Drowsy Caches: Simple Techniques for Reducing Leakage Power

Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, Trevor Mudge

krisztian.flautner@arm.com
ARM Ltd
110 Fulbourn Road
Cambridge, UK CB1 9NJ

{kimns, stevenmm, blaauw, tnm}@eecs.umich.edu
Advanced Computer Architecture Lab
The University of Michigan
1301 Beal Ave. Ann Arbor, MI 48109-2122

Abstract

On-chip caches represent a sizable fraction of the total power consumption of microprocessors. Although large caches can significantly improve performance, they have the potential to increase power consumption. As feature sizes shrink, the dominant component of this power loss will be leakage. However, during a fixed period of time the activity in a cache is only centered on a small subset of the lines. This behavior can be exploited to cut the leakage power of large caches by putting the cold cache lines into a state preserving, low-power drowsy mode. Moving lines into and out of drowsy state incurs a slight performance loss. In this paper we investigate policies and circuit techniques for implementing drowsy caches. We show that with simple architectural techniques, about 80%-90% of the cache lines can be maintained in a drowsy state without affecting performance by more than 1%. According to our projections, in a 0.07um CMOS process, drowsy caches will be able to reduce the total energy (static and dynamic) consumed in the caches by 50%-75%. We also argue that the use of drowsy caches can simplify the design and control of low-leakage caches, and avoid the need to completely turn off selected cache lines and lose their state.

1. Introduction

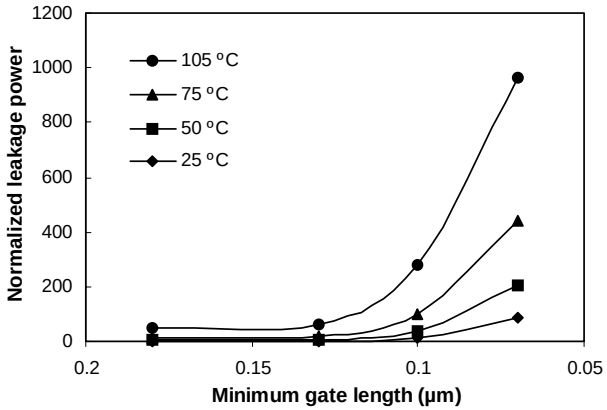
Historically one of the advantages of CMOS over competing technologies (e.g. ECL) has been its lower power consumption. When not switching, CMOS transistors have, in the past, consumed negligible amounts of power. However, as the speed of these devices has increased along with density, so has their leakage (static) power consumption. We now estimate that it currently accounts for about 15%-20% of the total power on chips implemented in high-speed processes. Moreover, as processor technology moves below 0.1 micron, static power consumption is set to increase exponentially, setting static power consumption on the path to dominating the total power used by the CPU (see Figure 1).

Various circuit techniques have been proposed to deal with the leakage problem. These techniques either completely turn off circuits by creating a high-impedance path to ground (gating) or trade off increased execution time for reduced static power consumption. In some cases, these techniques can be implemented entirely at the circuit level without any changes to the architecture or may involve only simple architectural modifications. The on-chip caches are one of the main candidates for leakage reduction since they contain a significant fraction of the processor's transistors.

Approaches for reducing static power consumption of caches by turning off cache lines using the gated-V_{DD} technique [1] have been described in [2][3]. These approaches reduce leakage power by selectively turning off cache lines that contain data that is not likely to be reused. The drawback of this approach is that the state of the cache line is lost when it is turned off and reloading it from the level 2 cache has the potential to negate any energy savings and have a significant impact on performance. To avoid these pitfalls, it is necessary to use complex adaptive algorithms and be conservative about which lines are turned off.

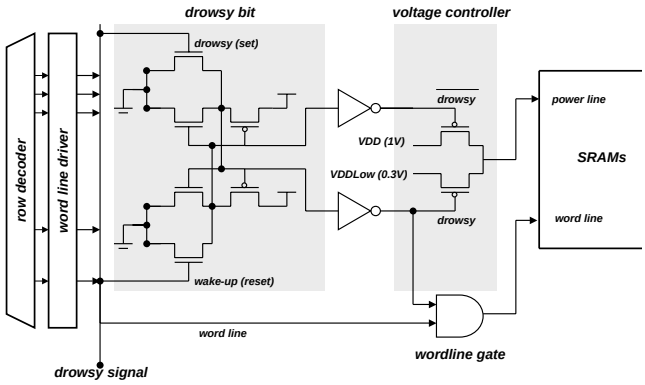
Turning off cache lines is not the only way that leakage energy can be reduced. Significant leakage reduction can also be achieved by putting a cache line into a low-power drowsy mode. When in drowsy mode, the information in the cache line is preserved; however, the line must be reinstated to a high-power mode before its contents can be accessed. One circuit technique for implementing drowsy caches is

FIGURE 1. Normalized leakage power through an inverter



The circuit simulation parameters including threshold voltage were obtained from the Berkeley Predictive Spice Models [4]. The leakage power numbers were obtained by HSPICE simulations.

FIGURE 2. Implementation of the drowsy cache line



Note that, for simplicity, the word line, bit lines, and two pass transistors in the drowsy bit are not shown in this picture.

adaptive body-biasing with multi-threshold CMOS (ABB-MTCMOS) [5], where the threshold voltage of a cache line is increased dynamically to yield reduction in leakage energy. We propose a simpler and more effective circuit technique for implementing drowsy caches, where one can choose between two different supply voltages in each cache line. Such a dynamic voltage scaling or selection (DVS) technique has been used in the past to trade off dynamic power consumption and performance [6][7][8]. In this case, however, we exploit voltage scaling to reduce *static power* consumption. Due to short-channel effects in deep-submicron processes, leakage current reduces significantly with voltage scaling [9]. The combined effect of reduced leakage current and voltage yields a dramatic reduction in leakage power.

On a per-bit basis, drowsy caches do not reduce leakage energy as much as those that rely on gated-V_{DD}. However, we show that for the total power consumption of the cache, drowsy caches can get close to the theoretical minimum. This is because the fraction of total energy consumed by the drowsy cache in low power mode (after applying our algorithms) tends to be only about 25%. Reducing this fraction further may be possible but the pay-off is not great (Amdahl's Law). Moreover, since the penalty for waking up a drowsy line is relatively small (it requires little energy and only 1 or 2 cycles, depending on circuit parameters), cache lines can be put into drowsy mode more aggressively, thus saving more power.

Figure 2 shows the changes necessary for implementing a cache line that supports a drowsy mode. There are very few additions required to a standard cache line. The main additions are a drowsy bit, a mechanism for controlling the voltage to the memory cells, and a word line gating circuit. In order to support the drowsy mode, the cache line circuit includes two more transistors than the traditional memory circuit. The operating voltage of an array of memory cells in the cache line is determined by the voltage controller, which switches the array voltage between the high (active) and low (drowsy) supply voltages depending on the state of the drowsy bit. If a drowsy cache line is accessed, the drowsy bit is cleared, and consequently the supply voltage is switched to high V_{DD}.The wordline gating circuit is used to prevent accesses when in drowsy mode, since the supply voltage of the drowsy cache line is lower than the bit line precharge voltage; unchecked accesses to a drowsy line could destroy the memory's contents.

Whenever a cache line is accessed, the cache controller monitors the condition of the voltage of the cache line by reading the drowsy bit. If the accessed line is in normal mode, we can read the contents of the cache line without losing any performance. No performance penalty is incurred, because the power mode of the line can be checked by reading the drowsy bit concurrently with the read and comparison of the tag. However, if the memory array is in drowsy mode, we need to prevent the discharge of the bit lines of the memory array because it may read out incorrect data. The line is woken up automatically during the next cycle, and the data can be accessed during consecutive cycles.

In this paper we focus on the policy implications of using L1 drowsy data caches. Since, compared to the L1 cache, the impact of an extra cycle of wake-up on the L2 access latencies is small, all lines in an L2 cache can be kept in drowsy mode without significant impact on performance. This intuition is confirmed by the data presented in [10]. In Section 2 and Section 3 we evaluate the design trade-offs between simple drowsy policies. We argue that the simplest policy of periodically putting the entire cache into drowsy mode does about as well as a policy that tracks accesses to cache lines. Section 4 provides details about various circuit techniques for reducing leakage power and Section 5 evaluates the impact of drowsy caches on energy consumption.

2. Policies

The key difference between drowsy caches and caches that use gated-V_{DD} is that in drowsy caches the cost of being wrong—putting a line into drowsy mode that is accessed soon thereafter—is relatively small. The only penalty one must contend with is an additional delay and energy cost for having to wake up a drowsy line. One of the simplest policies that one might consider is one where, periodically, all lines in the cache—regardless of access patterns—are put into drowsy mode and a line is woken up only when it is accessed again. This policy requires only a single global counter and no per-line statistics. Table 1 shows the working set characteristics of some of our workloads using a 2000 cycle update window, meaning that all cache lines are put into drowsy mode every 2000 cycles. Observations of cache activity are made over this same period. Based on this information we can estimate how effective this simpleminded policy could be.

The results show that on most of the benchmarks the working set—the fraction of unique cache lines accessed during an update window—is relatively small. On most benchmarks more than 90% of the lines can be in drowsy mode at any one time. This has the potential to significantly reduce the static power consumption of the cache. The downside of the approach is that the wake-up cost has to be amortized over a relatively small number of accesses: between 7 and 21, depending on the benchmark.

TABLE 1. Working set and reuse characteristics

	Level 1 data cache 32K, 4-way, 32byte line, window size = 2000 cycles						
	Working set	Number of accesses	Accesses per line	Accesses per cycle	Fraction of accesses same as in the n^{th} previous window		
					n=1	n=8	n=32
crafty	17.6%	1250.56	6.95	0.63	65.2%	54.9%	49.3%
vortex	10.8%	1209.07	10.89	0.60	54.3%	29.0%	31.0%
bzip	5.9%	1055.84	17.35	0.53	32.5%	19.7%	17.2%
vpr	9.2%	1438.69	15.27	0.72	62.2%	46.9%	45.6%
mcf	8.9%	1831.68	20.05	0.92	61.0%	60.8%	60.4%
parser	8.7%	971.73	10.85	0.49	46.9%	34.6%	28.4%
gcc	8.1%	809.69	9.78	0.40	36.9%	24.9%	21.1%
facerec	10.4%	970.04	9.15	0.49	37.4%	27.5%	33.6%
equake	7.0%	1513.27	21.09	0.76	92.8%	91.4%	90.7%
mesa	8.0%	1537.09	18.69	0.77	83.8%	76.8%	74.5%

$$ExecFactor = \frac{accs\left(\frac{wakelatency \times memimpact}{accsperline}\right) + (wsize - accs)}{wsize} \quad \text{(EQ 1)}$$

Equation 1 shows the formula for computing the expected worst-case execution time increase for the baseline algorithm. All variables except *memimpact* are directly from Table 1. The variable *accs* specifies the number of accesses, *wakelatency* the wakeup latency, *accsperline* the number of accesses per line, and *wsize* specifies the window size. Memimpact can be used to describe how much impact a single memory access has on overall performance. The simplifying assumption is that any increase in cache access latency translates directly into increased execution time, in which case memimpact is set to 1. Using this formula and assuming a 1 cycle wake-up latency, we get a maximum of 9% performance degradation for crafty and under 4% for equake. One can further refine the model by coming up with a more accurate value for memimpact. Its value is a function of both the microarchitecture and the workload:

- The workload determines the ratio of the number of memory accesses to instructions.
- The microarchitecture determines what fraction of wake-up transitions can be hidden, i.e., not translated into global performance degradation.
- The microarchitecture also has a significant bearing on IPC which in turn determines the number of memory accesses per cycle.

Assuming that half of the wake-up transition latencies can be hidden by the microarchitecture, and based on a ratio of 0.63 of memory accesses per cycle, the prediction for worst-case performance impact for the crafty benchmark reduces to 2.8%. Similarly, using the figure of 0.76 memory accesses per cycle and the same fraction of hidden wake-up transitions, we get a performance impact of about 1.4%. The actual impact of the baseline technique is likely to be significantly lower than the results from the analytical model, but nonetheless, these results show that there is no need to look for prediction techniques to control the drowsy cache; as long as the drowsy cache can transition between drowsy and awake modes relatively quickly, simple algorithms should suffice.

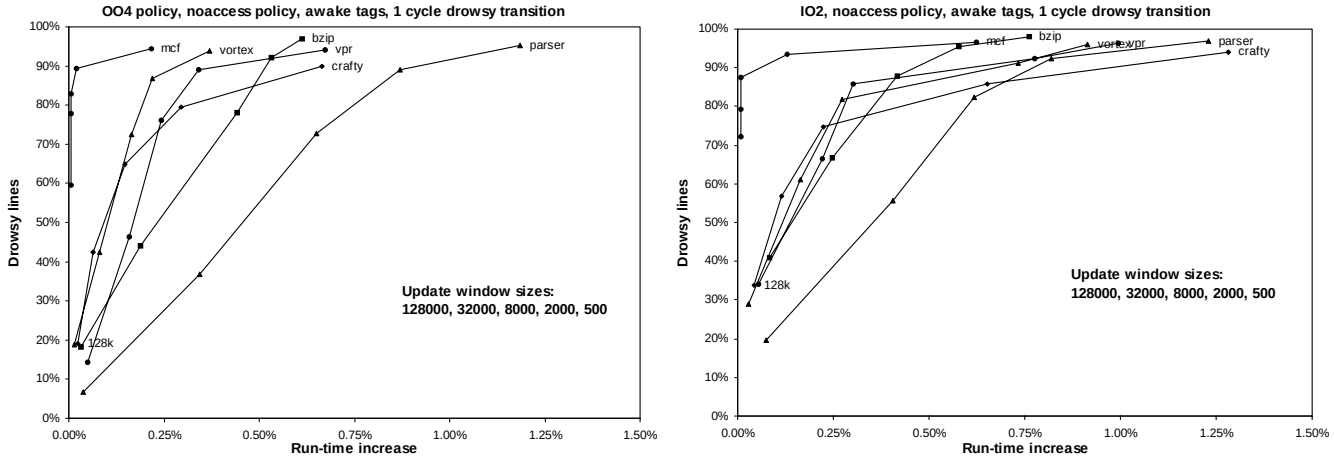
The right side of Table 1 contains information about how quickly the working set of the workloads are changing. The results in the table specify what fraction of references in a window are to lines that had been accessed 1, 8, or 32 windows before. This information can be used to gauge the applicability of control policies that predict the working set of applications based on past accesses. As can be seen, on many benchmarks (e.g. bzip, gcc), a significant fraction of lines are not accessed again in a successive drowsy window, which implies that past accesses are not always a good indication of future use. Aside from the equake and mesa benchmarks, where past accesses do correlate well with

TABLE 2.Latencies of accessing lines in the drowsy cache

		Awake	Drowsy
Awake Tags	Hit	1 cycle	1 cycle - wake up line 1 cycle - read/write line
	Miss	1 cycle - find line to replace memory latency	1 cycle - find line to replace memory latency Overlapped with memory latency: wake up line.

		Awake	Drowsy
Drowsy Tags	Hit	1 cycle	1 cycle - time for possible awake hit 1 cycle - wake up drowsy lines in set 1 cycle - read/write line Off-path: put unneeded lines in set back to drowsy mode
		All lines in set are awake	**Not all lines in set are awake**
	Miss	1cycle - find line to replace memory latency Off-path: put unneeded lines in set back to drowsy mode	1cycle - time for possible awake hit 1cycle - wake up drowsy lines in set 1cycle - find line to replace memory latency Off-path: put unneeded lines in set back to drowsy mode

FIGURE 3. Impact of window size on performance and on the fraction of drowsy lines



future accesses, most benchmarks only reaccess 40%-60% of the lines between windows. The implications of this observation are twofold: If an algorithm keeps track of which cache lines are accessed in a window, and only puts the ones into drowsy mode that have not been accessed in a certain number of past windows, then the number of awake to drowsy transitions per window can be reduced by about 50%. This in turn decreases the number of later wakeups, which reduces the impact on execution time. However, the impact on energy savings is negative since a larger fraction of lines are kept in full power mode, and in fact many of those lines will not be accessed for the next several windows, if at all.

Another important consideration is whether the tags are put into drowsy mode along with the data or whether they are always on. Table 2 shows the latencies associated with the different modes of operation. In both cases, no extra latencies are involved when an awake line is accessed. If tags are always on, then the cost of accessing a drowsy line is an additional cycle for waking it up first. Hits and misses are determined the same way as in normal caches. However, if tags—along with the data—can be drowsy, then the situation gets more complicated. Access hits to an awake cache line take a single cycle just as in the first case, however access hits to a drowsy line take a cycle longer. The reason for the extra delay is that during the first access cycle the cache is indexed, awake lines are read out and their tags are compared. If none of the awake tags match after the first read, then the controller wakes up all the drowsy lines in the indexed set and only then, an additional cycle later, can it read and compare the data. Thus, a drowsy access takes at least three cycles to complete. Another issue is that when drowsy lines are woken up in a set just so that their tags can be compared, they should be put back to sleep soon thereafter. It is not likely that these lines will be accessed soon, since in that case they would have been awake already. The controller can easily put these lines back into drowsy mode without having to wait for the sleep transition to complete.

Note that in direct-mapped caches there is no performance advantage to keeping the tags awake. There is only one possible line for each index, thus if that line is drowsy, it needs to be woken up immediately to be accessed.

3. Policy evaluation

In this section we evaluate the different policy configurations with respect to how they impact performance and the fraction of cache lines that are in drowsy mode during the execution of our benchmarks. All our algorithms work by periodically evaluating the contents of the cache and selectively putting lines into drowsy mode. The following parameters can be varied:

- *Update window size*: specifies in cycles how frequently decisions are made about which lines to put into drowsy mode.
- *Simple or Noaccess policy*: The policy that uses no per-line access history is referred to as the *simple* policy. In this case, all lines in the cache are put into drowsy mode periodically (the period is the window size). The *noaccess* policy means that only lines that have not been accessed in a window are put into drowsy mode.
- *Awake or drowsy tag*: specifies whether tags in the cache may be drowsy or not.
- *Transition time*: the number of cycles for waking up or putting to sleep cache lines. We only consider 1 or 2 cycle transition times, since our circuit simulations indicate that these are reasonable assumptions.

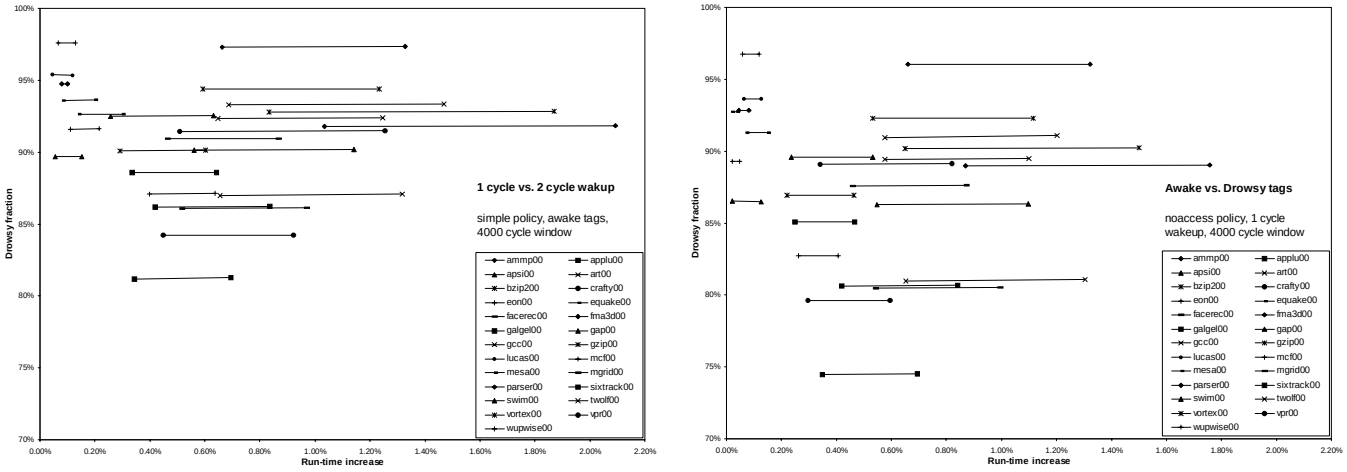
We use various benchmarks from the SPEC2000 suite on SimpleScalar using the Alpha instruction set to illustrate our points. Most of the results are shown using the out-of-order core in SimpleScalar. However, when appropriate we also show results for a simpler in-order core. The simulator configuration parameters are summarized below:

- *OO4*: 4-wide superscalar pipeline, 32K direct-mapped L1 icache, 32 byte line size - 1 cycle hit latency, 32K 4-way set associative L1 dcache, 32 byte line size - 1 cycle hit latency, 8 cycle L2 cache latency.
- *IO2*: 2-wide in-order pipeline, cache parameters same as for OO4.

All simulations were run for 1 billion instructions.

Figure 3 shows how window size impacts performance and the fraction of drowsy lines. For clarity, we are showing only a subset of the benchmarks. On an out-of-order core,

FIGURE 4. Impact of increased drowsy access latencies



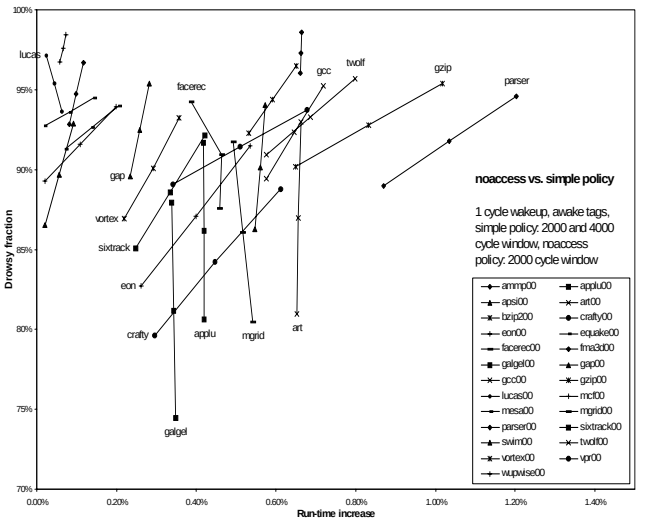
the sweetspot—where the energy-delay product is maximized—is around 2000 cycles. The same spot moves to between a window size of 4000 and 8000 cycles on the in-order core. The reason for this is that since the IPCs of the benchmarks using the IO2 model tend to be a little less than half as much as in the OO4 model, fewer memory accesses are made within the same amount of clock cycles. Since both the drowsy cache's run-time overhead and its effectiveness are correlated with the number of cache accesses within a window, comparable power-performance trade-off points are found at about twice the window size on the IO2 model as on OO4. Aside from the data for small window sizes, the two graphs look very similar.

The reason for the relatively small impact of the drowsy wake-up penalty on the in-order processor's performance is due to the non-blocking memory system, which can handle a number of outstanding loads and stores while continuing execution of independent instructions. Moreover, the drowsy wake-up penalty is usually only incurred with load instructions, since stores are put into a write buffer, which—if not full—allows execution to continue without having to wait for the completion of the store instruction.

The impact of increased transition latencies is shown in Figure 4. The top graph in the figure shows the impact of doubled wakeup latency using the *simple* policy, while the bottom graph shows the impact on the *noaccess* policy due to the use of drowsy tags. In both graphs, the two end points of a line represent the two different configurations of each benchmark. Both of the different types of overhead have similar impact on the given policy: the fraction of drowsy lines is unchanged, while the impact on run-time increases (the lines connecting the two points are horizontal and the points corresponding to the two cycle wakeup or the drowsy tags are always on the right). The run-time impact on the *simple* policy is larger compared to the *noaccess* policy, since a larger fraction of the cache is drowsy at any one time. Also note that for a given policy, the run-time overhead of using drowsy tags should be very similar to increasing the transition latency to two cycles. This is because both models increase the most common type of drowsy access—the drowsy hit—by the same amount.

Figure 5 contrasts the *noaccess* and the *simple* policies. The main question that we are trying to answer is whether there is a point to keeping any per-line statistics to guide drowsy decisions or if the indiscriminate approach is good enough. We show three different configurations for each benchmark on the graph: the *noaccess* policy with a 2000 cycle window and two configurations of the *simple* policy (4000 cycle and 2000 cycle windows). In all cases, the policy configurations follow each other from bottom to top in the aforementioned order. This means that in all cases, the *noaccess* policy has the smallest fraction of drowsy lines, which is to be expected, since it is conservative about which lines are put into drowsy mode. In all configurations, the performance impact is never more than 1.2% and the fraction of drowsy lines is never under 74%.

FIGURE 5. Comparison of the *noaccess* and *simple* policies



The bottom marker on each line corresponds to the *noaccess* policy with 2000 cycle window, the markers above it represent the *simple* policy with 4000 and 2000 cycle windows respectively

The benchmarks on the graph can be partitioned into two groups: ones on lines whose slopes are close to the vertical, and ones on lines that are more horizontal and thus have a smaller positive slope. All the benchmarks that are close to the vertical are floating point benchmarks and their orientation implies that there is very little or no performance benefit to using the *noaccess* policy or larger window sizes. In fact, the mgrid, galgel, applu, facerec, and lucas benchmarks have a slight negative slope, implying that not only would the simpler policy win on power savings, it would also win on performance. However, in all cases the performance difference is negligible and the potential power improvement is under 5%. The reason for this behavior is the very bad reuse characteristics of data accesses in these benchmarks. Thus keeping lines awake (i.e. *noaccess* policy, or larger window sizes) is unnecessary and even counterproductive.

This anomalous behavior is not replicated on the integer benchmarks, where in all cases the *noaccess* policy wins on performance but saves the least amount of power. Does this statement imply that if performance degradation is an issue then one should go with the more sophisticated *noaccess* policy? It does not. The slope between the upper two points on each line is almost always the same as the slope between the bottom two points, which implies that the rates of change between the datapoints of a benchmark are the same; the data point for the *noaccess* policy should be able to be matched by a different configuration of the *simple* policy. We ran experiments to verify this hypothesis and found that a window size of 8000 of the *simple* policy comes very close to the coordinates for the *noaccess* policy with a window size of 2000.

We find that the *simple* policy with a window size of 4000 cycles reaches a reasonable compromise between simplicity of implementation, power savings, and performance. The impact of this policy on leakage energy is evaluated in Section 5.

4. Circuit issues

Traditionally, two circuit techniques have been used to reduce leakage power in CMOS circuits: V_{DD}-gating and ABB-MTCMOS. Recently, both of these methods were applied to cache design as well [2][3][10]. In this paper, we instead propose the use of dynamic voltage scaling (DVS) for leakage control. While voltage scaling has seen extensive use for dynamic power reduction, short-channel effects also make it very effective for leakage reduction. Below, we discuss the traditional V_{DD}-gating and ABB-MTCMOS techniques for cache leakage reduction, as well as our proposed method using dynamic voltage scaling and a comparison between the different methods.

4.1 Gated-V_{DD}

The gated-V_{DD} structure was introduced in [1]. This technique reduces the leakage power by using a high threshold (high-V_t) transistor to turn off the power to the memory cell when the cell is set to low-power mode. This high-V_t device drastically reduces the leakage of the circuit because of the exponential dependence of leakage on V_t. While this method is very effective at reducing leakage, its main disadvantage lies in that it loses any information stored in the cell when switched into low-leakage mode. This means that a significant performance penalty is incurred when data in the cell is accessed and more complex and conservative cache policies must be employed.

4.2 ABB-MTCMOS

The ABB-MTCMOS scheme was presented in [5]. In this method, the threshold voltages of the transistors in the cell are dynamically increased when the cell is set to drowsy mode by raising the source to body voltage of the transistors in the circuit. Consequently, this higher V_t reduces the leakage current while allowing the memory cell to maintain its state even in drowsy mode. However, to avoid the need for a twin-well process, the dynamic V_t scaling is accomplished by increasing the source of the NMOS devices and by increasing the body voltage of the wells of the PMOS devices significantly when the circuit is in drowsy mode. While the leakage current through the memory cell is reduced significantly in this scheme, the supply voltage of the circuit is increased, thereby offsetting some of the gain in total leakage power.

Also, this leakage reduction technique requires that the voltage of the N-well and of the power and ground supply lines are changed each time the circuit enters or exits drowsy mode. Since the N-well capacitance of the PMOS devices is quite significant, this increases the energy required to switch the cache cell to high-power mode and can also significantly increase the time needed to transition to/from drowsy mode. Similarly to the gated-V_{DD} technique, ABB-MTCMOS also requires special high-V_t devices for the control logic.

4.3 Dynamic V_{DD} Scaling (DVS)

The method proposed in this paper utilizes dynamic voltage scaling (DVS) to reduce the leakage power of cache cells. By scaling the voltage of the cell to approximately 1.5 times V_t, the state of the memory cell can be maintained. For a typical 0.07um process, this drowsy voltage is conservatively set to 0.3V. Due to the short-channel effects in high-performance processes, the leakage current will reduce dramatically with voltage scaling. Since both voltage and current are reduced in DVS, a dramatic reduction in leakage power can be obtained. Since the capacitance of the power rail is significantly less than the capacitance of the N-wells, the transition between the two power states occurs more

FIGURE 6. Schematic of the drowsy memory circuit

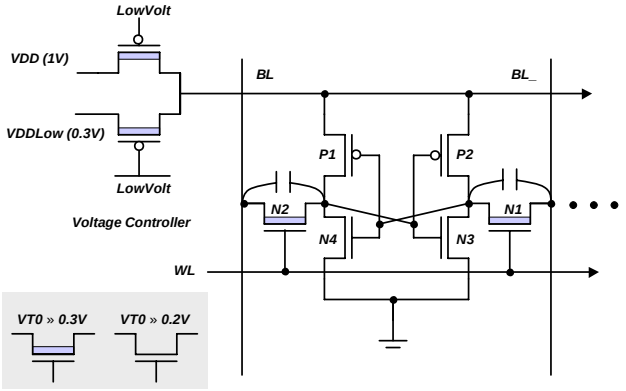
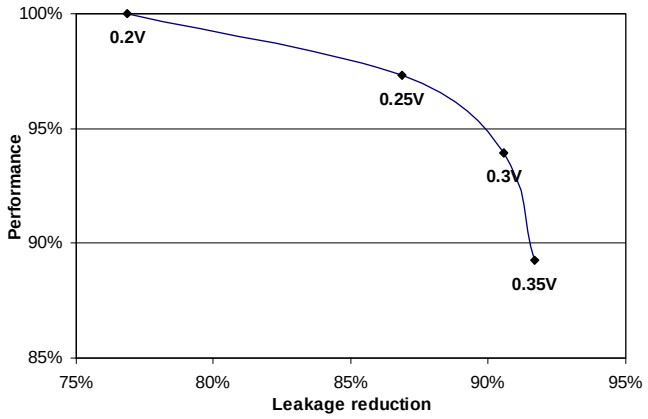
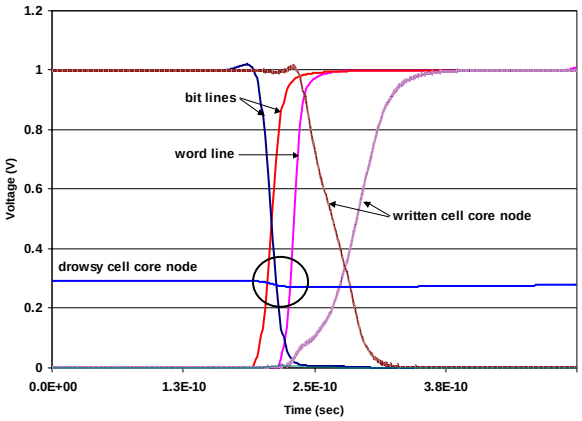


FIGURE 7. Leakage power reduction and performance



The figure shows various V_t numbers (next to the data points) and how these values impact performance and leakage reduction.

FIGURE 8. Cross-talk stability of the drowsy memory cell



Simulation of a write operation to the normal mode (awake) memory cell adjacent to a drowsy memory cell.

quickly in the DVS scheme than the ABB-MTCMOS scheme.

Figure 6 illustrates the circuit schematic of memory cells connected to the voltage-scaling controller. High-V_t devices are used as the pass transistors that connect the memory's internal inverters to the read/write lines (N1 and N2). This is necessary because the read/write lines are maintained in high-power mode and the leakage through the pass transistors would be too large otherwise. One PMOS pass gate switch supplies the normal supply voltage and the other supplies the low supply voltage for the drowsy cache line. Each pass gate is a high-V_t device to prevent leakage current from the normal supply to the low supply through the two PMOS pass gate transistors. A separate voltage controller is needed for each cache line.

In determining the high-V_t value for the access transistors of the memory cell (N1 and N2) we must consider the leakage power reduction as well as the performance impact of using a high-V_t device. Figure 7 shows leakage power reduction versus performance impact for various V_t values. Our HSPICE simulations using access transistors with 0.2V V_t show that the portion of leakage power caused by the pass transistors is 16.78% and 71.77% for normal and low supply voltage modes respectively. These values are quite significant. As we increase V_t of the access transistors towards 0.35V, the leakage power decreases exponentially and it approaches the maximum reduction ratio (92.15%) that can be achieved by eliminating leakage current entirely through the access transistor.

To estimate the performance degradation from increasing V_t, we measured the delay from the word line assertion to the point where there is a 50mV voltage difference between two complementary bit lines using a netlist with extract capacitances. This voltage difference is the threshold for sense-amp activation. The value of 50mV was obtained by conservatively scaling the value used in the former memory design technology [11]. Clearly the delay is increased as the V_t is increased as shown in Figure 7, but the fraction of the delay from the word line activation to the sense-amp activation is only about 22% among other delay factors that contribute to the total access time of the memory system, according to our CACTI calculations. We chose 0.3V for the high-V_t value because it results in a sensible trade-off point between performance loss (6.05%) and leakage reduction (91.98%). However, we can compensate for the performance loss by carefully tuning the size of the access and cross-coupled inverter transistors. We have not done so in this study because it may increase the dynamic power dissipation as a result of the increase in switching capacitance of the bit lines.

A possible disadvantage of the circuit in Figure 6 is that it has increased susceptibility to noise and the variation of V_t across process corners. The first problem may be corrected with careful layout because the capacitive coupling of the lines is small. To examine the stability of a memory cell in the low power mode, we simulated a write operation to an adjacent memory cell that shares the same bit lines but whose supply voltage was normal. The coupling capacitance and the large voltage swing across the bit lines would make the bit in the drowsy memory cell vulnerable to flipping if this circuit had a stability problem. However, our experiments show (Figure 8) that the state of the drowsy memory cell is stable. There is just a slight fluctuation in the core node voltage caused by the signal cross-talk between the bit lines and the memory internal nodes. In addition, there is no cross-talk noise between the word line and the internal node voltage, because word line gating prevents accesses to drowsy mode memory cells. Of course, this voltage scaling technique has less immunity against a single event upset (SEU) from alpha particles [12], but this problem can be relieved by process techniques such as silicon on insulator (SOI). Other static memory structures also suffer from this problem. The second problem, variation of V_t, may be handled by choosing a conservative V_{DD} value, as we have done in our design.

The memory cell layout was done in TSMC 0.18um technology, which is the smallest feature size available to the academic community. The dimensions of our memory cell is 1.84um by 3.66um, and those for the voltage controller are 6.18um by 3.66um. We estimate the area overhead of the voltage controller is equivalent to 3.35 memory cells for

TABLE 3. Comparison of various low-leakage circuit techniques

	Advantages	Disadvantages	Leakage power in low power mode
DVS	• Retains cell information in low-power mode. • Fast switching between power modes. • Easy implementation. • More power reduction than ABB-MTCMOS.	• Process variation dependent. • More SEU noise susceptible.	6.24nW
ABB-MTCMOS	• Retains cell information in low-power mode.	• Higher leakage power. • Slower switching between power modes.	13.20nW
Gated-V_{DD}	• Largest power reduction. • Fast switching between power modes. • Easy implementation.	• Loses cell information in low-power mode.	0.02nW

a 64 x Leff (effective gate length) voltage controller. This relatively low area overhead can be achieved because the routing in the voltage controller is simple compared to the memory cell. In addition, we assumed the following (conservative) area overhead factors: 1) 1.5 equivalent memory cells for the drowsy bit (the 0.5 factor arises from the two additional transistors for set and reset); 2) 1 equivalent memory cell for the control signal driver (two inverters); and 3) 1.5 equivalent memory cells for the wordline gating circuit (a nand gate). The total overhead is thus equivalent to 7.35 memory cells per cache line. The total area overhead is less than 3% for the entire cache line. To examine the effects of circuit issues like stability and leakage power reduction, we applied a linear scaling technique to all the extracted capacitances.

In Table 3, we list the advantages and disadvantages for the two traditional circuit techniques for leakage reduction as well as for DVS, and we show the power consumption for the three schemes in both normal and low power mode. The leakage power in the gated-V_{DD} method is very small compared to the other schemes, however, this technique does not preserve the state of the cache cell. Comparing the DVS and ABB-MTCMOS techniques, the DVS method reduces leakage power by a factor of 12.5, while the ABB-MTCMOS method reduces leakage by only a factor of 5.9.

Detailed power values for drowsy mode and normal-power mode for the proposed method are shown in Table 4. In order to determine the time required to switch a cache line from drowsy mode to normal power mode, we measured the delay time of the supply lines with HSPICE and the Berkeley Predictive Model [4] for a 0.07um process. To measure the transition delay, we connected a 32KB memory cell array to the supply voltage controllers and then estimated the capacitances of the supply voltage metal line and bit lines. The transition delay varies depending on the transistor width of the pass gate switch in the voltage controller. A 16 x Leff PMOS pass-transistor is needed for a two cycle transition delay. A single cycle transition delay can be obtained by increasing the width of this transistor to 64 x Leff. The cycle time of the cache was estimated using the CACTI model with the supported process scaling. We found that the access time of the cache is 0.57ns and that the transition time to and from drowsy mode is 0.28ns with a 64 x Leff width PMOS pass-transistor in the normal mode voltage supplier.

5. Energy consumption

Table 5 compares the energy savings due to the *simple* policy with a window size of 4000 cycles and with and without the use of drowsy tags. Normalized total energy is the ratio of total energy used in the drowsy cache divided by the total energy consumed in a regular cache. Similarly, normalized leakage energy is the ratio of leakage energy in the drowsy cache to leakage energy in a normal cache. The data in the DVS columns correspond to the energy savings resulting from the scaled-V_{DD} (DVS) circuit technique while the theoretical minimum column assumes that leakage in low-power mode can be reduced to zero (without losing state). The theoretical minimum column estimates the energy savings given the best possible hypothetical circuit technique. For all the results in the table, we conservatively assume that there are only 19 tag bits (corresponding to 32 bit addressing) per line, which translates into 6.9% of the bits on a cache line.

The table shows that our implementation of a drowsy cache can reduce the total energy consumed in the data cache by more than 50% without significantly impacting performance. Total leakage energy is reduced by an average of 71% when tags are always awake and by an average of 76% using the drowsy tag scheme. Leakage energy could potentially be cut in half if the efficiency of the drowsy cir-

TABLE 4. Energy parameters and drowsy transition time for 32-KB 4-way set associative cache with 32Wmin for voltage controller switch size

Dynamic energy per access	Leakage energy per bit	Drowsy leakage energy per bit	Transition energy (W=64Leff)	Drowsy transition latency
2.94E-10J	1.63E-15J	2.59E-16J	2.56E-11J	1 cycle

TABLE 5. Normalized energy results and run-time increase for the OO4 core, *simple* policy, 1 cycle drowsy transition, 4000 cycle window for both awake and drowsy tags

	Awake tags					Drowsy tags				
	Normalized total energy		Normalized leakage energy		Run-time	Normalized total energy		Normalized leakage energy		Run-time
	DVS	Theoretical min.	DVS	Theoretical min.	increase	DVS	Theoretical min.	DVS	Theoretical min.	increase
ammp	0.25	0.11	0.24	0.09	0.66%	0.20	0.05	0.18	0.03	1.33%
applu	0.47	0.36	0.33	0.20	0.42%	0.43	0.32	0.28	0.14	0.84%
apsi	0.46	0.36	0.30	0.17	0.06%	0.43	0.32	0.25	0.11	0.22%
art	0.39	0.27	0.32	0.19	0.66%	0.35	0.22	0.27	0.13	1.32%
bzip2	0.45	0.34	0.26	0.12	0.59%	0.41	0.30	0.21	0.06	1.23%
crafty	0.53	0.44	0.34	0.22	0.45%	0.50	0.41	0.29	0.16	0.92%
eon	0.55	0.47	0.32	0.19	0.40%	0.52	0.43	0.27	0.13	0.64%
equake	0.50	0.41	0.27	0.13	0.08%	0.47	0.37	0.21	0.06	0.20%
facerec	0.46	0.36	0.29	0.15	0.46%	0.42	0.31	0.24	0.09	0.87%
fma3d	0.41	0.30	0.26	0.12	0.10%	0.37	0.25	0.20	0.05	0.08%
galgel	0.52	0.42	0.37	0.25	0.34%	0.49	0.39	0.32	0.19	0.69%
gap	0.52	0.43	0.28	0.14	0.26%	0.49	0.40	0.22	0.08	0.63%
gcc	0.43	0.32	0.28	0.14	0.65%	0.39	0.27	0.23	0.08	1.25%
gzip	0.46	0.36	0.28	0.14	0.83%	0.43	0.32	0.22	0.07	1.87%
lucas	0.43	0.32	0.25	0.11	0.04%	0.39	0.27	0.20	0.05	0.12%
mcf	0.55	0.46	0.28	0.15	0.11%	0.52	0.42	0.23	0.08	0.22%
mesa	0.51	0.42	0.27	0.14	0.14%	0.48	0.38	0.22	0.07	0.30%
mgrid	0.47	0.36	0.33	0.20	0.52%	0.43	0.32	0.28	0.14	0.97%
parser	0.46	0.35	0.28	0.15	1.03%	0.42	0.31	0.23	0.08	2.09%
sixtrack	0.48	0.38	0.31	0.18	0.33%	0.44	0.34	0.26	0.11	0.65%
swim	0.40	0.28	0.30	0.16	0.56%	0.36	0.24	0.24	0.10	1.14%
twolf	0.40	0.28	0.27	0.13	0.69%	0.35	0.23	0.22	0.07	1.48%
vortex	0.49	0.39	0.30	0.16	0.29%	0.46	0.35	0.24	0.10	0.60%
vpr	0.51	0.42	0.29	0.15	0.51%	0.48	0.38	0.23	0.09	1.25%
wupwise	0.36	0.24	0.24	0.09	0.07%	0.31	0.18	0.18	0.02	0.13%
Average	0.46	0.35	0.29	0.15	0.41%	0.42	0.31	0.24	0.09	0.84%

cuit techniques are improved (see the theoretical minimum column). However, the benefits of any further reduction of leakage in drowsy mode are tempered by the fact that lines must still spend time in full-power mode when being accessed, and consequently their leakage power consumption is at least an order of magnitude higher. Thus, the impact of more efficient drowsy circuits on total energy savings is reduced. Nonetheless, if there are circuits that are more efficient without significantly increasing the transition time, the *simple* policy can take advantage of it.

An important question is whether it is worth using the more complex drowsy tag scheme. The energy-delay product of the benchmarks when using drowsy tags are always lower than without. So the decision about whether to use drowsy tags or not comes down to acceptable engineering trade-offs. Also note that in direct-mapped caches, tags can be put into drowsy mode without undue performance impact or implementation complexity (see Section 2). However, using a direct mapped data cache instead of the 4-way associative one has its own costs: miss rates on our benchmarks are approximately tripled, which—aside from the performance penalty—can significantly impact the total energy consumption of the processor.

6. Conclusions and future work

During our investigations of drowsy caches we found that our simplest policy—where cachelines are periodically put into a low-power mode without regard to their access histories—can reduce the cache's static power consumption by more than 80%. The fact that we do not propose a more sophisticated policy with better characteristics should not be taken as proof that such policy does not exist. However, we believe that our combination of a simple circuit technique with a simple microarchitectural mechanism provides sufficient static power savings at a modest performance impact, that it makes sophistication a poor trade-off. The *simple* policy is not a solution to all caches in the processor. In particular, the L1 instruction cache does not do as well with the simple algorithm and only slightly better with the *noaccess* policy. We are investigating the use of instruction prefetch algorithms combined with the drowsy circuit technique for reducing leakage power in the instruction caches. Our ongoing work also includes the extension of our techniques to other memory structures, such as branch predictors.

An open question remains as to the role of adaptivity in determining the window size. We found that for a given machine configuration, a single static window size (2000 to 8000 cycles, depending on configuration) performs adequately on all of our benchmarks. However, the optimum varies slightly for each workload, thus making the window size adaptive would allow a finer power-performance trade-off. One way of accomplishing this is by monitoring the microarchitecture and counting the number of stall cycles that can be attributed to the drowsy wakeup latency, and only putting the cachelines into drowsy mode again after their previous wakeup overheads have been amortized (i.e. the performance impact falls under a given threshold). The user's preference for trading-off performance and power savings could be controlled by the setting of the threshold value.

While our dynamic voltage scaling circuit does not offer the lowest leakage per bit, the fact that it maintains its stored value, allows for an aggressive algorithm when deciding which cache lines to put into drowsy mode. Using our policy and circuit technique, the total energy consumed in the cache can be reduced by an average of 54%. This compares well with the theoretical maximum reduction of 65% corresponding to a hypothetical circuit that consumes no leakage energy in drowsy mode. Since the amount of leakage energy consumed in drowsy mode is only an average of 24% of total, further reductions of this fraction will yield only diminished returns.

Moreover, as the fraction of leakage energy is reduced from an average of 76% in projected conventional caches to an average of 50% in the drowsy cache, dynamic energy once again becomes a prime candidate for reduction.

7. Acknowledgements

This work was supported by DARPA/AFRL F33615-00-C-1678, DARPA MDA972-33-1-0010, MACO 98-DF-660, and SRC-2001-HJ-904.

References

[1] M. Powell, et. al. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. *Proc. of Int. Symp. Low Power Electronics and Design*, 2000, pp. 90-95.

[2] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. *Proc. of Int. Symp. Computer Architecture*, 2001, pp. 240-251.

[3] H. Zhou, et. al. Adaptive mode-control: A static-power-efficient cache design. *Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques*, 2001, pp. 61-70.

[4] http://www-device.eecs.berkeley.edu

[5] K. Nii, et. al. A low power SRAM using auto-backgate-controlled MT-CMOS. *Proc. of Int. Symp. Low Power Electronics and Design*, 1998, pp. 293-298.

[6] M. Weiser, et. al. Scheduling for reduced CPU energy. *Proc. of the First Symp. of Operating Systems Design and Implementation*. November 1994

[7] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. *Proceedings of International Symposium on Low Power Electronics and Design*, June, 1998, pp. 76-81.

[8] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. *Proc. of Int. Conf. on Mobile Computing and Networking (MOBICOM-7)*, July 2001, pp. 260-271.

[9] S. Wolf. Silicon processing for the VLSI era Volume 3 - The submicron MOSFET. *Lattice Press*, 1995, pp. 213-222.

[10] H. Hanson, et al. Static energy reduction techniques for microprocessor caches. *Proc. of the Int. Conf. Computer Design*, 2001.

[11] K. Itoh. VLSI memory chip design. *Springer Publisher*, 2001, pp. 413-423.

[12] T. May and M. Woods. Alpha-particled-induced soft errors in dynamic memories. *IEEE Trans. on Electron Devices*, Vol. ED-26, No. 1, Jan. 1979.