Drowsy Instruction Caches

Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction

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

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

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

Abstract

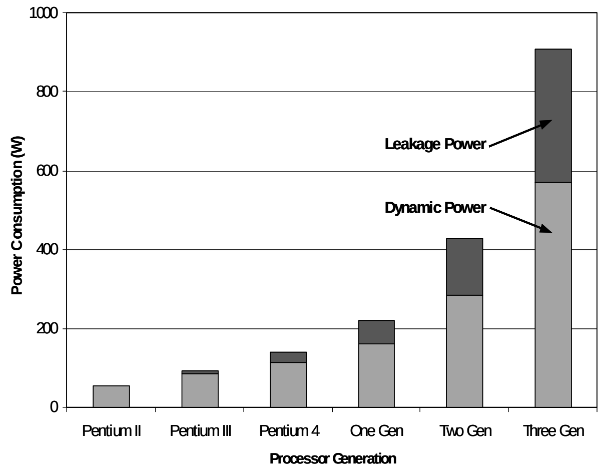
On-chip caches represent a sizeable fraction of the total power consumption of microprocessors. Although large caches can significantly improve performance, they have the potential to increase power consumption. As feature sizes shrink, the dominant component of this power loss will be leakage. In our previous work we have shown how the drowsy circuit—a simple, state-preserving, low-leakage circuit that relies on voltage scaling for leakage reduction—can be used to reduce the total energy consumption of data caches by more than 50%. In this paper, we extend the architectural control mechanism of the drowsy cache to reduce leakage power consumption of instruction caches without significant impact on execution time. Our results show that data and instruction caches require different control strategies for efficient execution. To enable drowsy instruction caches, we propose a technique called cache sub-bank prediction which is used to selectively wake up only the necessary parts of the instruction cache, while allowing most of the cache to stay in a low leakage drowsy mode. This prediction technique reduces the negative performance impact by 76% compared to the no-prediction policy. Our technique works well even with small predictor sizes and enables an 86% reduction of leakage energy in a 64K byte instruction cache.

1. Introduction

Leakage power is a difficult issue confronting processor designers. On one hand, performance demands require the use of fast transistors that consume ("leak") energy even when they are turned off, on the other hand, new applications and cost issues favor designs that are energy efficient. Figure 1 illustrates the magnitude of the problem with data from some existing processors and projections based on future transistor counts and circuit parameters. As can be seen, even in current processes, leakage power consumption can be as much as 20% of total and this fraction will increase significantly in the future. We believe that in three generations, leakage power will amount to as much as 30%-40% of total power consumed by the processor. Moreover, total power will be a severely limiting factor of integration levels and complexity, since it is unlikely that the amount of projected power (close to 900W) would fit in most people's power budget.

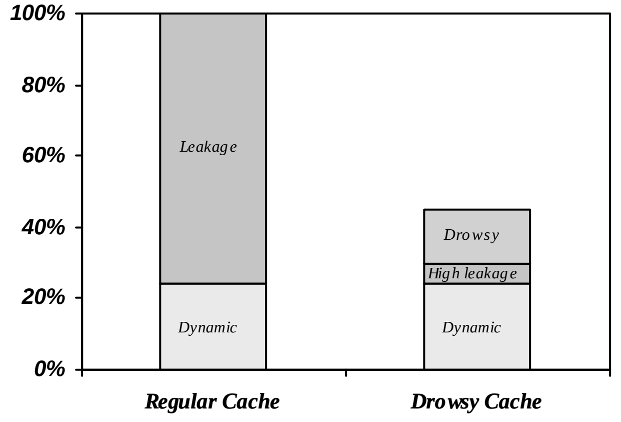
Leakage is a problem for all transistors, but it's a particularly important problem in caches, where leakage power can be the dominant fraction of total power consumption. We project that in a 0.07 micron process, leakage will amount to more than 70% of power consumed in caches if left unchecked. Most data in a cache are accessed relatively infrequently, thus as the cost of storing data increases in the form of leakage power, the contribution of dynamic power consumption diminishes. To alleviate this problem, transistors in caches could be statically designed such that they have less leakage, for example by assigning them a higher threshold voltage. However, this trade-off ultimately implies slower caches. Architects would like to have the best of all worlds: large caches, fast access

FIGURE 1. Dynamic and static power trends



Data based on published Intel values and projections based on the Int'l Technology Roadmap for Semiconductors.

FIGURE 2. Energy reduction using the drowsy cache



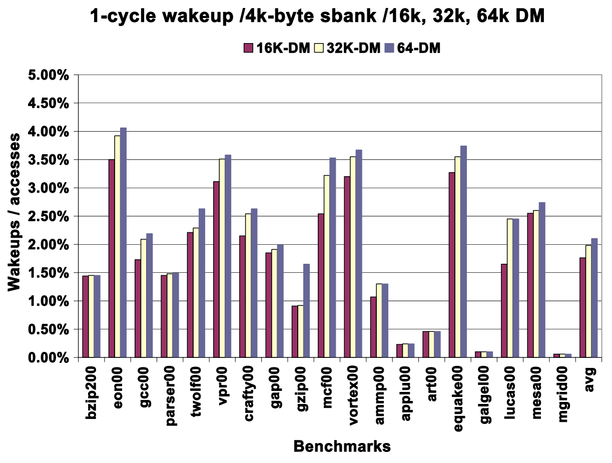
times, and low power consumption. We believe that it is possible to reconcile these aims by taking advantage of the run-time characteristics of workloads and by approaching the problem from both the circuit and microarchitecture directions simultaneously.

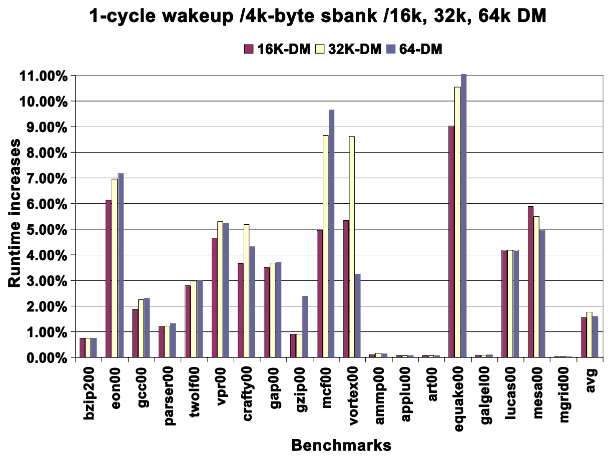
The main idea behind the drowsy cache is that cachelines can be in one of two modes: in a low-leakage drowsy mode, in which data is preserved but cannot be accessed, and in a high-leakage awake mode, which is used during accesses. To reduce leakage power consumption, an algorithm is used to decide which lines will be accessed in the near future and these are kept in the awake state, and the rest of the lines are put into the low power drowsy mode. Figure 2 illustrates how the different energy components vary between regular and drowsy caches. While in a regular cache all lines leak at a high rate, in the drowsy cache the high leakage component is only incurred when the line is in awake mode and is predicted to be accessed. Although leakage is not zero in drowsy mode, it provides a 6x to 10x reduction (depending on design) over the regular high-leakage mode. The drowsy circuit technique, which uses dynamic voltage scaling for leakage reduction is described in [1]. Voltage scaling yields significant leakage power reduction, due to short-channel effects in deep-submicron processes [2]. The combined effect of reduced leakage current and reduced voltage yields a significant reduction in leakage power. While voltage scaling does not reduce leakage as much as gated-V$_{dd}$ [3][4], it has the crucial advantage of being able to preserve the state of the transistors.

The circuit technique used in this paper is an improvement to the one described in our previous work [1]. In addition, we are proposing a new microarchitectural control technique for making drowsy instruction caches—as opposed to data caches that were the focus of our previous investigations. We found that while our previous algorithm was very effective for data caches, it does not work well for instruction caches, due to the different locality characteristics.

Various researchers have proposed the use of sub-banks as a means of reducing power consumption in caches. In [5], the instruction cache is partitioned into several sub-banks, and on each access only a limited set of sub-banks are checked for their contents. This approach reduces the dynamic power consumption at the cost of slightly increasing the cache access time due to additional decoder logic for indexing the sub-banks. In [6], a leakage power reduction circuit technique is applied to the sub-bank that has been most recently accessed. The circuit technique optimizes leakage power consumption by biasing the bit-lines based on the number of ones and zeros connected to each bit-line. The weakness of this technique is that it requires the processor to wake up the next target sub-bank on the critical path, where the penalty for the wake-up can be several cycles. According to our experiments the use of techniques employed in [6] can

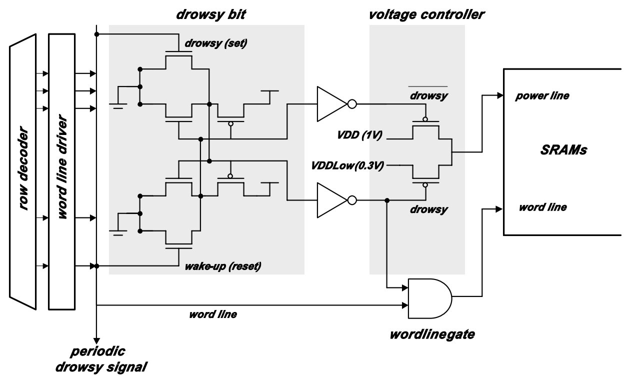
FIGURE 3. Wake-up overhead and run-time increase





For the simulations, 16K-byte, 32K-byte, and 64K-byte direct mapped instruction caches are used, and the sub-bank size is 4K bytes for all the cache sizes (see Section 5.1 for the detailed simulation machine configuration).

FIGURE 4. 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.

result in a run-time increase of 4.06% to 12.46% on SPEC 2000 benchmarks, even when assuming an aggressive singe cycle wake-up penalty (see Figure 3).

To minimize the performance impact as well as leakage power consumption, in this paper we propose a low leakage instruction cache architecture using our drowsy circuit technique combined with cache sub-banking and various sub-bank prediction techniques. Our prediction techniques rely on the insight that transitions between sub-banks are often correlated with specific types of instructions. Due to program loops, the program counter, which is the instruction cache access index, remains in small cache regions for relatively long periods of time. On the other hand, there are often abrupt changes in the accessed cache region when subroutines are called, or when subroutine returns, and long distance unconditional branches are executed. Most conditional branches stay within the current cache region and it is rare that the these branches jump across page boundaries.

In this paper we focus mainly on the implication of the various sub-bank prediction techniques for the instruction caches since other low leakage circuit techniques can be deployed instead of our drowsy circuit technique. Section 2 reviews the drowsy cache technique for data caches and examines the behavior of the policy for data caches on instruction caches. Section 3 presents low leakage power instruction cache architectures using the drowsy circuit and sub-bank prediction techniques. Section 4 discusses detailed circuit issues for the proposed drowsy circuit technique and cache sub-bank design. Section 5 shows simulation models and experiment results for the proposed architectures. Section 6 concludes this work and suggests directions for future research.

2. Drowsy techniques for caches

2.1 Operation of drowsy cache line

Figure 4 shows a cache line that supports a drowsy mode proposed in [1]. In order to support the drowsy mode, each 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 cache line voltage between normal (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 causing the supply voltage to be switched to normal Vdd. The word-line gating circuit is used to prevent accesses of the drowsy memory cells in the cache line, since the supply voltage of the drowsy cache line is far lower than the precharged bit-line voltage and thus unchecked accesses to a drowsy line could destroy its contents. Whenever a cache line is accessed, the cache controller monitors the voltage level of the cache line by checking the drowsy bit. If the accessed line is in normal mode, its contents can be read 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 discharging 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.

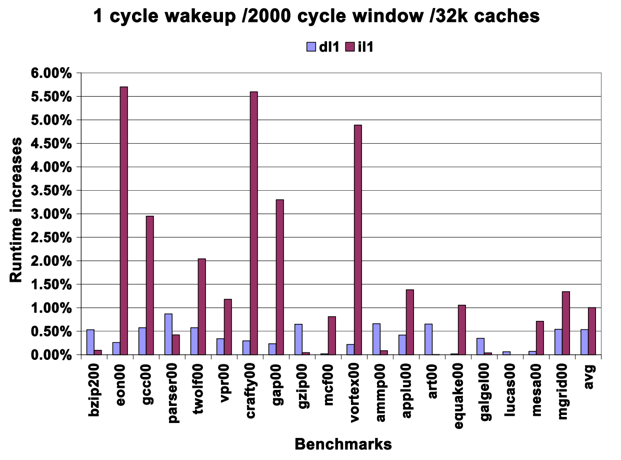
2.2 Drowsy Policy

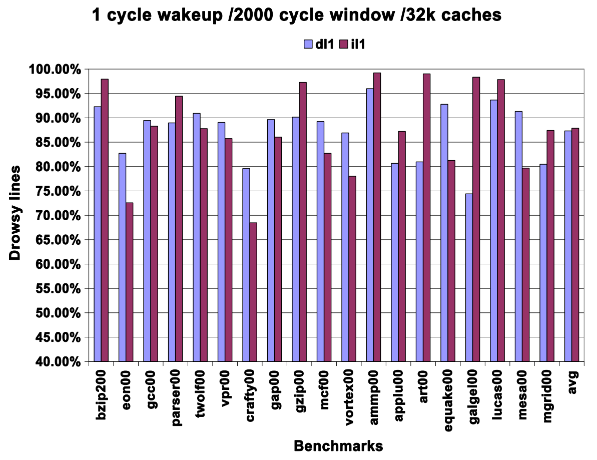
The key difference between drowsy caches and the caches that use gated-Vdd is that in drowsy caches the cost of being wrong—putting a line into drowsy mode that is accessed soon thereafter—is relatively small. The only penalty one must contend with is an additional delay and energy cost for having to 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. 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) [1].

2.3 Drowsy Policy Evaluation

In this section, we evaluate the *simple* policy for data and instruction caches, and show that this policy, which was designed for data caches, is not as effective for instruction caches. Figure 5 shows the run-time increases and the percentage of the drowsy lines—which is proportional to leakage power reduction—of workloads using a 2000-cycle update window, meaning that all cache lines are put into drowsy mode every 2000 cycles. According to the experimental results shown in Figure 5, using the simple policy on 32K byte direct-mapped instruction cache may have a run-time impact of as much 5.7%, and the percentage of drowsy lines can be as low as 68.5%. This is in sharp contrast with the simple policy on a 32K byte 4-way set-associative data cache, where on the same benchmarks the run-time impact is no more than 0.87% and the frac-

FIGURE 5. Run-time increase and drowsy line percentage using the simple policy





For the simulations, 32K-byte direct mapped instruction and 4-way data caches are used, and 2000-cycle window size and 1-cycle wake-up latency is used for the drowsy policy. (see Section 5.1 for the detailed simulation configuration).

tion of drowsy lines is no lower than 74.4%. These experimental results show that the application of the drowsy technique for the instruction cache can result in both poor performance and relatively low leakage reduction compared to the data cache. The main reason for this behavior is that data caches tend to have better temporal locality while instruction caches tend to have better spatial locality.

3. Drowsy caches using memory bank predictions

3.1 The drowsy circuit and memory sub-banking techniques

Figure 6 illustrates a 16K byte direct-mapped cache architecture using a technique based on voltage scaling and four 4K byte sub-banks. The pre-decoder identifies which sub-bank is accessed with a cache access address, and the decoder in each sub-bank selects an appropriate cache line in the sub-bank with the pre-decoded address. In this technique, the pre-decoder includes the wake-up logic which drives the wake-up signal to the target sub-bank. Only one sub-bank is active at a time, while the rest of the sub-banks are in drowsy mode by scaling the voltage levels of all lines in the sub-bank. Whenever the processor accesses a cache line in a non-active sub-bank, the pre-decoder activates the next target sub-bank, and puts the currently active sub-bank into drowsy mode. During the wake-up of the next target sub-bank, the processor halts since it must wait until reinstating the power supply lines of the target sub-bank to the normal voltage level. On a hit, this wake-up latency is incurred on the critical path. On a cache miss, the wake-up latency can be hidden during the miss handling cycles. Therefore, to avoid undue performance degradation on a cache hit, it is critical to wake-up the next sub-bank as soon as possible.

Each cache line in the drowsy instruction cache consists of a cache line which is illustrated in Figure 7.

FIGURE 6. Cache architecture using memory sub-banking and voltage scaling techniques.

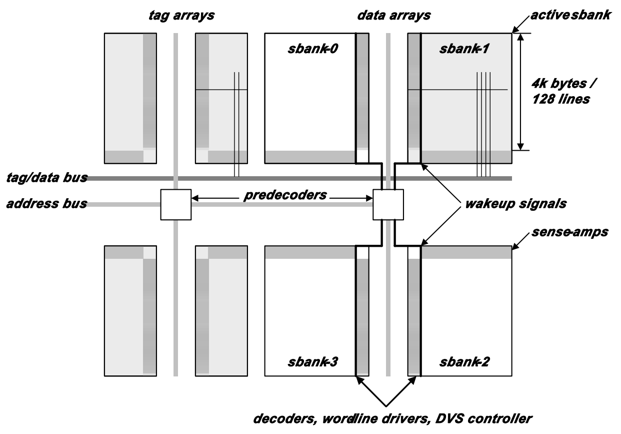


FIGURE 7. Implementation of the drowsy cache line

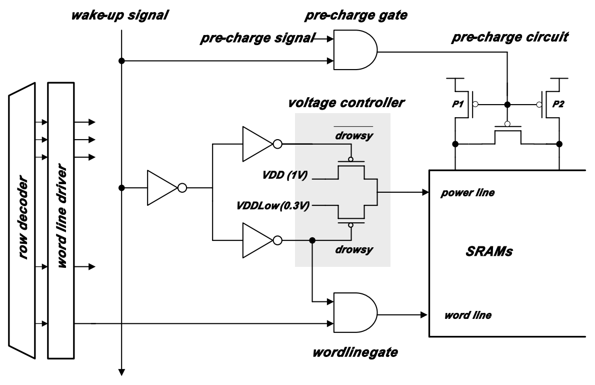
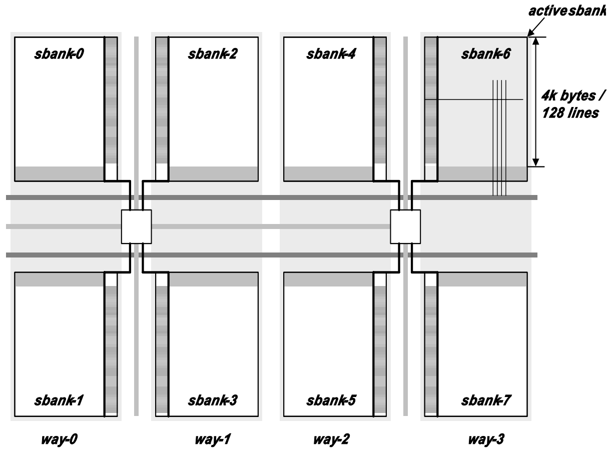
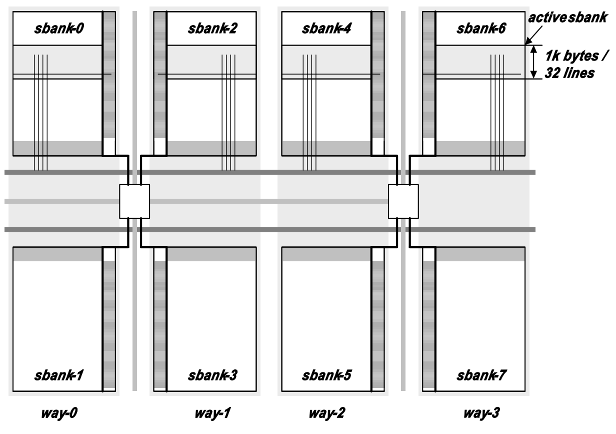


FIGURE 8. **Vertical and horizontal configurations for sub-bank cache partitioning.**



Vertical configuration.



Horizontal configuration.

Unlike in the drowsy data cache, illustrated in Figure 4, we do not need a drowsy bit for each line. Instead, the wake-up logic in the pre-decoder sends wake-up signals to the target sub-bank. Furthermore, the pre-charge circuit is modified to reduce the leakage current through the wordline pass transistors in the conventional 6T memory cell by gating the pre-charge signal with the wake-up signal. With this pre-charge gating technique, we do not need to use high-Vt pass transistors to reduce the leakage power via the pass transistors, which improves access time of the sub-banks compared to our previous scheme [1]. More detail of the circuit technique is provided in Section 4.

A cache can be sub-banked in two different ways: vertically or horizontally. Assuming a 32K byte 4-way associative cache, using the vertical organization, one would assign 2 4K byte sub-banks in each way, and activate only one sub-bank among 8 sub-banks as shown in the vertical configuration of Figure 8 (the sub-bank 6 of the way-3 is active, and the rest of the sub-banks are in drowsy mode). In a horizontal organization, sub-banks are distributed through 4 ways. In this case, we can assign 1K for each way, and activate each 1K portion of the sub-bank in all 4 ways as illustrated on the right side of Figure 8. The horizontal scheme also requires a separate modified precharge circuit shown in Figure 7 for each 1K portion of the 4K sub-bank. In both cases, the same amount of cache area is activated.

In the vertical configuration, the change of either the sub-bank address or the way address causes performance loss since it is likely that the processor is looking for data that is contained in a currently inactive way of the cache. However, this scheme has the advantage of lower dynamic power consumption, since only one way is accessed at a time [5][7][8][9][10]. Because it is simpler of the two, we limit our study to the vertical organization.

3.2 Memory sub-bank prediction buffers

Without any prediction for the next target sub-bank, performance is degraded significantly by the wake-up penalties as shown in Figure 3. According to our analysis, most transitions among the sub-banks are caused by subroutine calls, returns and long distance unconditional branches. These transition points from one sub-bank to another tend to repeat. If the instructions that cause sub-bank transitions could be marked in the cache, this information could be used to hide the wake-up.

Figure 9 illustrates the next sub-bank prediction buffer scheme for a 16K byte direct mapped cache. The assumption in the figure is that there is a single cycle wake-up latency and both code regions are already in the cache. Each prediction buffer entry contains an instruction address which is the address of the instruction one before the instruction (usually a branch) which leads to another sub-bank. The buffer entry also contains the next target sub-bank index and a valid bit. On each cache access, this buffer is consulted to

FIGURE 9. **Next sub-bank prediction buffer**

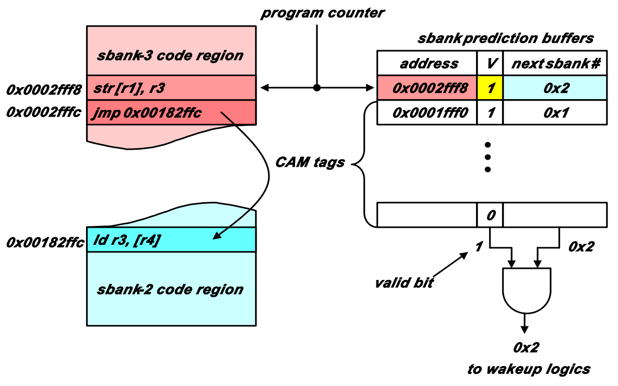
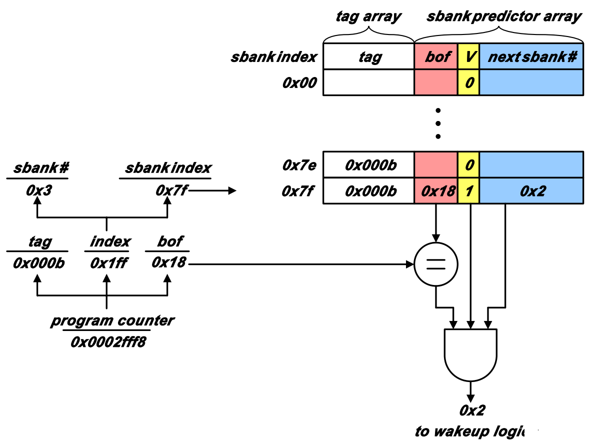


FIGURE 10. Next sub-bank predictors in cache tags



see whether or not a new sub-bank is predicted to be awakened. If there is a misprediction, or no prediction at all, the old entry is updated or a new one allocated. In the example of Figure 9 the control flow is predicted to jump to the sub-bank 2 code region.

It is important to predict the next target sub-bank address one instruction ahead in order to avoid losing performance due to the one cycle wake-up overhead. For the vertical sub-bank configuration of a set associative cache, the sub-bank index field also contains the target way information along with the sub-bank number.

3.3 Next sub-bank predictors in cache tags

The area as well as power overhead of the prediction buffer can be significant using the sub-bank prediction technique shown in Section 3.2. In particular, the CAM tag in the prediction buffers can consume significant amounts of dynamic power. In this section, a sub-bank prediction technique is presented which extends cache tags to minimize the power cost of prediction.

Figure 10 illustrates the extended cache tags to support the next sub-bank prediction. Each tag entry contains a block address of the transition instruction (bof field in Figure 10), the next sub-bank address, and a valid bit. Whenever the processor accesses the cache, it compares the block address of the current instruction and checks the validity of the prediction information. If the address matches and the information is valid, the processor sends the predicted next sub-bank address to the wake-up logic.

The disadvantage of this technique is that prediction information is lost if the cache line is replaced. Furthermore, multiple next sub-bank addresses cannot be kept in a cache tag when there are multiple transition addresses in a cache line. However, our experiments show that this situation arises relatively infrequently.

4. Circuit issues

Traditionally, three circuit techniques have been used to reduce leakage power in CMOS circuits: gated-Vdd, ABB-MTCMOS (Adaptive Body Biasing - Multi Threshold CMOS) and leakage-biased bitlines. Recently, these methods have been applied to cache design as well [6][11][12][13]. In this paper, we instead use dynamic voltage scaling (DVS) for leakage control [1]. While voltage scaling has seen extensive use for dynamic power reduction, short-channel effects also make it very effective for leakage reduction [2]. Furthermore, DVS also reduces gate-oxide leakage, which has increased dramatically with process scaling. Below, we discuss the traditional gated-Vdd and ABB-MTCMOS techniques for cache leakage reduction, as well as our proposed technique using DVS and compare the different techniques.

4.1 Gated-Vdd

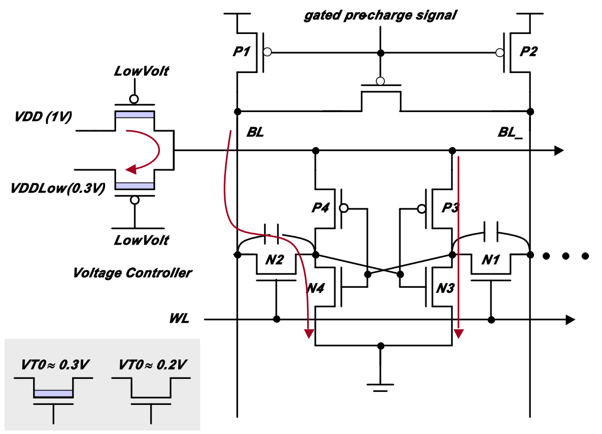
The gated-Vdd structure was introduced in [3][4]. This technique reduces the leakage power by using a high threshold (high-V_t) transistor to turn off the power to the memory cell when the cell is set to low-power mode. This high-V_t device drastically reduces the leakage of the circuit because of the exponential dependence of leakage on V_t. This method is very effective at reducing leakage, however it has the disadvantage 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 [14]. 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. 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 when the circuit is in drowsy mode. Although 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-Vdd technique, ABB-MTCMOS also requires special high-V_t devices for the control logic.

FIGURE 11. Implementation of a drowsy memory cell.



The arrows are leakage current path in the memory cell.

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 [1]. 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 substantially with voltage scaling. Since both voltage and current are reduced in DVS, a dramatic reduction in leakage power is 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 quickly in the DVS scheme than the ABB-MTCMOS scheme.

Figure 11 illustrates the circuit schematic of memory cells connected to the voltage-selection controller. No high-V_t device is used in the memory cell itself in our proposed technique as opposed to the method in [1] where high-V_t devices were used for the pass transistors that connect the memory's internal inverters to the read / write lines (N1 and N2). Because each cache line in [1] is controlled independently and each bit line is shared by all the cache lines in a sub-bank, all the read / write lines are maintained at high-Vdd, making it necessary to use high-V_{dd}, making it necessary to use high-V_t transistors for the pass gates in order to maintain acceptable leakage current [1].

However, since for the instruction cache, the entire sub-bank is switched between low-V_{dd} and high-V_{dd}, the read / write lines in each sub-bank are included in the DVS and no high-vt pass-transistors are needed. Avoiding the use of high-V_t device for the memory cells has several advantages against the previous approach [1]. First, the access time of the cache is not compromised. High-V_t devices show poor current driving capability at the same gate input voltage, which results in slower caches. Particularly for I-caches, which are critical in determining the cycle time of the processor, it is important to avoid any increase of the access time. This is why a direct-mapped cache is usually employed for an instruction cache since a set-associative cache is slower than a direct-mapped cache. Second, use of low-V_t pass-transistors reduces the dynamic power, since in our previous approach, significantly larger pass transistors are used to compensate the reduced current driving capability which is impaired by high-V_t threshold voltage.

In Figure 11, one PMOS pass gate connects the supply line to the normal supply voltage and the other connects it to the low supply voltage for. 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 can be implemented for each sub-bank or for each cache line.

A possible disadvantage of the circuit in Figure 11 is that it has increased susceptibility to noise and 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 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 memory cells in drowsy mode. Of course, this voltage scaling technique has less immunity against a single event upset (SEU) from alpha particles, but this problem can be relieved by process techniques such as silicon on insulator (SOI). Other static memory structures also suffer from this problem. making it necessary to implement error correction codes (ECC) even for non-drowsy caches. The second problem, variation of V_t, may be handled by choosing a conservative Vdd 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 a 64 x L_{eff} (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 cell for the control signal driver (three inverters); and 2) 1.5 equivalent memory cells for the wordline gating circuit (a nand gate). The total overhead is thus equivalent to 6.35 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 Figure 11, 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.

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 [15] for a 0.07um process. To measure the transition delay, we connected a 32K byte 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 L_{eff} 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 L_{eff}. 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 L_{eff} width PMOS pass-transistor.

5. Experiments

5.1 Experiment Setup

The evaluation methodology combines detailed processor simulation for performance analysis and for gathering event counts. In addition, analytical modeling is employed for estimating the energy dissipation for both the conventional caches and those employing drowsy techniques. We used the SimpleScalar toolset [16] to model an out-of-order speculative processor with a two-level cache hierarchy. The simulation parameters, listed in Table 1, roughly correspond to those of a present-day high-end microprocessor such as the HP PA-8000 or Alpha 21264.

We choose three different L1 cache sizes: 16K, 32K, and 64K bytes and various degrees of associativity: 1, 2, 4. In our experiments we use 4K bytes as the sub-bank or sub-array size, which corresponds to the page size of the virtual memory system. The trade-off of when using smaller sub-bank sizes is between more leakage reduction and increased wake-up penalties.

TABLE 1. Simulation parameters.

Parameters	Value
fetch/issue/decode/commit	4 instructions
fetch queue / speed	4 instructions / 1x
branch prediction	bimodal, 2k
BTB	512 entry, 4-way
RAS	8 entry
RUU size	64 entry
LSQ size	32 entry
integer ALUs/multi-divs	4 / 1
floating point ALUs / mul-div	1 / 1
memory bus width / latency	8 bytes / 80 and 8 cycles for the first and inter chunks
inst. / data TLBs	16 entry / 32 entry in each way, 4KB page size, 4-way, LRU, 30-cycle latency
L1 caches	16KB ~ 64KB, 1~4-way, 32B blocks, LRU, 1 cycle latency, write-back
L2 unified cache	256KB, 4-way, 64B line block, LRU, 8 cycle latency

In this paper benchmarks from the SPEC2000 suite were used, which were run on a modified SimpleScalar simulator. The benchmarks were compiled with GCC 2.6.3 using O2 level optimizations and were statically linked with library code. We ran 1 billion instructions for each simulation to complete the simulation within reasonable simulation time.

5.2 Experiment results and analysis

5.2.1 Prediction accuracy and run-time increase of the sub-bank predictors

Figure 12 shows the average prediction accuracies of the various sub-bank predictors illustrated in Section 3 for various cache configurations (see Equation 1). From the results of the experiment we observe that the prediction accuracy increases as the number of entries in the prediction buffers are increased. This in turn results in reduced run-time overhead compared to the base-line machines. However, prediction accuracy decreases as the cache sizes increase. Given a fixed sub-bank size, there are more sub-banks for larger caches, which require more prediction entries to maintain the same level of prediction accuracy. Of course, the positive effect of a larger cache still yields improved run-times.

The associativity of the cache also affects prediction accuracy. As the associativity increases the prediction accuracy decreases, because the correct set also needs to be predicted and awakened. In this experiment, we keep the target set prediction with the target sub-bank address in the prediction buffer entry, although one could use other way-prediction techniques as well [8][9][10]. Although we employ very simple BTB-like predictors, the prediction buffer is quite effective.

$$accuracy = \frac{total\ correct\ sub\text{-}bank\ predictions}{total\ wake\text{-}ups} \qquad \text{(EQ 1)}$$

When prediction information is kept in cache-line tags overall, the accuracy of the cache-line tag based predictor is between the 64 and 128 entry configurations of the sub-bank prediction buffer.There are two reasons accounting for this result: First, the prediction information is lost when tag lines containing valid predictions are replaced. This causes unnecessary wake-up cycles until the prediction information is updated. This situation is avoided in the prediction buffers, where there is no direct correlation between cache entries and predictions. Second, each cache tag line can keep only one prediction per line, while perhaps multiple predictions might be necessary. On the other hand the accuracy of the cache-line tag based predictor increases as the cache size is increased because the number of the predictor entries are proportional to the number of lines.

Figure 13 shows the run-time impact with and without the sub-bank predictors when the drowsy circuit is used. The run-time increase by the proposed cache architectures are measured against the base-line machines (see Equation 2). The experiment results show that the prediction technique using 128 entry prediction buffer can reduce the run-time impact by 83%, 74%, and 76% for 16K, 32K, and 64K byte caches compared to no prediction at all.

$$run\ time\ inc. = \frac{drowsy\ sim\ cycles - base\text{-}line\ sim\ cycles}{base\text{-}line\ sim\ cycles} \qquad \text{(EQ 2)}$$

According to the experimental results, the run-time increase of the 64K cache is smaller than that of the 32K byte cache, which does not mean that there are more sub-bank wake-up's in 32K byte cache machine compared to 64K byte one. The performance of the drowsy cache is measured against the base-line machine of each cache size. In other words, it is a relative performance against the base-line machine of each cache configuration. In addition, there are other factors which influence on the performance: The amount of the wake-up latencies hidden during out-of-order executions would be different for various cache sizes.

5.2.2 Predictor overhead and leakage power reduction

Table 2 shows the number of required bits of each predictor type for a 32K byte direct-mapped cache. The tag-based sub-bank predictor requires the same number of bits as a 64 entry prediction buffer.

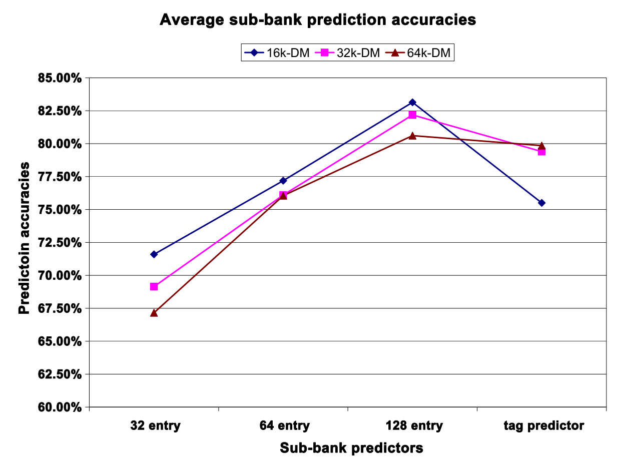
TABLE 2. Predictor overheads for 40-bit address, 32-byte line size, and 32K byte direct-mapped cache.

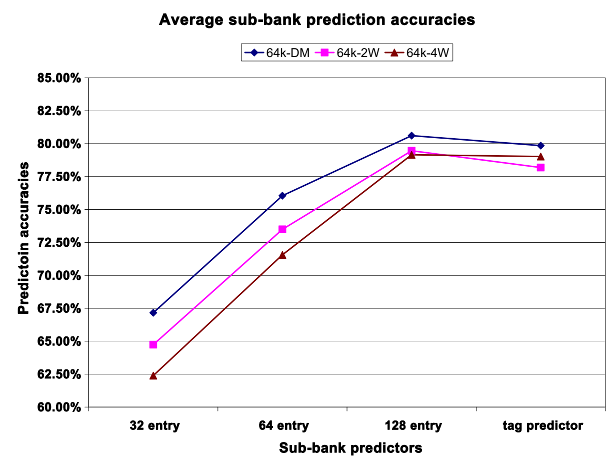
	32	64	128	tag
# req. bits	4096	8192	16384	8192

For example, in a 32-entry predictor, the number of required bits are 4096 bits (512 bytes), which is equivalent to 16 cache lines (32-byte per line). If we assume that the size of the cache is 64k-bytes the number of the cache lines is 2048 lines. The fractions of the 32, 64, and 128 entry predictors compared to the 64k cache are just 0.78%, 1.56%, and 3.12%.

Table 5 on page 12 shows leakage power reductions for SPEC2000 benchmarks when the DVS and

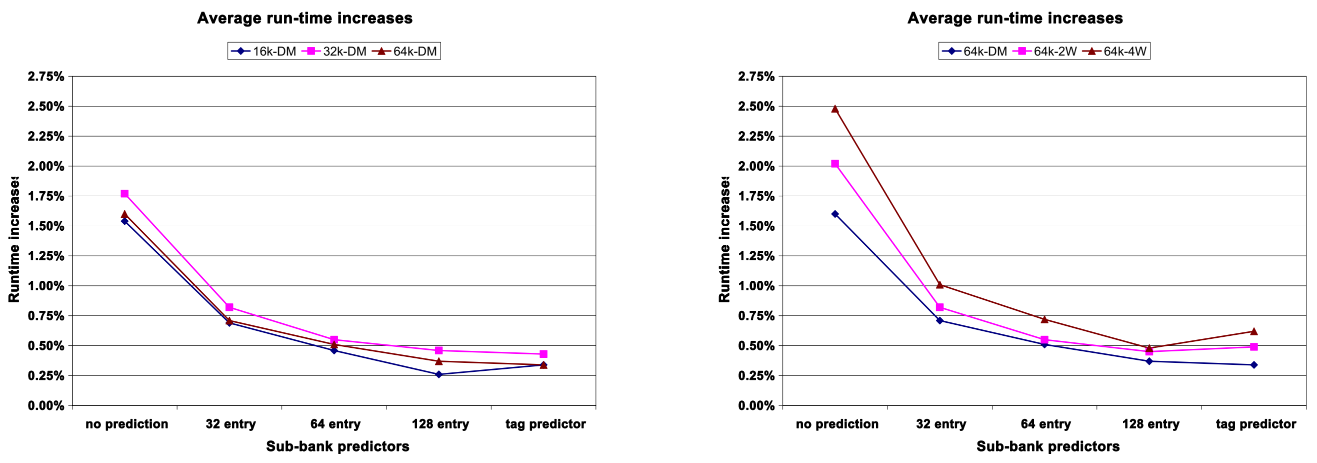
FIGURE 12. Average prediction accuracies of various sub-bank predictors.





See Table 3 on page 11 for the detailed experiment results

FIGURE 13. Average run-time increases of various sub-bank predictors.



See Table 4 on page 12 for the detailed experiment results

sub-bank prediction techniques are applied. The leakage energy reductions are measured against conventional caches. Leakage is reduced by about 75%, 88%, and 94% in the data array for 16K, 32K and 64K byte caches consisting of 4K byte sub-banks. However, since the tag array is always active and the use of the leakage reduction technique implies extra run time, the total energy reduction is slightly smaller. When this is accounted for, our measurements show the average leakage energy reductions are about 68%, 80%, and 86% for 16K, 32K, and 64K byte configurations, respectively.

6. Conclusion

During our investigations of drowsy instruction caches we found that our sub-banked cache with the next target sub-bank predictor—where only one sub-bank is active and the rest of the sub-banks are in drowsy mode—can reduce the cache's static power consumption by more than 86% for 64K byte caches. Furthermore, the experiment results show that the prediction technique using 128 entry prediction buffer can reduce the run-time overhead by 83%, 74%, and 76% for 16K, 32K, and 64K byte caches compared to the default policy where no prediction was employed. 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 to make more complex solutions unattractive. Future work will examine extending our techniques to other memory structures, such as branch predictors, L2, and L3 caches, and using BTB for the prediction buffer because there is a close relation between sub-bank transitions and the branches.

Acknowledgements

This work was supported by an Intel Graduate Fellowship, by DARPA contract number F33615-00-C-1678, by SRC 2001-HJ-904, by NSF Grant CCR-0073497, and by MARCO 98-DF-660.

Reference

[1] K. Flautner, et al. Drowsy Caches: Simple Techniques for Reducing Leakage Power. *To appear in Proc. of Int. Symp. Computer Architecture*, 2002

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

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

[4] S. Yang, et al. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. *Proc. of Int. Symp. High-Performance Computer Architecture*, 2001, pp. 147 -157.

[5] K. Ghose, and M. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. Low Power Electronics and Design, *Proc. of Int. Symp. on Low Power Electronics and Design*, 1999. pp. 70 -75

[6] S. Heo, et al. Dynamic Fine-Grain Leakage Reduction using Leakage-Biased Bitlines. *To appear in Proc. of Int. Symp. Computer Architecture*, 2002

[7] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. *Proc. of Int. Symp. High-Performance Computer Architecture*, 1996. pp. 244 -253

[8] D. Albonesi, Selective cache ways: on-demand cache resource allocation. *Proc. of Int. Conf. on Microarchitecture*, 1999. pp. 248 -259.

[9] M. Powell, et al. Reducing set-associative cache energy via way-prediction and selective direct-

mapping. *Proc. of Int. Conf. on Microarchitecture*, 2001. pp. 54 -65.

[10] B. Batson, and T. Vijaykumar. Reactive-associative caches. *Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques*. 2001. pp. 49 -60

[11] 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.

[12] 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.

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

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

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

[16] D. Burger and T. Austin. *The SimpleScalar Toolset, Version 2.0.* Tech. Rept. TR-97-1342, Univ. of Wisconsin-Madison, June 1997.

Appendix

The figures in the Tables are all rounded to 2 digits.

TABLE 3. Sub-bank predictor accuracies of direct-mapped caches for SPEC 2000 benchmarks.

SPEC 2000	prediction buffers (%)									tag predictor (%)		
	32 entry			64 entry			128 entry					
	16k	32k	64k	16k	32k	64k	16k	32k	64k	16k	32k	64k
bzip200	95	91	91	99	95	95	100	96	96	84	67	85
eon00	45	43	41	47	44	42	55	52	50	70	71	70
gcc00	45	42	41	54	51	50	62	59	59	55	68	60
parser00	86	86	87	88	85	86	85	78	78	73	83	76
twolf00	56	54	44	73	72	63	80	82	79	72	75	81
vpr00	53	58	56	59	61	60	78	77	76	81	79	82
crafty00	38	32	32	54	52	51	59	58	59	56	68	69
gap00	64	65	64	67	68	66	73	70	72	73	76	79
gzip00	82	82	72	69	62	86	100	100	75	54	73	51
mcf00	73	46	42	78	80	82	78	80	78	83	79	82
vortex00	41	37	35	53	48	44	57	57	54	52	63	64
ammp00	85	84	84	88	89	87	95	92	92	76	71	81
applu00	93	97	98	93	97	98	93	97	98	90	98	97
art00	100	100	100	100	100	100	100	100	100	92	97	93
equake00	46	43	39	75	73	72	82	82	84	77	86	85
galgel00	97	97	97	97	97	97	97	97	97	97	97	97
lucas00	100	100	100	100	100	100	100	100	100	75	83	83
mesa00	63	60	56	75	73	68	88	86	87	73	85	83
mgrid00	99	98	98	99	99	99	99	99	99	99	89	99
avg	72	69	67	77	76	76	83	82	81	76	79	80

TABLE 4. Run-time increases of direct-mapped caches for SPEC 2000 benchmarks.

SPEC 2000	no prediction (%)			prediction buffers (%)									tag predictor (%)		
				32 entry			64 entry			128 entry					
	16k	32k	64k	16k	32k	64k	16k	32k	64k	16k	32k	64k	16k	32k	64k
bzip200	0.8	0.8	0.8	0.1	0.2	0.1	0.0	0.1	0.1	0.0	0.0	0.0	0.2	0.4	0.4
eon00	6.1	7.0	7.2	4.1	4.4	4.2	3.9	4.4	4.4	2.9	3.3	3.4	2.0	3.0	3.2
gcc00	1.9	2.3	2.3	1.1	1.5	1.6	0.9	1.3	1.4	0.8	1.0	1.1	0.8	1.1	1.1
parser00	1.2	1.2	1.3	0.2	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.7	0.7
twolf00	2.8	3.0	3.0	0.9	1.2	1.2	0.5	0.8	0.8	0.3	0.5	0.4	0.5	0.6	0.7
vpr00	4.7	5.3	5.3	2.1	2.1	2.0	1.6	1.7	1.7	0.7	0.7	0.8	1.2	1.2	1.6
crafty00	3.7	5.2	4.3	2.3	3.8	3.1	1.5	2.7	2.5	1.3	2.3	2.2	1.6	2.0	2.2
gap00	3.5	3.7	3.7	1.6	1.6	1.8	1.3	1.3	1.4	1.0	1.2	1.0	1.0	1.1	1.0
gzip00	0.9	0.9	2.4	0.5	0.5	0.6	0.5	0.6	0.1	0.0	0.0	0.5	0.1	0.9	1.8
mcf00	5.0	8.7	9.7	0.9	3.4	3.7	0.5	0.7	0.2	0.5	0.7	0.4	1.0	1.5	1.9
vortex00	5.4	8.6	3.2	2.8	3.7	1.5	2.0	3.0	3.1	0.2	5.1	2.6	1.0	2.4	2.6
ammp00	0.1	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
applu00	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
art00	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
equake00	9.0	10.6	12.5	5.4	6.0	6.9	1.4	1.6	1.9	0.5	0.6	0.6	2.1	1.6	1.8
galgel00	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lucas00	4.2	4.2	4.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	1.1	1.1
mesa00	5.9	5.5	5.0	2.1	2.4	2.2	1.4	1.5	1.2	0.4	0.4	0.3	1.4	1.8	1.3
mgrid00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
avg	1.5	1.8	1.6	0.7	0.8	0.7	0.5	0.6	0.5	0.3	0.5	0.4	0.3	0.4	0.3

TABLE 5. Leakage energy reduction of direct-mapped caches for SPEC 2000 benchmarks.

SPEC 2000	no prediction (%)			prediction buffers (%)									tag predictor (%)		
				32 entry			64 entry			128 entry					
	16k	32k	64k	16k	32k	64k	16k	32k	64k	16k	32k	64k	16k	32k	64k
bzip200	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
eon00	67	79	85	68	80	86	68	80	86	68	80	86	68	84	86
gcc00	69	80	86	69	80	86	69	80	86	69	80	86	69	84	86
parser00	69	80	86	69	81	86	69	81	86	69	81	86	69	84	86
twolf00	68	80	86	69	80	86	69	80	86	69	81	86	69	84	86
vpr00	68	80	86	68	80	86	69	80	86	69	81	86	69	84	86
crafty00	68	80	86	68	80	86	69	80	86	69	80	86	69	84	86
gap00	68	80	86	69	80	86	69	80	86	69	80	86	69	84	86
gzip00	69	80	86	69	81	86	69	81	86	69	81	86	69	84	86
mcf00	68	79	85	69	80	86	69	81	86	69	81	86	69	84	86
vortex00	67	79	86	68	80	86	68	80	86	69	80	86	69	84	86
ammp00	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
applu00	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
art00	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
equake00	66	79	85	67	79	85	69	80	86	69	81	86	68	84	86
galgel00	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
lucas00	68	80	86	69	81	86	69	81	86	69	81	86	68	84	86
mesa00	67	80	86	68	80	86	69	80	86	69	81	86	69	84	86
mgrid00	69	81	86	69	81	86	69	81	86	69	81	86	69	84	86
avg	69	80	86	69	80	86	69	81	86	69	81	86	69	84	86