Microarchitectural Techniques for Power Gating of Execution Units

Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan,
Victor Zyuban, Hans Jacobson, Pradip Bose
IBM T. J. Watson Research Center

ABSTRACT

Leakage power is a major concern in current and future microprocessor designs. In this paper, we explore the potential of architectural techniques to reduce leakage through power-gating of execution units. This paper first develops parameterized analytical equations that estimate the break-even point for application of power-gating techniques. The potential for power gating execution units is then evaluated, for the range of relevant break-even points determined by the analytical equations, using a state-of-the-art out-of-order superscalar processor model. The power gating potential of the floating-point and fixed-point units of this processor is then evaluated using three different techniques to detect opportunities for entering sleep mode; ideal, time-based, and branch-misprediction-guided. Our results show that using the time-based approach, floating-point units can be put to sleep for up to 28% of the execution cycles at a performance loss of 2%. For the more difficult to power-gate fixed-point units, the branch misprediction guided technique allows the fixed-point units to be put to sleep for up to 40% more of the execution cycles compared to the simpler time-based technique, with similar performance impact. Overall, our experiments demonstrate that architectural techniques can be used effectively in power-gating execution units.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles - Pipeline processors

General Terms

Design, Performance

Keywords

low power, execution units, power-gating, microarchitecture

1. INTRODUCTION

The impact of CMOS technology scaling on power dissipation is well known. In particular, due to the scaling down of the threshold voltage, an exponential growth in subthreshold leakage current is expected with every crank of the technology wheel [1]. Similarly, scaling down of gate (and in particular, oxide) geometries is resulting in a very rapid growth of the gate leakage currents [6]. Without corrective measures at the device, circuit and/or microarchitecture-level, the total standby (leakage) power may well become the dominant part of the total power consumed by a microprocessor chip in future technologies.

Prior circuit-level approaches to leakage power reduction include: body-bias control [4], dual-threshold domino circuits [10], input vector control [9] and power-gating [14]. Architecture-level leakage power reduction techniques have focused primarily on SRAMs (the caches and buffers). Some techniques [5, 11] rely on accurate predictions of idle periods for different sections of the storage structures, and gate the supply voltage for these sections during the idle periods. Heo et al. [7] show that tristating the drivers of the bitlines reduces the leakage power associated with bitlines. More recently, Dropsho et.al [3] have explored analytical models to determine the sleep-mode activation policies for the integer functional units using a dual-threshold domino logic circuits. Rele et al. [15] use the compiler to identify the onset long idle periods for different functional units, and enable power gating for those units during these idle periods.

Many vendor products in the traditional “low power” embedded space (e.g. [2]) provide power-gating support in the form of “sleep” modes, typically under software (OS) control. The processor core, in such a system, can be power-gated off when the operating system detects a long idle loop. Exploiting workload phases and characteristics to dynamically power-gate off/on selected units within a processor pipeline, is a technique that to the best of our knowledge is not employed in any commercial processor yet.

In this paper we focus on this latter, hardware mechanism: namely, workload-driven, dynamic power-gating. We first explain the microarchitecture and circuit-level design parameters that need to be considered in investigating the overall power savings potential in such an approach. Subsequently, we present experimental results, based on detailed, cycle-accurate simulation with benchmark application programs, to quantify the power savings potential for a selected range of the parameter values. In presenting these results we consider: (a) the idealized maximum savings based on an “oracle” prediction of the onset of idle and active periods of a given processor execution unit; and (b) the actual savings, based on realistic overheads and two very simple microarchitectural prediction heuristics to determine the onset of “wakeup” or “power down”.

One of the simple predictors studied is a time-based technique to power gate an execution unit after observing a pre-determined number of idle cycles; and restarting the execution unit with a performance penalty once a pending operation is detected. The other technique studied uses the branch prediction mechanism to guide the gating of execution units. An execution unit is turned off as soon as a branch misprediction is detected.

2. POWER GATING

2.1 Fundamentals

Power gating is achieved by using a suitably sized header (Fig. 1) or footer transistor for a circuit block that is deemed to be a power-gating

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.
Copyright 2004 ACM 1-58113-929-2/04/0008 ...$5.00.

candidate. When the logic detects the onset of a sufficiently long idle period of the target circuit block, a “sleep” signal is applied to the gate of the header or footer transistor to turn-off the supply voltage to the circuit block. Similarly, once it is determined that the circuit block is being requested for use, the “sleep” signal is de-asserted to restore the voltage at the virtual Vdd.

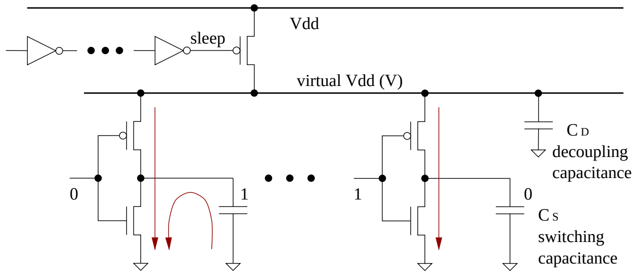

Figure 1: Using headers for power gating

Figure 2 shows the key intervals of power gating, assuming power gating with a header device. All derivations and results also apply to the footer device implementation.

The interval of inactivity begins at $t = 0$, and at $t = T_1$ ($T_{\text{idle detect}}$) the control circuit makes a decision to power-gate the unit. Until this point the unit still operates in the active mode, dissipating E^L_{cyc} leakage energy every cycle. During the interval $[T_1, T_2]$ ($T_2 - T_1 = T_{\text{idle delay}}$) the “sleep” signal is re-buffered and distributed to the header device incurring an overhead energy, $E_{\text{overhead 1}}$.

When the “sleep” signal is delivered to the gate of the header device at $t = T_2$, the voltage at the virtual Vdd V starts going down. If the leakage current were independent of the power supply, the voltage drop at the virtual Vdd would be linear in time, and savings in the leakage energy would begin only after the virtual Vdd is totally discharged ($t = T_4 = T_{full\,discharge}$). In reality, as the voltage at the virtual Vdd goes down, the leakage current also reduces, and the savings in leakage energy begins as soon as the “sleep” signal is asserted. As the voltage at the virtual Vdd goes down, the amount of leakage energy saved per cycle increases, resulting in a super-linear growth in the aggregate saved leakage energy.

At $t = T_4 = T_{\text{full discharge}}$ the reduction in voltage at the virtual Vdd saturates (not necessarily at zero, considering leakage through the header device). This leads to a constant leakage energy dissipated per cycle, and a linear growth in the aggregate saved leakage energy.

At $t = T_5$ the control logic detects the next busy interval, and the “sleep” signal is de-asserted, resulting in an energy overhead of $E_{\text{overhead 2}}$ dissipated for generating and driving the signal. At $t = T_6$ ($T_6 - T_5 = T_{\text{busy delay}}$) the header device is turned on, and during the interval $[T_6, T_7]$ ($T_7 - T_6 = T_{\text{wakeup}}$) the virtual Vdd is charged up to the $\mathrm{V_{dd}}$ level. As the virtual Vdd is charged up, the amount of leakage energy savings per cycle gradually reduces, reaching zero at T_7.

The break-even point $t = T_3$ ($T3 - T2 = T_{\text{break even}}$) is the point when the aggregate leakage energy savings $E_{ag\,saved}$ equals the energy overhead of switching on and off the header device, $E_{\text{overhead}} = E_{\text{overhead 1}} + E_{\text{overhead 2}}$. Depending on the size of the header device, the amount of decoupling capacitance, and other parameters (discussed later), the break-even point may occur either before or after the virtual Vdd is fully discharged.

Rather than giving simulation results specific to a particular CMOS technology which are inevitably subject to the accuracy of the leakage models, we derive analytical formulas expressing the key parameters of the power gating mechanism through easy-to-understand terms.

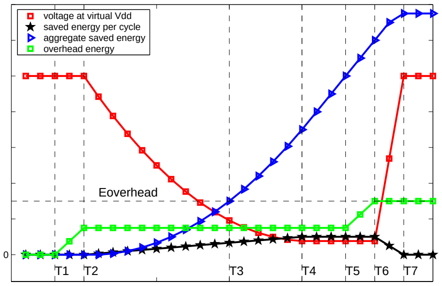

Figure 2: Key intervals in the power gating cycle

We use the conventional formula for the average switching energy dissipated per cycle in a particular macro or unit as

$$E^S_{cyc} = \frac{1}{2}\alpha C_S \mathrm{V_{dd}}^2, \tag{1}$$

where C_S is the total switching capacitance, including gate, source and drain capacitance of all transistors as well as wires; α is the switching factor, averaged over the whole macro, over all cycles including those when parts of the macro are clock-gated. The average leakage energy dissipated per clock cycle when the macro is *not* $\mathrm{V_{dd}}$-gated off is $E^L_{cyc} = TI_L\mathrm{V_{dd}}$, where T is the clock period, and I_L is the average leakage current through all leakage paths in the macro or unit. For the following analysis we introduce the *leakage factor* L as a ratio of the average leakage and switching energy dissipated per cycle, $L = E^L_{cyc}/E^S_{cyc}$.

When the power supply to the the macro is gated off, the voltage at all internal nodes that are in logical ‘one’ state (Figure 1) is gradually decreasing, tracking the voltage at the local supply distribution, $V(t)$. Assuming that on the average half of the internal nodes are in the state of logical ‘one’, and the total capacitance at the local power supply is C_D, the total internal capacitance that loses charge is $\frac{1}{2}C_S + C_D$. Notice that C_D includes the total source capacitance of all transistors connected to the local power supply (including the header), the wire capacitance of the virtual $\mathrm{V_{dd}}$ distribution and the decoupling capacitance. Furthermore, suppose that while the macro is $\mathrm{V_{dd}}$-gated, the voltage V at the local power supply drops by $\triangle V^i_{cyc}$ during cycle i. Then the amount of energy needed to restore the voltage at the local supply distribution to the $\mathrm{V_{dd}}$ level is

$$E^i_{cyc} = (C_D + \frac{1}{2}C_S)\mathrm{V_{dd}}\triangle V^i_{cyc}. \tag{2}$$

Notice that for the first cycle that the macro is $\mathrm{V_{dd}}$-gated ($i = 0$), $E^0_{cyc} \approx E^L_{cyc}$ since the voltage at the virtual $\mathrm{V_{dd}}$ changes by only a small amount in one cycle, and therefore the leakage current during the first cycle approximately equals the leakage current through the macro while it is active. Assuming that there is no leakage through the header device, and using (2) and (1) and the definition of the leakage factor L, the voltage drop during the first cycle that the macro is $\mathrm{V_{dd}}$-gated can be expressed as

$$\triangle V^0_{cyc} = \frac{1}{2}\alpha L \mathrm{V_{dd}} \frac{1}{(\frac{1}{2} + \frac{C_D}{C_S})}. \tag{3}$$

Since the leakage current depends on the voltage level at the local power supply network, $I_L = I_L(V)$, the voltage drop in subsequent cycles i that the macro is $\mathrm{V_{dd}}$-gated can be expressed as

$$\triangle V^i_{cyc} = \triangle V^0_{cyc} \frac{I_L(V)}{I_L(\mathrm{V_{dd}})}. \tag{4}$$

To qualitatively understand the dependence of leakage energy on fundamental parameters of the CMOS technology and characteristics of the microprocessor, we use the common expression for the subthreshold leakage current and the linear approximation for changes in the threshold voltage which is applicable in the vicinity of the nominal power supply:

$$I_L(V) = I_0 \exp\left(-\frac{V_T(V)}{mV_t}\right), \tag{5}$$

$$V_T(V) - V_T(\mathrm{V_{dd}}) = -\mathrm{DIBL}(V - \mathrm{V_{dd}}), \tag{6}$$

where DIBL is the drain-induce barrier lowering factor, which is 'typically' close to the value of 0.1, $V_t = kT/q \approx 25mV$ is the thermal voltage, and $m \approx 1.3$. Then

$$\triangle V^i_{cyc} = \triangle V^0_{cyc} \exp\left(\frac{\mathrm{DIBL}}{mV_t}(V(T_i) - \mathrm{V_{dd}})\right), \tag{7}$$

where $V(T_i)$ is the voltage at the local power distribution network i cycles after that the macro has been $\mathrm{V_{dd}}$-gated,

$$V(T_i) = \mathrm{V_{dd}} - \sum_{j=0}^{i-1} \triangle V^j_{cyc}. \tag{8}$$

For typical technology parameters, stated earlier $\frac{\mathrm{DIBL}}{mV_t} \approx 3$. Also, assuming the typically quoted values for general purpose microprocessors: $\alpha = 0.1$, $\mathrm{V_{dd}} = 1.0V$, $L = 0.5$, $\frac{C_D}{C_S} = 0.5$, we estimate $\triangle V^0_{cyc}$ in (3) to be 0.025. Then, for the first several cycles that the macro has been $\mathrm{V_{dd}}$-gated the argument of the exponent in (7) is a small negative number ($|x| < 0.25$), and a linear approximation $\exp(-x) \approx 1 - x$ can be used. Then, using (8), expression (7) can be re-written as:

$$\triangle V^i_{cyc} = \triangle V^0_{cyc}\left(1 - \frac{\mathrm{DIBL}}{mV_t}\sum_{j=0}^{i-1} \triangle V^j_{cyc}\right). \tag{9}$$

Taking advantage of the small magnitude of $\frac{\mathrm{DIBL}}{mV_t}\triangle V^0_{cyc}$, and neglecting second-order terms of the form $\left(\frac{\mathrm{DIBL}}{mV_t}\triangle V^0_{cyc}\right)^2$, the above expression for $\triangle V^i_{cyc}$ can be reduced as follows:

$$\triangle V^i_{cyc} \approx \triangle V^0_{cyc}\left(1 - i\frac{\mathrm{DIBL}}{mV_t}\triangle V^0_{cyc}\right). \tag{10}$$

This formula can only be used for small values of i, such that $\triangle V^i_{cyc} > 0$. For $i > N_{\text{full discharge}}$ (marked as T_4 in Figure 2), the local supply distribution is fully discharged and $\triangle V^i_{cyc} = 0$.

Using expressions (2) and (10), the amount of energy saved during cycle i due to $\mathrm{V_{dd}}$-gating the macro can be expressed as

$$E^i_{cyc\,saved} = E^L_{cyc} - E^i_{cyc} = E^L_{cyc} i \frac{\mathrm{DIBL}}{mV_t}\triangle V^0_{cyc}. \tag{11}$$

After the virtual $\mathrm{V_{dd}}$ is fully discharged, ($i > N_{\text{full discharge}}$) $E^i_{cyc\,saved} = E^L_{cyc}$.

For $N < N_{\text{full discharge}}$ the total (aggregate) energy saved over N cycles that the macro has been $\mathrm{V_{dd}}$-gated can be expressed as

$$E^N_{ag\,saved} = \sum_{i=0}^{N} E^i_{cyc\,saved} = E^L_{cyc}\frac{\mathrm{DIBL}}{mV_t}\frac{N^2}{2}\frac{\alpha L \mathrm{V_{dd}}}{2(\frac{1}{2} + \frac{C_D}{C_S})}. \tag{12}$$

For $i > N_{\text{full discharge}}$ the dependence of $E^N_{ag\,saved}$ on N becomes linear, $E^N_{ag\,saved} = E^{N_{\text{full discharge}}}_{ag\,saved} + (N - N_{\text{full discharge}})E^L_{cyc}$.

We estimate the amount of switching energy dissipated in the rebuffering network driving the header device, E_{overhead}, as follows. Let W_H be the ratio of the total area of the header device to the area of the clock-gated macro. Then, the gate capacitance of the header device C_{header} is approximately $W_H\times$ the total gate capacitance of all transistors in the macro (assuming the same density of gate area in the header and the rest of the macro). The latter is responsible for approximately half of the total switching capacitance in the macro C_S. Then in order to turn the header device on and off (to gate and un-gate the macro), the energy overhead is:

$$E_{\text{overhead}} = 2C_{\text{header}}\mathrm{V_{dd}^2} \approx 2W_H\frac{1}{2}C_S\mathrm{V_{dd}^2} = 2\frac{W_H}{\alpha}E^S_{cyc} \tag{13}$$

Using this result (note that the multiplier 2 takes care of the rebuffering network to drive the header device.) and (12), the number of cycle needed for the macro to be $\mathrm{V_{dd}}$-gated to compensate for the overhead of switching the header device, is found from $E^N_{ag\,saved} = E_{\text{overhead}}$ as:

$$N_{\text{break even}} = 2\frac{1}{L\alpha}\sqrt{\frac{mV_tW_H}{\mathrm{V_{dd}}\cdot\mathrm{DIBL}}(1 + 2\frac{C_D}{C_S})} \tag{14}$$

For the typical technology parameters, stated earlier, and typically quoted ratio of the size of the header device to the size of the macro, $W_H = 0.1$ we estimate the number of cycles until the break-even point as $N_{\text{break even}} \approx 10$. Notice that for the given assumptions the "break-even" point occurs before the local power supply distribution is fully discharged, $N_{\text{break even}} < N_{\text{fulldischarge}}$. Larger size of the decoupling capacitance C_D and the header transistors W_H increase $N_{\text{break even}}$, whereas, higher value of the DIBL effect factor and the leakage factor L reduce $N_{\text{break even}}$.

2.2 Power Gating Potential for Execution Units

We now quantify the power gating potential for various execution units of a state-of-the-art out-of-order superscalar processor model (details in Section 3) using several different applications from SPEC2K suite. For this study we assume a perfect predictor that can predict the idle intervals of these units with no delay (i.e., $T_{idledelay}$=0 and $T_{busydelay}$=0). Note that the unit uses T_{wakeup} (interval $T_7 - T_6$ in Figure 2) cycles out of the idle period so that it is ready for operation at the start of the busy period, thereby there is no performance penalty incurred. The following equations are used to estimate the fraction of cycles the units can be power-gated assuming that T_{idle} is greater than $T_{overhead}$.

$$T_{overhead} = (T_{wakeup} + T_{breakeven})$$

$$Opp_{cycles} = \sum_{i=1}^{allidleintv}(Tidle_i - T_{overhead})$$

$$PowerGate_{pot} = \frac{Opp_{cycles}}{totalexecutioncycles}$$

For example, the sequence of activity bits of some unit "11110000011111100001111000000111" has three idle intervals. Assuming $T_{overhead} = 3$, the unit can be power-gated for 6 (calculated as $OppCycles = (5-3)+(4-3)+(6-3)$) cycles out of a total execution time of 32 cycles, thereby achieving a 19% $PowerGate_{pot}$.

Figures 3 and 4 show the power gating potential for various execution units using different $T_{overhead}$ values averaged across a set of SPEC2K benchmarks. As expected the floating point benchmarks have more opportunity to power-gate the fixed point unit (fxu) compared to the integer benchmarks. Similarly, the integer benchmarks do not have any floating point operations and therefore the floating point unit (fpu) can be power-gated for the entire run of the application.

Different execution units will require different $T_{overhead}$ to effectively power-gate their logic, and the total power gating potential can be determined independently for each of the execution units. For example, if the fxu requires 9 cycles and fpu requires 14 cycles of

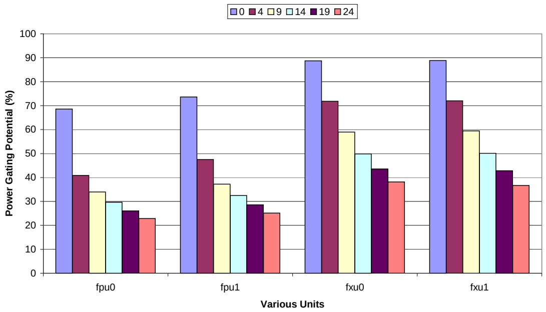

Figure 3: Power gating potential averaged across SPEC2K floating point applications for various values of $T_{overhead}$. FPU: Floating point unit, FXU: Fixed point unit.

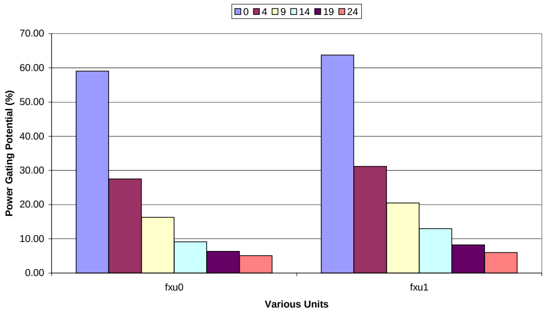

Figure 4: Power gating potential averaged across SPEC2K integer applications for various values of $T_{overhead}$

$T_{overhead}$, we see from Figure 3 that we can achieve power gating potential numbers of up to 60% each for the two fixed point units, and over 30% for the floating point units.

3. EVALUATION METHODOLOGY

We use a generic, parameterized, out-of-order superscalar processor model called Turandot [13] with configuration shown in Table 1. The modeled microarchitecture is similar to a current generation POWER4 like microprocessor [16]. This research simulator was calibrated against a pre-RTL, detailed, latch-accurate processor model [12].

Processor Core	
Decode rate	5 instructions per cycle
Issue queues	fxu/lsu queue (2x20), fpu queue (2x10)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BRU
Physical registers	128 GPR, 128 FPR
Branch predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector, all with 1-bit entries
Memory Hierarchy	
L1 Dcache Size	32KB, 2-way, 128B blocks
L1 Icache Size	64KB, 4-way, 128B blocks
L2 I/D	1MB, 4-way LRU, 128B blocks 9-cycle latency
Memory Latency	77 cycles

Table 1: Configuration of simulated processor

In this paper, we report experimental results based on PowerPC traces of a set of 21 SPEC2000 benchmarks, namely, *ammp*, *applu*, *apsi*, *art*, *bzip2*, *crafty*, *equake*, *facerec*, *gap*, *gcc*, *gzip*, *lucas*, *mcf*, *mesa*, *mgrid*, *perl*, *sixtrack*, *swim*, *twolf*, *vpr* and *wupwise*. For all the results, in reporting average statistics, we use geometric mean across the corresponding benchmark suite (SPECINT and SPECFP). These traces were generated using the tracing facility called *Aria* within the MET toolkit [13]. The particular SPEC2000 trace repository used in this study was created by using the full reference input set. However, sampling was used to reduce the total trace length to 100 million instructions per benchmark program. A systematic validation study to compare the sampled traces against the full traces was done in finalizing the choice of exact sampling parameters [8].

4. TIME-BASED POWER GATING

In this and the following sections, we describe two techniques to dynamically power-gate execution units at program runtime. For ease of explanation, the $T_{idledelay}$ component will be lumped into $T_{breakeven}$ value. Similarly, the $T_{busydelay}$ will be lumped into T_{wakeup} value.

One simple technique to dynamically power gate execution units, is to observe the state of a execution unit, and turn it off after seeing a streak of idle cycles. Similar techniques have been used for cache memories [11, 5] and have been shown to produce significant leakage savings with minimal performance impact. To apply such a technique to execution units, a state machine, as shown in Figure 5, can be logically associated with each execution unit. In normal execution mode, the execution unit is in the *idle_detect* state, fully-powered and ready for execution. When it is detected that the number of consecutive idle cycles seen exceeds the threshold $T_{idledetect}$ cycles, the unit is power-gated and transfers to an interim state denoted as *uncompensated*. If the unit continues to stay idle for another $T_{breakeven}$ cycles, then it will transfer to the *compensated/ sleep* state. A unit is power-gated in both *compensated* and *uncompensated* states. However, only *compensated* state sees net leakage savings, while the leakage savings in *uncompensated* state are used to compensate the dynamic power overhead of power-gating. Hence, in this study we measure the percentage of cycles a unit stays in the *compensated/ sleep* mode as an indication of net leakage savings with power-gating.

While in *compensated* or *uncompensated* state, an execution unit must wake up to serve an instruction that becomes ready. This is because in our simulated processor, each issue queue serves to one particular execution unit [1]. The number of cycles needed for an execution unit to transfer from sleep mode to a transient state *wakeup*, and finally to normal execution mode, is denoted as T_{wakeup}. If the unit is not yet fully powered-down (i.e., at the first several cycles of sleep mode), the time it takes to wake up the unit will be less than the full wakeup latency of T_{wakeup} cycles. Also, some leakage savings could be achieved in the wakeup process, before the unit is fully powered. In our experiment, we take a conservative approach by always charging a full wakeup latency, and assuming there is no leakage savings during wakeup (T6 to T7 in Figure 2).

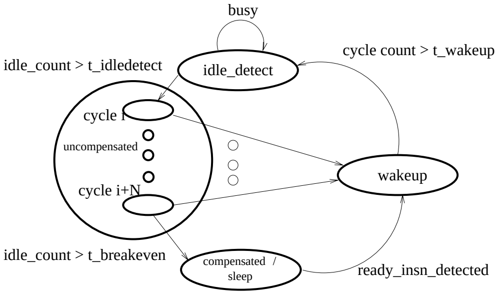

Figure 5: State machine of an execution unit when power gating is engaged

From the above description it is clear that the effectiveness of a

[1] If the issue queue is shared between multiple execution units, when a ready instruction is detected the issue queue has the option of not waking up a powered-down unit and instead waiting for an already powered-up one (which is busy at current cycle).

time-based dynamic power-gating technique hinges upon the value of the three parameters, namely, $T_{breakeven}$, T_{wakeup}, and $T_{idledetect}$. $T_{breakeven}$ represents number of cycles it takes to compensate for the energy overhead of switching the header. T_{wakeup} represents the timing overhead of repowering a powered-down unit, and determines the performance impact of power-gating. Both $T_{breakeven}$ and T_{wakeup} are mainly decided by circuit design limits, the third parameter, $T_{idledetect}$, provide an architecture-level knob to fine tune the trade-off between leakage saving opportunities and performance loss. More specifically, with a large $T_{idledetect}$, idle intervals that are relatively short are skipped by the power-gating mechanism, and consequently some leakage savings opportunities are given up for reduced performance loss.

Since the floating point applications stress the *fpu*, and the integer applications stress the *fxu*, we present the percentage of cycles spent in sleep mode by the *fpu* and the *fxu* using the floating point and integer applications, respectively. For the rest of the paper, the fraction of cycles spent in the sleep mode by an execution unit of a given type is determined as follows:

$$\frac{1}{n * total\ execution\ cycles} * \sum_{i=0}^{i=n} (cycles\ in\ sleep\ mode)_i$$

$$where\ n = number\ of\ units$$

Figures 6 and 7 show the tradeoff between leakage savings and performance loss for the *fpu*. As we can see from the figures, the percentage of cycles spent in sleep mode for *fpu*, decreases roughly linearly with increasing $T_{idledetect}$ values. This indicates that although long idle periods occur less frequently compared to short idle periods, the total number of sleep cycles derived from the long and short idle periods are roughly equal once these periods are weighted by the length of their sleep cycles. The performance, on the other hand, improves significantly when $T_{idledetect}$ increases from 1 cycle to 6 cycles, and then gradually reaches the performance of the base case, where power-gating is disabled. The big performance jump from 1 cycle to 6 cycles of $T_{idledetect}$ indicates the presence of short idle periods and these are not amenable for power gating. Though the number of such idle periods are large, power-gating them causes a significant performance loss since each of them will incur a timing overhead of T_{wakeup} cycles. This also explains the unexpected dip at $T_{idledetect}$ = 1 in Figure 6: though the absolute number of cycles *fpu* in sleep mode is much larger at $T_{idledetect}$ = 1 cycle compared to that of 6 cycles, the large performance degradation causes much more execution cycles leading to a much larger denominator in the above equation.

In general, long idle periods coupled with smaller values of $T_{breakeven}$ and T_{wakeup}, help achieve large leakage reductions, and mitigate the overall performance loss. Overall, a $T_{idledetect}$ of 6 - 12 cycles achieves a good balance between performance loss and power savings.

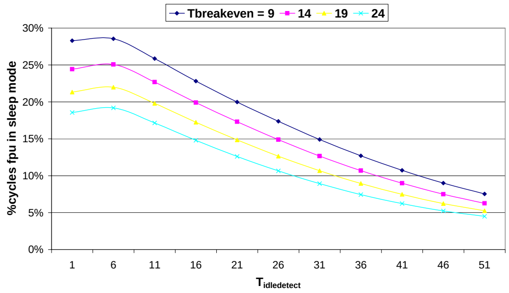

Figure 6: Percent of cycles in sleep mode for FPU (including FPU0 and FPU1) with different $T_{idledetect}$ and $T_{breakeven}$ = one of 9, 14, 19, or 24 cycles. T_{wakeup} is fixed at 3 cycles.

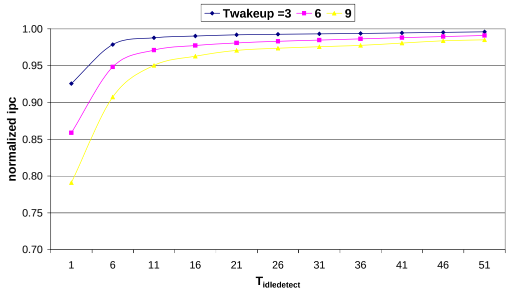

Figure 7: Average IPC of SPECFP2K benchmarks with different $T_{idledetect}$ and T_{wakeup} values. $T_{breakeven}$ is fixed at 9 cycles. IPC is normalized to the base case where power gating is disabled.

Figure 8 and Figure 9 show similar curves for fixed-point unit when the SPEC2K integer benchmarks are run. Unlike Figure 6, where the curves are roughly linear over the full range of $T_{idledetect}$, Figure 8 demonstrates a trend that can be divided into two segments at the turn point of around 16 cycles. Before the turn point, the curve drops very fast, indicating that very short idle periods dominate the total length of idle periods in SPEC2K integer benchmarks. We know for integer applications, loads/stores and branches typically occur once every 4 or 5 instructions. This implies that the fixed point operations which provide the source operands for these loads/stores and branches ought to occur once every 4 or 5 instructions as well. We observe that this frequency of fixed point operations correlates well with the predominantly short idle times observed in Figure 8. The long idle periods are very rare, with majority of them observed soon after an L2 miss is detected.

Overall, we found that using the same value of T_{wakeup} the FXU running SPEC2K integer benchmarks has not only less opportunities for power-gating compared to the FPU running SPEC2K floating-point benchmarks, but also incurrs much larger performance loss. In the next section, we propose an architectural technique that achieves a better tradeoff between performance loss and power savings in FXU for integer benchmarks.

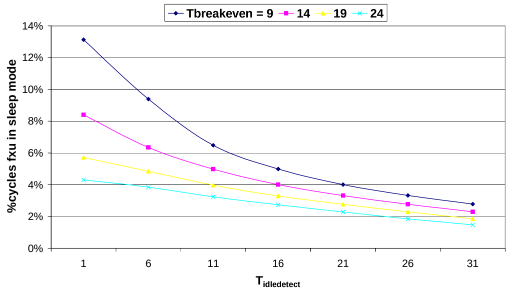

Figure 8: Percent of cycles in sleep mode for FXU (including FXU0 and FXU1) with different $T_{idledetect}$ and $T_{breakeven}$ values. T_{wakeup} is fixed at 3 cycles. The four curves correspond to $T_{breakeven}$ of 9, 14, 19, 24 cycles, respectively.

5. BRANCH PREDICTION GUIDED POWER-GATING

In the previous section, we showed the effectiveness of a time-based technique to power gate the *fpu* with only a small performance impact. Our results also showed that since the *fxu* typically had short idle periods, there weren't many power gating opportunities using the time-based technique. In this section, we augment the simple time-based technique by using the outcome of branch prediction to guide

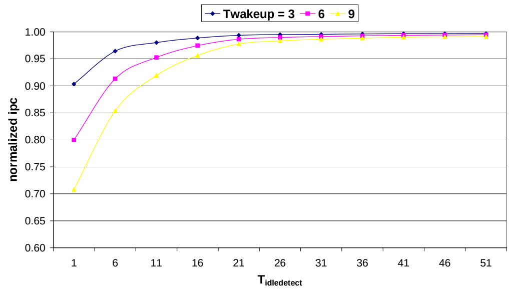

Figure 9: Average IPC of SPECINT2K benchmarks with different $T_{idledetect}$ and T_{wakeup} values. $T_{breakeven}$ is fixed at 9 cycles.

the power gating of the execution units, especially the *fxu*. Branch mispredictions are well-known as highly disruptive events for the performance of speculative out-of-order microprocessors; but, they also create opportunities for employing energy saving techniques such as power-gating of the execution units. In our processor model, we predict branches using a predictor, and continue execution along the predicted control flow path. When a branch is mispredicted, instructions following the branch are flushed from the pipeline, and the fetch engine is then re-directed to the correct path of execution. Hence, in the cycles following a branch misprediction, a large part of the issue queue is flushed and the execution units are likely to be idle (until the instructions from the correct path are re-fetched). We propose a new branch prediction guided power-gating technique, in which whenever a branch misprediction is detected, all the *fxu*s (if they are idle) are transferred to *uncompensated* state immediately , instead of waiting for the full $T_{idledetect}$ period. In Figure 10, we compare such a technique with the baseline time-based technique described in the previous section. Relative to a time-based technique we observe higher percent of cycles in sleep mode for a given performance loss, and a smaller performance loss for any given percent of cycles in sleep mode while using the branch prediction guided technique. Furthermore, the branch prediction guided technique effectively reduces the value of $T_{idledetect}$ by aggressively gating the *fxu* as soon as a misprediction is detected. For example, in Figure 10 we see that the the branch prediction guided technique has the same performance loss for $T_{idledetect}$ values ranging from 11 to 31 cycles. These results show that similar techniques can be devised based on other architecture-level events such as L2 cache misses, instruction cache misses, and rejects of load instructions from the pipeline after data cache misses. This is a promising venue that deserves further investigation.

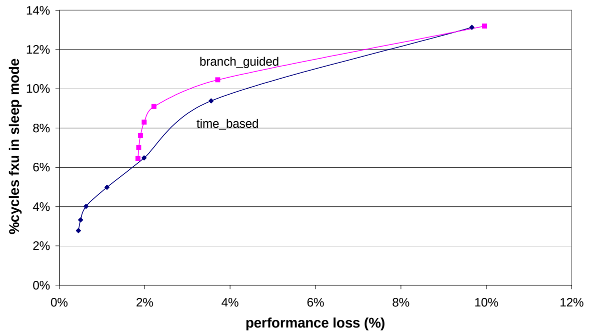

Figure 10: Percent of cycles FXUs are in sleep mode versus performance degradation for both techniques when running SPECINT2K benchmarks. $T_{breakeven}$ and T_{wakeup} are fixed at 9 and 3, respectively. $T_{idledetect}$ varies from 1 (far right) to 31 (left).

6. CONCLUSION

This paper demonstrates the potential of power gating the execution units of a state-of-the-art out-of-order superscalar processor model. A parameterized analytical model is derived that determines the breakeven point for power gating, which, for a typical modern technology, is shown to be close to 10 clock cycles at a static to dynamic power ratio of 33/67. This result indicates that there is ample opportunity for power gating of execution units, where idle periods of more than 10 cycles occur frequently. Two simple, yet effective, architecture-level techniques to power gate execution units are presented. A time-based technique is shown to enable the floating-point units to enter sleep-mode for up to 28% of the total execution cycles in floating point benchmarks at a performance penalty of 2%. A more sophisticated technique based on branch prediction is shown to enable fix-point units to enter sleep-mode by up to 40% more for a given performance in comparison to time-based technique. These experiments demonstrate that a substantial leakage power reduction can be achieved in processor execution units through power gating based on architectural techniques.

7. REFERENCES

[1] BORKAR, S. Design Challenges of Technology Scaling. *IEEE Micro 19*, 4 (1999).

[2] CLARK, L., DEMMONS, S., DEUTSCHER, N., AND RICCI, F. Standby Power Management for a 0.18um Microprocessor. In *ISLPED* (2002).

[3] DROPSHO, S., KURSUN, V., ALBONESI, D. H., DWARKADAS, S., AND FRIEDMAN, E. G. Managing Static Leakage Energy in Microprocessor Functional Units. In *MICRO* (2002).

[4] DUARTE, D., TSAI, Y. F., VIJAYKRISHNAN, N., AND IRWIN, M. J. Evaluating Run-Time Techniques for Leakage Power Reduction. In *ASPDAC* (2002).

[5] FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., AND MUDGE, T. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *ISCA* (2002).

[6] HAMZAOGLU, F., AND STAN, M. R. Circuit-Level Techniques to Control Gate Leakage for sub-100nm CMOS. In *ISLPED* (2002).

[7] HEO, S., BARR, K., HAMPTON, M., AND ASANOVIC, K. Dynamic Fine-Grain Leakage Reduction Using Leakage-Biased Bitlines. In *ISCA* (137-147, 2002).

[8] IYENGAR, V., TREVILLYAN, L. H., AND BOSE, P. Representative Traces for Processor Models with Infinite Cache. In *HPCA* (1996).

[9] JOHNSON, M., SOMASEKHAR, D., CHIOU, L., AND ROY, K. Leakage Control With Efficient Use of Transistor Stacks in Single Threshold CMOS . *IEEE Transactions on VLSI Systems 10* (2002).

[10] KAO, J., AND CHANDRAKASAN, A. Dual-Threshold Voltage Techniques for Low-Power Digital Circuits. *IEEE Journal of Solid State Circuits 35* (2000).

[11] KAXIRAS, S., HU, Z., AND MARTONOSI, M. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *ISCA* (2001).

[12] MOUDGILL, M., BOSE, P., AND MORENO, J. H. Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration. In *IPCCC* (1999).

[13] MOUDGILL, M., WELLMAN, J. D., AND MORENO, J. H. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro 19*, 3 (1999).

[14] POWELL, M., YANG, S., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In *ISLPED* (2000).

[15] RELE, S., PANDE, S., ONDER, S., AND GUPTA, R. Optimizing Static Power Dissipation by Functional Units in Superscalar Processors. In *Int'l Conf. on Compiler Construction* (2002).

[16] TENDLER, J. M., DODSON, J. S., FIELDS, J. S., LE, H., AND SINHAROY, B. POWER4 System Microarchitecture. *IBM Journal Research and Development 46*, 1 (2002).