runtime (and called *efficiency counters*) and the latter two of which can be approximated to a constant for a given system:

- Number of successful instances per monitoring interval,
- Number of harmful instances per monitoring interval,
- Reward of a successful instance, and
- Cost of a harmful instance.

In the context of power gating, a successful instance is any compensated cycle (i.e., a cycle when a power gated unit remains idle after reaching the break-even point). We keep track of energy savings or penalties by using tokens, one token corresponding to the leakage energy used by the unit during one cycle. The reward of a successful instance is thus one token. A harmful instance is represented by any case when the unit needs to be woken up before reaching the break-even point. We pessimistically assign a cost equal to the Energy Overhead for that unit for any harmful state. The unit might, in fact, have been idle for a significant number of cycles before being woken up, so using the pessimistic estimate might disable power gating even when the energy savings benefit is at worst only marginally negative. However, it allows us to guarantee, by using the Token Counting Guard described in Section 3.2, that the energy penalty is below the bound set by the user.

3.1.2 Using Success Monitor to Improve Prediction

The information from the Success Monitor can be used by a hardware mechanism or by a high level software entity to dynamically change the power gating policy. In this work, we use the Success Monitor to drive an enable/disable signal for the power gating control logic. When the monitor estimates that the power gating policy has been harmful over the previous monitoring interval, we disable the policy during the next monitoring interval. Otherwise, the policy remains enabled. We call this solution the Success Monitor Switch. The efficiency counters are incremented and made available to the monitor regardless of whether the power gating policy is enabled. This permits the Success Monitor Switch to re-enable power gating when the monitor expects it to be beneficial.

Figure 7 shows the monitoring mechanism and this particular scenario.The lower part of the figure depicts the baseline IdleCount algorithm in which the system can be in one of three states: on (or active), power gated off but uncompensated (Off_U in figure), and power gated off and compensated.

The success efficiency counter is incremented each time the unit remains in a power gated compensated state (Win++ in the figure). The harmful efficiency counter is incremented each time the unit goes from a power gated uncompensated state to being active again (Lose++).

3.2 Bounding Worst Case: Token Counting Guard

We propose adding a Token Counting Guard mechanism that provides a guarantee on the worst-case behavior of a policy. The guarantee is given over a time interval, called the *guarantee interval,* which is an integer multiple of the monitoring interval (Figure 8a). We associate tokens with the quantities we wish to limit for the power gating scheme. One token equals the leakage power of the unit over one cycle. A token bag holds the tokens that a unit can consume over the course of one guarantee interval, as illustrated in Figure 8b. Figure 8c shows how the number of tokens in the token bag varies over time. The token bag is updated as follows. At the beginning of a guarantee interval, the token

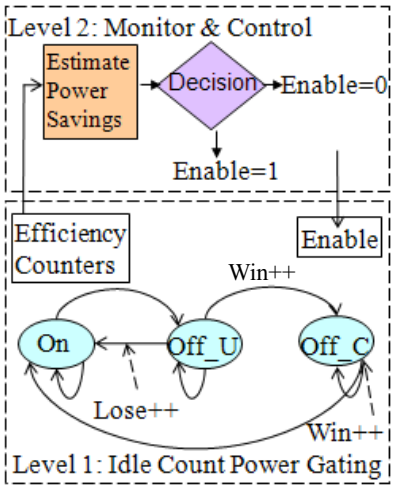

Fig. 7. Efficiency counters

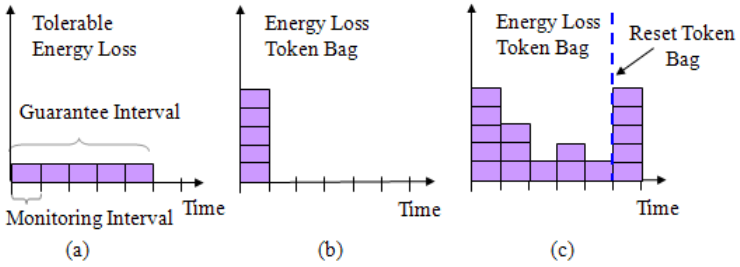

Fig. 8. Token bag concept

bag is reset to a fixed, non-zero value that represents the entire amount of energy penalty that can be tolerated over the current guarantee interval. For example if we wish to guarantee a maximum leakage energy penalty of 2% over 100 monitoring intervals each 50 cycles long, then the token bag is initiated to 100 tokens.[1] The choice of interval lengths depends on the technology (e.g., thermal time constants, tolerable power overshoots, etc.) and the guarantees we wish to provide.

At the end of each monitoring interval, the token bag is updated depending on the energy savings or penalty estimated by the Success Monitor over this interval. The token bag will be increased if energy was saved or it will be decreased if energy was wasted. The quantity by which the token bag is updated corresponds to the token equivalent of the energy saved or wasted.

At the beginning of each monitoring interval, a decision is made, based on the number of tokens in the bag, whether to enable power gating for the next monitoring interval. If there are enough tokens to tolerate the worst possible behavior for the next monitoring interval, then the power gating is enabled. Otherwise it is disabled. Once power gating is disabled, it remains disabled until the end of the guarantee interval when tokens become available again. The benefit of the token bound mechanism is that it limits the penalty incurred by power gating in the worst-case scenario. However, we wish to achieve this bound without disabling power gating when it could save energy. *The key to achieving this goal is that*

1. 100 intervals*50 cycles/interval*2tokens/100 cycles = 100 tokens

there is a significant amount of energy savings slack across one guarantee interval for most workloads. The power gating scheme is only disabled when all tokens have been consumed for that guarantee interval. By disabling power gating only in instances when it probably wastes energy, we see slightly greater energy savings for a system with Token Counting Guard compared to a system without it.

The application of the Success Monitor and the Token Counting Guard is not restricted to power gating schemes nor to power management in general. Any feature that, depending on runtime behavior, can succeed or not can benefit from these mechanisms.

3.3 Summary of Added Hardware

Success Monitor Switch. One counter, which we call the *power gating counter*, is required to count the number of cycles a unit is idle after being power gated, up to the break-even point. The counter is updated by an incrementer. Its size is $\lceil \log(\text{break\_even point}) \rceil$, 5 bits for a break-even point of 19. Two additional counters are the efficiency counters. The size of the *success efficiency counter* is $\lceil \log(\text{monitoring interval}) \rceil$, which is 6 bits for a 50-cycle monitoring interval. Its value is incremented each time the power gating counter is at the break-even point value and the unit remains power gated. The size of the *harmful efficiency counter* is $\lceil \log(\text{monitoring interval} / (\text{idle\_detect} + 1)) \rceil$, 4 bits when idle_detect is 5 (7-bit due to 3-bit left shift necessary explained below). Its value is incremented whenever the power gating counter is less than the break-even point and the unit needs to wake-up from power gating. Both efficiency counters are reset at the beginning of each monitoring interval and updated by incrementers. The total cost of all harmful instances is computed by shifting the harmful efficiency counter (a 3-bit left shift when cost of one harmful instance is 8 units). To calculate whether the scheme is successful, a 6-bit subtractor subtracts the success efficiency counter from the shifted value of the harmful efficiency counter.

Token Counting Guard. A 12-bit register is necessary to hold the value of the token bag.[2] This register is initiated to a fixed value at the beginning of each guarantee interval (100 in our experiments for a 2% bound over 100 monitoring intervals). A 12-bit adder is necessary for updating the value of the token bag at the end of each monitoring interval, and a 6-bit register holds the value to be added/subtracted.

Total Hardware Cost. Individual efficiency counters and token bag registers are required for each power gateable unit. However, the rest of the hardware necessary for the Success Monitor Switch and Token Counting Guard can be shared across closely located power gateable units, because the monitoring of the units can be done at different cycles for different units. We did a careful analysis of the added hardware overhead, by calculating the "latch count equivalent" of all the components within the Success Monitor Switch and the Token Counting Guard. Based on known VHDL-derived latch counts and macro-level area estimates of the processor core, we were able to (very conservatively) bound the power overhead of the added hardware to at most 0.1% of a PowerPC-like core described in Section 4.1.

---

2. The maximum value for the token bag is 3400 (<50*100). It can be calculated from the limit case when the scheme has maximum energy savings until interval i and maximum energy penalty after i: $100+50*i = 100(100-i) \Rightarrow i = 66$ and the maximum token bag value is 3400.

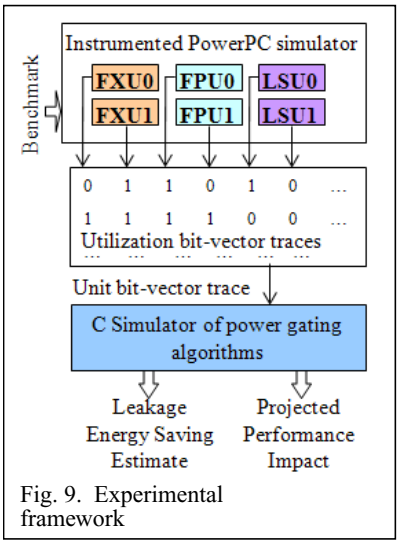

Fig. 9. Experimental framework

Table I. Target System Parameters

functional units	2 FXU, 2 FPU, 2 LSU
branch predictor	16K-entry BHT with 2 bits per entry to indicate direction of branch
instruction buffer	64 entries for each thread
decode width	8
L1D cache (within core)	64 KB, 8-way, line size 128 bytes
L1I cache (within core)	64 KB, 4-way, line size 128 bytes
L2 cache (on-chip)	4 MB, 8-way, line size 128 bytes
L3 cache (off-chip)	16 MB, 16-way, line size 128 bytes

4. EVALUATION METHODOLOGY

4.1 Simulation Methodology

To evaluate the results of applying our proposed mechanisms for power gating, we use processor functional unit utilization traces from all SPEC2006 benchmarks. All the solutions we compare are run on the same traces (100 million cycles simulation equivalent). Figure 9 depicts the generation of these traces and how we utilize them. To derive the unit-level utilization traces, we instrument a cycle-accurate performance simulator pre-configured to model an aggressive superscalar microprocessor core.

A high-level summary of the simulated microarchitecture is shown in Table 1. The model captures the details of the published core pipeline depth and super scalar execution semantics of each core within the POWER6 microprocessor [7][3]. Our intent was to set up the baseline power-gating experimental analysis using the IdleCount algorithm proposed by Hu et al. [3], while upgrading from an older POWER4-like microarchitecture (as used by

3. The results and analysis presented in this paper are not claimed to be an accurate representation of any real PowerPC product in the marketplace. In fact, as evident from published papers like [7], power-gating features are not part of server-class processor core logic used in designs like POWER6.

Hu et al.) to a more contemporary, POWER6-like model. We believe this new setting allows us to assess the leakage energy savings for the baseline power-gating algorithm as well as our new guarded, 2-level algorithms in a more realistic, modern setting.

The simulator is instrumented such that it reports the utilization of each processor unit on a cycle-by-cycle basis. Each row in the figure represents the utilization of all units during one cycle (a record in the trace). A value of one signifies that the unit was used during that cycle while a zero means the unit was idle. Each column holds the utilization of one particular unit for successive cycles. We then implemented our power gating algorithms and proposed bounding mechanisms in an analysis tool written in the C language. This analysis tool implements an energy model for power gating that allows us to obtain estimates of leakage energy savings. A limitation of this trace oriented framework is that it allows us only an approximate analysis of the potential performance impact a power gating scheme could have. We assign a fixed cost in terms of cycles lost whenever a power gated unit is awoken after being power gated. In our experiments we assign a 3-cycle performance penalty in this situation. However, in practice this wake-up penalty can be largely avoided. In many cases it is possible to know, at decode stage or from the issue queue, that a particular type of instruction is likely to execute within a predefined number of cycles. If decode-time signals are properly utilized towards this end, it is possible to wake-up the units proactively when they are needed, avoiding the performance cost altogether. However, we consider both the case when such a decode mechanism is present and the case when it is not, by assigning a penalty of zero or three cycles at wake-up. The advantage of our novel analysis framework, is that it allowed us to decouple the detailed, cycle-accurate performance model from the power-gating algorithm simulation, in such a manner that did not require us to modify the complex model semantics that were developed by a seasoned performance team. Only simple instrumentation code to monitor utilization traces for targeted units was required. We obtained traces of all SPEC2006 benchmarks (12 integer and 17 floating point) from the cycle-accurate PowerPC processor simulator alluded to above.

4.2 Energy Model

For estimating the energy saved by our proposed schemes, we refer to the energy model proposed by Hu et al. [3]. In this section we give a brief overview of the energy model.

Savings before full discharge. From equation (11) in Hu et al. [3], the energy savings in the i^{th} cycle after power gating and before the component is fully discharged, E^i, can be described by the formula:

$$E^i = E_L * i * (\mathrm{DIBL}/mV_t) * \Delta V^0$$

where E_L is the leakage power of the unit during one cycle, i is the number of cycles after power gating, DIBL is the drain-induced barrier lowering factor, V_t is the thermal voltage, and ΔV^0 is the voltage droop during the first cycle that the circuit is power gated. For current technology parameters, we obtain $E^i = 0.045 * i * E_L$.

Savings after full discharge. After full discharge, E^i equals E_L.[4]

Parameter values. The break-even point and energy overhead depend on the technology and the particular design macros. Hu et al. [3] used circuit-level simulations to validate viable ranges of these parameters, and they evaluated break-even points between 9-24. We

4. Although the point of full discharge is not identical with the break-even point for the precise constants used, we approximate that the two are the same. This small approximation allows a single energy savings value for every cycle after the break-even point, and we believe the difference to be in the noise of Hu et al.'s theoretical model [3].

use the same model input values as Hu et al., with two exceptions. We set the ratio between header device to the size of the power-gated circuit (W_H in their model) to 0.125, and we set the ratio between average leakage and switching energy dissipated per cycle (L in their model) to 0.3. Using equations (12) and (14) as in [3], we get a break-even point of 19 and energy overhead of $8*E_L$.

5. EXPERIMENTAL RESULTS

We evaluate the ability of our mechanisms to bound the *worst-case* energy penalty of applying power gating, and we analyze their impact on the *average* energy savings of power gating (Section 5.1) and the *average* performance loss (Section 5.2). We compare the following power gating solutions:

- The baseline IdleCount [3] that we evaluate for 2 values of the IdleCount idle_detect (5 and 15, data presented for the 5 value due to space constraints).
- IdleCount augmented with the Token Counting Guard (Token_IdleCount). The token bag is updated every 50 cycles and the bound is set for a maximum of 2% leakage energy loss over a guarantee interval of 100 monitoring intervals (a total of 5000 cycles). The length of the monitoring interval (50 cycles) was motivated by wanting to quickly adapt to workload changes and reduce extra logic.
- IdleCount augmented with the Success Monitor Switch (Succ_IdleCount in the figures). The Success Monitor Switch is invoked every 50 cycles.
- IdleCount with both the Success Monitor Switch and the Token Counting Guard (TokenSucc_IdleCount).

In our experiments, we consider the break-even point to be 19 and we vary the number of performance penalty cycles from zero to three. We present the potential performance impact as the percentage increase in cycles compared to the penalty incurred by the baseline. In reporting energy savings, we explicitly consider the leakage energy consumed during cycles added due to power gating. It is infeasible to evaluate our mechanisms for all possible parameter values or to evaluate them in the context of all previously developed power gating algorithms. However, we believe the chosen parameters and baseline are representative and that our results would be qualitatively similar for other parameters and baselines.

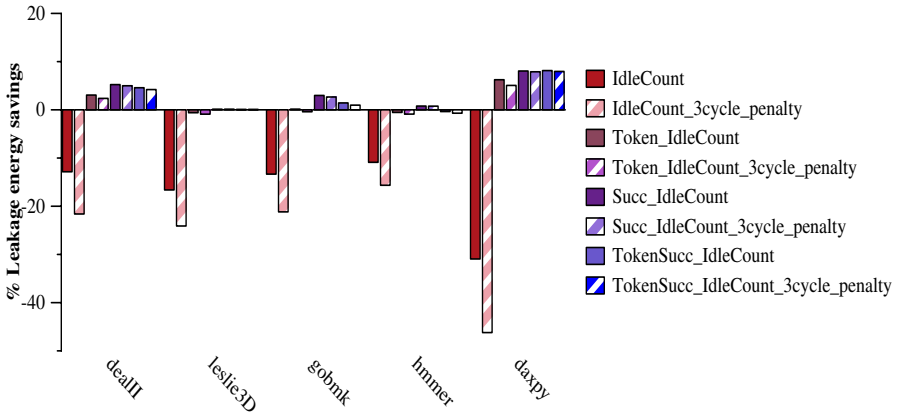

Fig. 10. Comparative energy savings for benchmarks wasting energy under IdleCount scheme

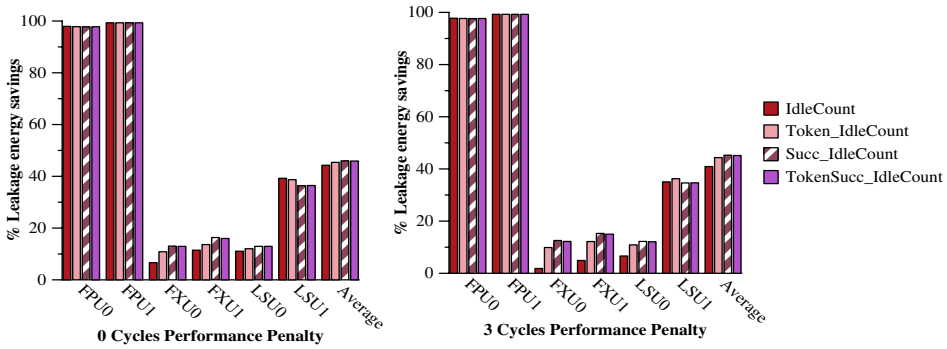

Fig. 11. Average energy savings (SPECint)

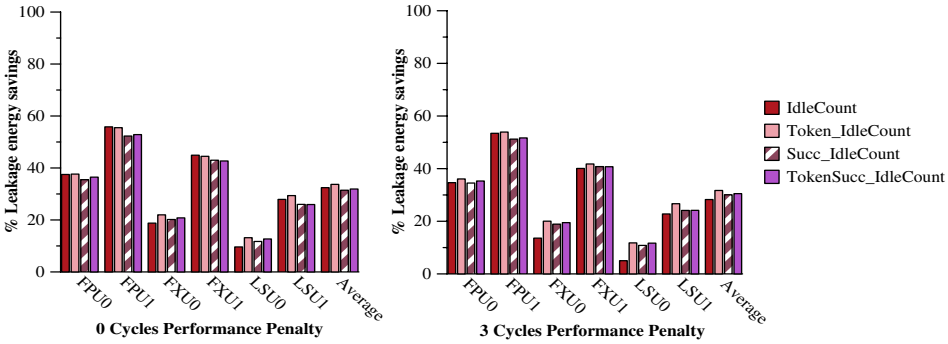

Fig. 12. Average energy savings (SPECfp)

5.1 Worst-Case and Average-Case Energy

Figure 10 compares the energy savings for the benchmarks that exhibited an energy penalty under IdleCount. We do not present the data for the other benchmarks, because our schemes have negligible impact on their behavior. There are two bars presented for each of the compared schemes. The solid bar represents the energy savings for a system using decode signals to mask the performance penalty while the striped bar represents the energy savings of a system with a 3-cycle penalty paid on unit wake-up. Success_IdleCount eliminates the energy penalties incurred by the baseline system and transforms them into energy savings for all benchmarks. TokenSucc_IdleCount has a marginal energy penalty of below 1% only for *hmmer*. Token_IdleCount exhibits a marginal energy penalty of below 1% for *leslie3D*, *gobmk*, and *hmmer*. Note that the bound that was set for both Token_IdleCount and TokenSucc_IdleCount schemes was a maximum loss of 2%, which means they respect the bound and function correctly.

Figure 11 and Figure 12 compare the average energy savings for an idle_detect threshold of 5 and a break-even point of 19 (for SPECint and SPECfp). The graphs on the left show data when we consider the performance penalty to be zero while the right figures show energy savings for a 3-cycle penalty on unit wake-up. We notice that in all four figures the Token_IdleCount scheme saves slightly more energy than the baseline. The same is true for the Success_IdleCount and TokenSuccess_IdleCount except for the SPECfp benchmarks for the 0-cycle performance penalty case. A trend can be seen where our proposed schemes perform increasingly better compared to the baseline when there is a performance penalty paid. The explanation is that our schemes incur a smaller performance penalty due to the fact that power gating is disabled when it is not useful. Due to the smaller

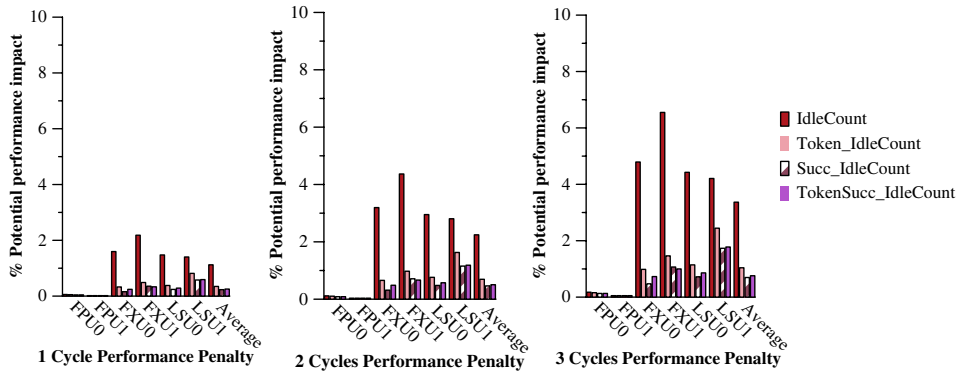

Fig. 13. Average Performance Impact (SPECint)

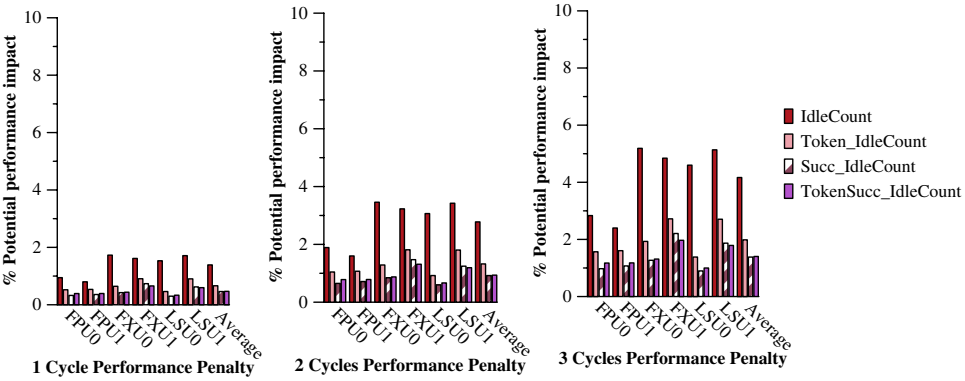

Fig. 14. Average Performance Impact (SPECfp)

performance penalty of our schemes the supplementary execution cycles due to power gating have a lower effect on our mechanisms. For a larger idle_detect of 15, our schemes are still advantageous in terms of average energy savings but only marginally so. Averaged across both values of the idle_detect, the Token_IdleCount saves 3.5% more energy, the TokenSuccess_IdleCount saves 2% more energy while the Success_IdleCount saves 1.3% less energy than the IdleCount baseline.

5.2 Performance

Figure 13 and Figure 14 show the worst-case potential performance impact of applying power gating with different wake-up penalties. We call it potential impact because much of it is likely to be masked in a real processor due to other causes of stalls. All our proposed schemes decrease the baseline impact to a large extent. This occurs because in many cases our schemes disable power gating when it was incurring an energy penalty. This disabling of power gating when it is not beneficial leads to a decrease in performance penalty as well. On average, the Token_IdleCount decreases the baseline performance impact by 51%, the Success_IdleCount by 71% and the TokenSuccess_IdleCount by 63%. In addition to the bound on the worst-case energy behavior we thus also obtain a significant decrease in average performance impact.

6. IMPLICATIONS FOR FUNCTIONAL CORRECTNESS AND VERIFICATION

Design verification consumes a significant proportion (up to 60%) of the total resources required for the creation of a new processor [1]. Considering the criticality of design veri-

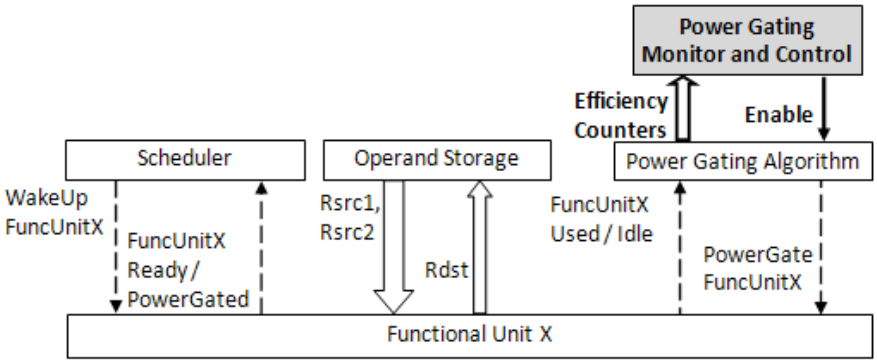

Fig. 15. Interface between Power Gating Scheme and the System

fication, we now discuss our expectations for the verification consequences of both power gating (Section 6.1) and our proposed mechanisms on verifying the processor's functional correctness (Section 6.2). Our discussion is restricted to implications on design verification, at the *microarchitectural* level, where we pursue our work. Thus we do not refer to circuit-level verification techniques and changes due to power gating that are not visible at microarchitectural level. We assume the initial processor design without power gateable units is correct.

In both discussions, we refer to Figure 15, which shows the interface between the functional unit that can be power gated and the rest of the system, namely the instruction scheduler, the operand storage component(s), and the power gating controller. For the baseline system, the capability to power gate the functional unit introduces four additional signals marked by dashed lines in Figure 15. The *FuncUnitXReady/PowerGated* signal informs the scheduler whether a power gateable unit is available for being issued an instruction during a cycle. The *WakeUpFuncUnitX* signal is used by the scheduler to wake up a power gated functional unit when there is work available for it. The other two signals represent the interface between the power gating scheme and the functional unit.

6.1 Verification Implications of Adding Power Gating Capability for Functional Units

The key to understanding why power gating functional units does not greatly increase microarchitectural design verification effort, with respect to a design without power gating, is that the scenario of a functional unit being unavailable due to power gating is very similar to the scenario in which the scheduler cannot issue an instruction to the functional unit due to a structural hazard. Assuming the initial processor design without power gateable units is correct, then its scheduler correctly identifies and handles the case when instructions with their operands ready cannot issue due to a structural hazard on the needed functional unit type. All that is necessary to maintain correctness, at microarchitectural level, after the addition of power gating, is for a new structural hazard signal to be generated by OR-ing the old signal with the *FuncUnitXPowerGated* signal.

The difference between unavailability due to a structural hazard and unavailability due to power gating is that we already know that a structural hazard will resolve. Thus, to make the verification equivalent, we must also verify that power gated units will eventually wake up, which involves verifying two properties:

- Property 1: We must verify that the scheduler correctly issues a wake-up signal to a unit when an instruction is ready for it and no other same-type units are available. A corollary is that the scheduler correctly identifies when a unit is unavailable to be issued instructions due to power gating.

- Property 2: The functional unit's power gating control logic needs to be verified to ensure that the functional unit always wakes up from a power gated state within a fixed number of cycles after the scheduler asserts the wake-up signal.

Verifying these two properties should not add a significant burden to microarchitectural level design verification because they can both be verified in isolation. Verifying Property 1 involves only the scheduler, by considering every possible value of its output *WakeUp-FuncUnitX* and input *FuncUnitXReady/PowerGated* interface signals. Verifying Property 2 involves only the functional unit controller and should not imply a system level verification effort.

6.2 Impact of Adding the Success Monitor and the Token Counting Mechanisms

We now analyze the impact of our proposed Success Monitor Switch and Token Counting Guard on verifying the processor's functional correctness, again at the microarchitectural level. The design we compare to is one that already enables power gating of processor execution units. We consider this initial design to be correct.

The mechanisms proposed in this work are illustrated in Figure 15 by the extra block on top of the Power Gating Algorithm (in bold). This level of Power Gating Monitor and Control interacts only with the Power Gating Algorithm block by reading the values of the efficiency counters and enabling or disabling the Power Gating Algorithm. Hence, our mechanisms do not change the initial interface between the functional unit and the rest of the system. The only change that occurs is that the Power Gating Algorithm can be disabled in which case the *PowerGateFuncUnitX* remains unasserted even when the unit is idle for longer than the idle_detect. The initial design needs to respond correctly to both an asserted or unasserted value of the *PowerGateFuncUnitX* signal, regardless of the length of its prior idle interval. Because of this, the addition of our mechanisms will not affect the functional correctness of the system compared to an initial design that allows for power gating of functional units.

7. RELATED WORK

Power gating mechanisms and algorithms. Narendra and Chandrakasan provide an overview of power gating and other techniques to decrease leakage energy [8]. Intel's Nehalem processor [6] incorporates power gating techniques at the core granularity (instead of the finer functional unit granularity we consider). Because Nehalem's power gating control algorithms have not been published, we cannot estimate whether adding our guard mechanisms would be beneficial. At the granularity of functional units, there have been several proposed power gating algorithms [2, 3, 9, 12], but all are capable of incurring energy penalties in the absence of a guard mechanism. At the circuit level, Jiang et al. [5] evaluate the benefits and costs of implementing power gating, while our focus here is on microarchitectural level mechanisms.

Energy management. Zeng et al. [13] introduce *currentcy* as a unifying abstraction used by the operating system for the management of devices that consume energy. Applications are allocated a certain currentcy that they can use during an epoch to satisfy their energy requirements. At a high level, the currentcy abstraction is similar to our token mechanism; however, we use tokens to bound the worst-case behavior of a power gating algorithm.

Speculation control. Speculation-control methods have been used in conjunction with branch predictors in order to save energy [4]. However, the power gating problem domain dealt with in this work is quite different. If we do not have a guard mechanism, such as the Token Counting Guard we propose, the original goal of energy reduction itself may be reversed into a net power increase situation. In contrast, the primary goal in branch predic-

tion is performance increase, and the speculation control mechanism is used to save wasted energy, at a small performance cost.

Other mechanisms for reducing leakage. Input vector control [11] applies an input pattern to a combinatorial circuit so as to decrease the leakage power while maintaining the same functional behavior. Adaptive body bias [10] is a post-silicon technique that reduces the effects of process variation on leakage power. These circuit-level schemes are complementary to power gating.

8. CONCLUSIONS

Computer architecture papers seldom address issues like the risk of adding a feature or modifying a microarchitecture. And adding a guard mechanism to bridge an exposed risk gap is hardly ever proposed to make the solution risk-free and acceptable in a real design setting. Power efficiency has been a major design constraint in recent years, and this constraint has naturally fueled the proliferation of publications that propose architectural solutions to manage power while preserving performance targets. The main risk that usually stands in the way of early adoption of such ideas is that of a bug that might escape pre-silicon verification. The bug could be a functional bug or a "performance" bug, where the term performance is used in a broad sense. In the case of dynamic power management architectures, the key bug of concern is: are there scenarios in which the proposed added feature might cause system power to increase, rather than the intended reduction objective? What if there is a breach in which, accidentally or by design, a workload is able to cause a power overrun that would damage the system or cause it to fail? Similar risk scenarios are considered by experienced design teams when considering ideas that propose to improve other metrics, like reliability. Thus, even if the added feature is not complicated enough to raise functional verification concerns, making sure that the intended feature functions without risk of ever incurring a negative benefit is paramount in such cases. Without such basic quality guarantees or safeguards, the proposed feature is unlikely to be accepted into a real design, even if the proposal is simple, elegant, and found to be very beneficial through simulation-based studies across target applications of interest. In this paper, we have presented what we believe is the first paper to conclusively demonstrate that an added feature—in this case, power gating—can be designed so as to eliminate risk.

Power efficiency is arguably the single most important metric in computer architecture today, and power gating is a broadly applicable and effective mechanism for improving this metric. Given that leakage power has rapidly grown into a significant fraction (up to 50% in some units) of total power in current and future technologies, power-gating is an especially relevant power reduction knob today. Because dynamic voltage scaling may be close to its limit—due to SRAM functional issues and soft error rate (SER) exacerbation, etc.—power-gating has increased in importance, as evidenced by its introduction in mainstream processors (like Intel's Nehalem). Being able to implement power-gating without risk is a major contribution that will enable seamless acceptance into quality-conscious and reliability-conscious high-end server systems. The proposed second-level guard mechanisms can be retrofitted into power management firmware (e.g., the on-chip power-control unit in Nehalem) or system software. Subsequently, aspects of the guard mechanism can be migrated into hardware as well.

ACKNOWLEDGMENTS

This material is based upon work supported (in part) by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002 and the National Science Foundation under grant CCF-0811290.

REFERENCES

[1] P. Bose, D. H. Albonesi, and D. Marculescu. Guest Editors' Introduction: Power and Complexity Aware Design. *IEEE Micro*, pages 8–11, Sept/Oct 2003.

[2] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing Static Leakage Energy in Microprocessor Functional Units. In *Proc. of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–332, Nov. 2002.

[3] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, P. Bose, and H. Jacobson. Microarchitectural Techniques for Power Gating of Execution Units. In *Proc. of the International Symposium on Low Power Electronics and Design*, Aug. 2004.

[4] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning Confidence to Conditional Branch Predictions. In *Proc. of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 1996.

[5] H. Jiang, M. Marek-Sadowska, and S. R. Nassif. Benefits and Costs of Power-Gating Technique. In *Proc. of the International Conference on Computer Design*, 2005.

[6] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *Proc. of the International Solid-State Circuits Conference*, Feb. 2009.

[7] H. Q. Le et al. IBM POWER6 Microarchitecture. *IBM Journal of Research and Development*, 51(6), 2007.

[8] S. G. Narendra and A. Chandrakasan, editors. *Leakage in Nanometer CMOS Technologies*. Springer-Verlag, 2006.

[9] S. Rele, S. Pande, S. Onder, and R. Gupta. Optimizing Static Power Dissipation by Functional Units in Superscalar Processors. In *Proc. of the International Conference on Compiler Construction*, Apr. 2002.

[10] J. W. Tschanz et al. Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Variations on Microprocessor Frequency and Leakage. *IEEE Journal of Solid-State Circuits*, 37(11):1396–1402, Nov. 2002.

[11] Y. Ye, S. Borkar, and V. De. A New Technique for Standby Leakage Reduction in High-Performance Circuits. In *Proc. of the Symposium on VLSI Circuits*, pages 40–41, 1998.

[12] A. Youssef, M. Anis, and M. Elmasry. Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units. In *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 371–384, Dec. 2006.

[13] H. Zeng, X. Fan, C. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proc. of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.