Single-V_{DD} and Single-V_T Super-Drowsy Techniques for Low-Leakage High-Performance Instruction Caches

Nam Sung Kim†, Krisztián Flautner‡, David Blaauw, Trevor Mudge
Intel Corp.†, ARM Ltd.‡, University of Michigan
nam.sung.kim@intel.com†, kristian.flautner@arm.com‡, {blaauw, tnm}@eecs.umich.edu

ABSTRACT

In this paper, we present a circuit technique that supports a super-drowsy mode with a single-V_{DD}. In addition, we perform a detailed working set analysis for various cache line update policies for placing lines in a drowsy state. The analysis presents a policy for an instruction cache and shows it is as good as or better than more complex schemes proposed in the past. Furthermore, as an alternative to using high-threshold devices to reduce the bitline leakage through access transistors in drowsy caches, we propose a gated bitline precharge technique. A single threshold process is now sufficient. The gated precharge employs a simple but effective predictor that almost completely hides any performance loss incurred by the transitions between sub-banks. A 64-entry predictor with 3 bits per entry reduces the run-time increase by 78%, which is as effective as previous proposals that used content addressable predictors with 40 bits per entry. Overall, the combination of the proposed techniques reduces the leakage power by 72% with negligible (0.4%) run-time increase.

Categories and Subject Descriptors:

B.3.2 [Memory Structures]: Design Styles—Cache memories; **B.7.1 [Integrated Circuits]:** Types and Design Styles—Memory technologies

General Terms: Design, Performance

Keywords: Low power. Leakage current

1. Introduction

Until very recently, only dynamic power has been a significant source of power consumption, and Moore's law has helped to control it. Shrinking processor technology below 100*nm* has allowed, and actually required, reducing the supply voltage to reduce dynamic power consumption. However, smaller geometries with a low threshold voltage exacerbate leakage, so static power is beginning to dominate the power consumption equation. In particular, the leakage power of on-chip caches is becoming a significant problem. While dynamic power is dissipated by a cache sub-bank accessed, static leakage power is dissipated by all the sub-banks even if they are not accessed.

To alleviate this problem, transistors in on-chip caches could be designed for low leakage, for example, by assigning them a high threshold voltage, V_T, or by controlling the V_T with adaptive body biasing or, if a better balance of speed and power is required, by employing dual V_T, or combining those circuit techniques with microarchitectural controls [1-3]. A complementary approach, the drowsy cache, a low-leakage static memory circuit technique based on dynamic voltage scaling, has received significant attention, because of its memory state-preserving capability and short wake-up latency.

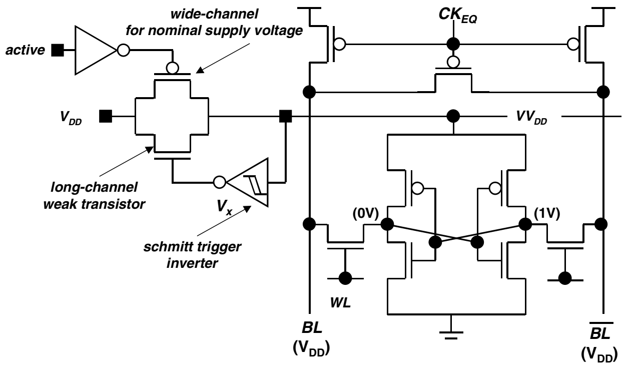
There have been a number of earlier studies that have used dynamic voltage scaling to reduce leakage power in instruction caches [6, 8]. In [6, 7], a sub-bank-based cache leakage control was proposed with a *next target sub-bank predictor.* While this technique was effective at reducing both the cell and bit-line leakage power with only a slight run-time increase, it has the potential to frequently cycle an entire sub-bank on and off if a small loop spans two sub-banks. In such cases, a significant fraction of leakage reduction is offset by the frequent sub-bank switching to wake up whole cache lines. In addition, the next target sub-bank predictor implementation can require a *content addressable memory* (CAM) that is expensive in terms of power and area. In [8], to reduce the penalties waking up drowsy lines, a small number of prediction bits were added to the BTB, but it still used the same cache line control policy and window size as those reported by [5], a scheme that was originally proposed for data caches. However, instruction and data caches have different access patterns or working set re-use characteristics. Hence, a policy and window size optimized for data caches is almost certain to be sub-optimal for instruction caches. Furthermore, [8] assumed that high-V_T access transistors were used to reduce the bit-line leakage power that is responsible for 20% of total leakage power in single-port static random access memories. Unfortunately, the high-V_T access transistors can increase instruction cache access time by nearly 10%, which is critical in determining a microprocessor clock cycle time.

In this paper we propose solutions to these potential problems as follows. First, we propose a single-V_{DD} drowsy cache technique. The circuit technique in [5-8] requires an extra supply voltage source to support the stand-by or drowsy mode of cache lines; the extra supply voltage source incurs additional interconnect routing space. Second, we analyze instruction cache working sets and their re-use characteristics, and evaluate trade-offs between performance, the energy of cache control policies, and window sizes or number of cycles between re-entering drowsy states. This analysis reveals that a simple policy and window size optimized for instruction caches can minimize run-time increases, saving a substantial amount of leakage power without employing any complex mechanisms. Finally, we propose a gated bitline precharge technique to reduce the bitline leakage power of instruction caches instead of using high-V_T access transistors, which slow down cache access. However, the on-demand gated precharge can result in significant average run-time increases. To minimize these, we propose a next target sub-bank prediction technique that enables an inactive sub-bank just before it is accessed. The proposed predictor is much simpler but as effective as the CAM-based next target sub-bank predictor presented in [6].

The remainder of this paper is organized as follows. Section 2 explains the single-V_{DD} super-drowsy cache circuit technique. Section 3 investigates cache line update policies and window sizes for instruction caches. Section 4 proposes a new class of next target sub-bank predictor with a delayed precharge clock turn-off technique. Section 5 details the experimental methodology and presents results for the proposed techniques. Section 6 concludes the paper.

2. Single-V_{DD} Super-Drowsy Cache Circuit

Figure 1 shows a circuit for a single-V_{DD} cache line voltage controller to support drowsy caches. Once the "active" signal in Figure 1 is turned off the PMOS transistor supplying the nominal voltage is turned off and the cache line is placed into drowsy mode. Since there is leakage current through memory cells connected to "VV_{DD}", the voltage level of "VV_{DD}" decreases. However, as the "VV_{DD}" reaches a minimum voltage level preserving memory state, the Schmitt trigger inverter turns on the long-channel weak NMOS transistor. The weak NMOS transistor is *conservatively* sized to maintain a certain minimum level of the supply voltage where the



wide-channel for nominal supply voltage
active
V_{DD}
long-channel weak transistor
V_x
schmitt trigger inverter
CK_{EQ}
VV_{DD}
(0V)
(1V)
WL
BL (V_{DD})
$\overline{BL}$ (V_{DD})

Figure 1: Single-V_{DD} cache line voltage controller circuit.

Schmitt trigger inverter is triggered so that the memory state is not lost.

To derive this minimum state-preserving voltage, we sweep the supply voltage of a memory cell from 1 to 0V using HSPICE. As the supply voltage is scaled down, the voltages of the complementary nodes of the memory cell approach each other and becoming indistinguishable below 100*m*V. This implies that the memory cell state has been destroyed. To keep a meaningful state, one cross-coupled node should maintain logic "1" while the other should hold logic "0". There is no way to recover the original logic state once the voltage level of both nodes become indistinguishable even if the supply voltage level is fully restored. Therefore, the stand-by voltage must be higher than the minimum state-preserving voltage. Allowing for a 3σ process variation for V_T and L_{eff}, our HSPICE studies indicated that 165*mV* would be the minimum state-preserving voltage of a 6 transistor memory cell [9]. However, we set the stand-by voltage to 250*mV* to ensure against other noise issues such as supply voltage fluctuation. At 250*mV* we can reduce the cell leakage power by 98%. This analysis shows that we can lower the supply voltage much further than the voltage originally reported by [5] with the same technology.

3. Super-Drowsy Cache Line Control Policy

3.1 Working Set Analysis

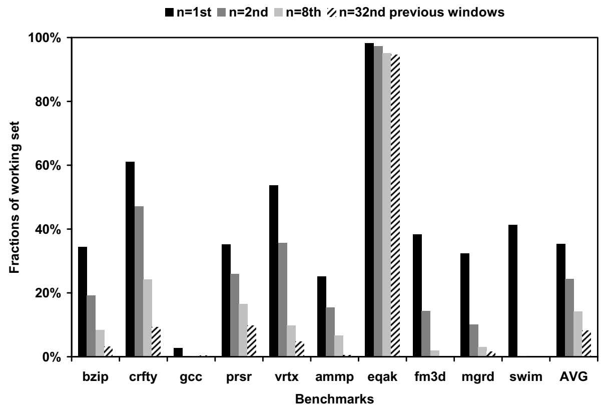
Figure 2-(a) shows a 32KB 2-way set associative data cache working set re-use characteristics—the fraction of accesses that are the same as in the *n-th* previous 2K-cycle observation windows for the sub-set of SPEC2K benchmarks (see Table 1 in Section 5 for the processor simulation parameters). The results in the figure specify what fraction of references in a current window are to lines that had been accessed 1, 2, 8, or 32 2K-cycle windows before. Based on this experimental results reproduced from [5], we were able to make two observations on data caches. First, the average fraction of the working set of a 32KB 2-way set associative data cache is around 10% in a 2K-cycle observation window. This implies that only a small fraction of data cache lines need be in active mode, while the rest of the lines can be in drowsy mode to reduce the data cache leakage power. Second, more than 60% of the working sets in the 2K-cycle window will not be used in next consecutive windows. This means that 40% of the working set put into drowsy mode will not need to be woken up in the near future; past accesses are not always a good indication of the future cache line uses. In the case of *equake*, whose data cache working set is very small (5%), the high working re-use characteristic does not impact the run-time very much.

Unfortunately, the data cache working set characteristics, which makes the policy in [5] work well, are not replicated for instruction caches, because of the different access patterns of instruction caches. When a 32KB 2-way set associative instruction cache is used with a 2K cycle observation time window, the average working set percentage of the instruction cache is similar to that of the data cache (~10%), but in many workloads such as *crafty* and *vortex*, the fractions of the working sets are very high, 28% and 21%, respectively. When the policy in [5] is employed for instruction cache leakage power control, the performance loss of those workloads will be significant because more cache lines need to be woken up in a fixed time window size. Figure 2-(b) shows the same working set re-use characteristics for the same size instruction cache. The fractions of the re-used working sets from the previous windows in the instruction cache are much higher than those in the data cache. In other words, the working sets of instruction caches barely change for long cycles in many workloads. In other words, the past access patterns for the instruction cache lines are good indicators for the future usage of the instruction cache lines.

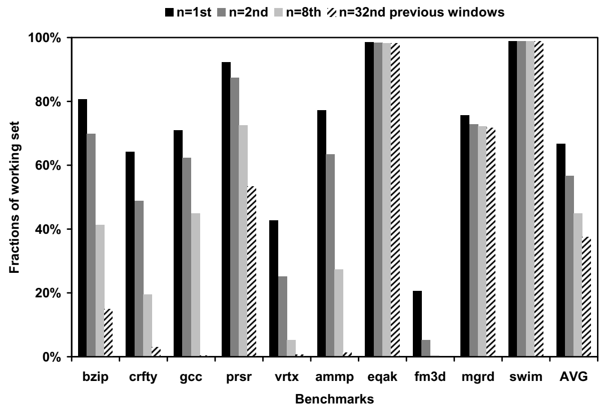
3.2 Policy and Window Size Effects

Considering the working set re-use characteristics and the high memory impact of the instruction caches, the *noaccess* policy (only lines that have not been accessed during a fixed time period are put into drowsy mode) will have less run-time increases than the *simple* policy (all lines in the cache are put into drowsy mode periodically). The noaccess policy can be implemented with a hierarchical counter implementation technique—one global counter and a 2-bit counter per cache line, see [4]. The "noaccess INT" and "simple INT" plots in Figure 3 show the run-time increase and the fraction of drowsy lines of the noaccess and simple policies. In this experiments, we used 2K, 8K, 32K, and 128K update window sizes and a 1-cycle drowsy-line wake-up latency. As expected, the "noaccess" policy tracking the past access patterns of each cache line performs better in terms of both the run-time increase and leakage reduction for all workloads and update windows. The noaccess policy shows 2.4%, 0.9%, 0.1%, and ~0.0% average run-time increases with 91%, 85%, 81%, and 78% average fractions of drowsy lines, while the simple

Figure 2: Working set re-use characteristics of a 32KB 2-way set associative data and instruction caches in (a) and (b).



(a)



(b)

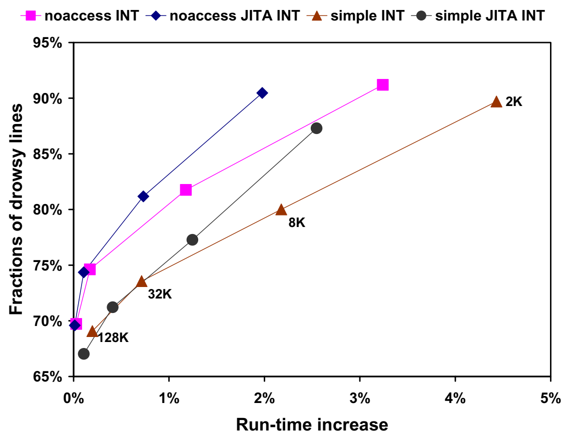


Figure 3: Policy and window size effects.

policy shows 4%, 2%, 1%, and ~0% average run-time increases with 91%, 84%, 80%, and 78% average fractions of drowsy lines for 2K, 8K, 32K, and 128K window sizes. Compared to the simple policy, the noaccess policy reduces the run-time increases by 41%, 52%, 81%, and 88% with a slight decrease in fractions of drowsy lines for 2K, 8K, 32K, and 128K window sizes.

In addition, "noaccess JITA INT" and "simple JITA INT" in Figure 3 show the run-time increase and the fraction of drowsy lines of the noaccess and simple policies with the *just-in-time-activation* (JITA) of [8]. Except for the 2K window size case, "simple-JITA" cannot outperform the noaccess policy without JITA. However, the experiment results shown in Figure 3 suggest that the 2K window size used in [8] does not seem to be acceptable for high-performance applications because of significant run-time increases. Considering both the leakage reduction and run-time increases, the noaccess policy with a window size between 8K and 32K cycles seems to be the optimal range and policy for 32KB 2-way set associative instruction caches. This is a much larger window size than the one for the same size data caches.

In summary, rather than employing additional techniques to reduce the run-time increases like those in [8], we can achieve similar leakage reduction and better performance with the above policy and window size that has been optimized for instruction caches.

4. Super-Drowsy Gated Bitline Precharge

In data caches, high-V_T access transistors are used to reduce the bitline leakage, but it is a less desirable solution for instruction caches, because the instruction cache access or cycle time is critical in determining the cycle-time of the processor. To reduce the bitline leakage power, we can employ a gated bitline precharge (or bitline isolation) technique. In this technique, the bitlines of a currently accessed sub-bank are precharged and the precharge enable signals for the inactive sub-banks are gated to isolate the bitlines from the supply voltage. This reduces the bitline leakage power. However, this technique—on-demand gated precharge—incurs an extra penalty cycle when a cache access pointer transitions from one to another sub-bank. According to [6], the run-time increases due to those extra cycles can be up to 20% for some benchmark programs.

To reduce the run-time increase of the sub-bank transitions, we need to identify their sources and enable the precharge signals of the next target sub-bank in advance. The fundamental insight is that the transitions between sub-banks are often correlated with specific types of instructions. For example, the program counter, which is the instruction cache access index or pointer, remains in small cache regions for relatively long periods as a result of program loops. On the other hand, there are often abrupt changes in the cache access pointer on subroutine calls and returns. Most conditional branches stay within the current cache region and it is rare that these branches jump across page boundaries. In both SPEC2K integer and floating-point workloads, the unconditional jumps are responsible for an average of 40% of the sub-bank transitions. The sequential accesses

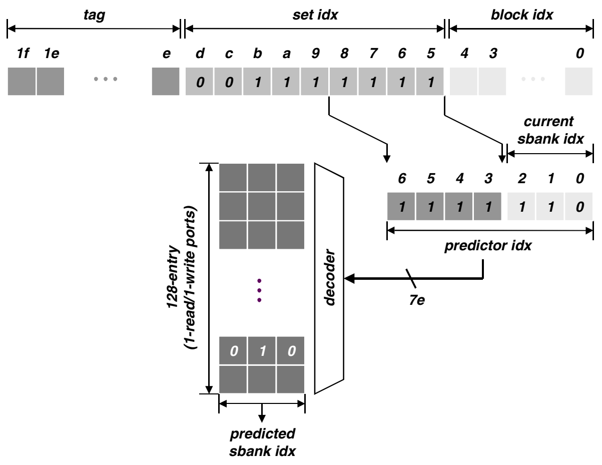


Figure 4: Next target sub-bank index generation.

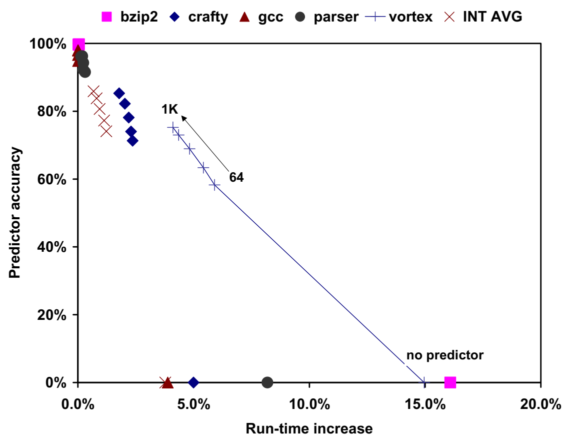
to the next set in a different way (only for set associative caches) also causes 40% of the transitions. Hence, unconditional jumps and sequential accesses between two sub-bank boundaries are dominant factors of the sub-bank transitions.

4.1 Next Target Sub-Bank Predictor

The sub-bank transitions triggered by the unconditional jumps are quite predictable. The basic idea of the proposed next target sub-bank predictor is as follows. When an instruction triggering the transitions is encountered (e.g., from the sub-bank `0x6` to `0x2`), a next target sub-bank predictor index (e.g., `0x7e`) is generated from the cache set index of the previous cycle access (e.g., `0x7f`) and the sub-bank index of the current cycle one (e.g., `0x6`). With the predictor index, the target sub-bank index (e.g., `0x2`) is stored in the predictor; see Figure 4 for how to generate the predictor index. When the cache set of `0x7f` in the sub-bank `0x6` is accessed in later cycles, the pointer gives the next target sub-bank index—`0x2`. Since the instructions cache equipped with a single port only allows the processor to access a single cache line in a cycle, the set address instead of the individual instruction address is enough to detect the sub-bank transitions and to index predictors.

Figure 5 shows the run-time increase vs. the predictor accuracy for a sub-set of SPEC2K integer workloads. We used a 32KB, 2-way set associative instruction cache of 8×4KB sub-banks with 64-, 128-, 256-, 512-, and 1K-entry predictors. As we increase the number of the predictor entries, the average accuracy increases by 76%, 80%, 83%, 86%, and 88% with 3.3%, 0.9%, 0.7%, 0.6%, 0.5%, and 0.4% average run-time increases, respectively for 64-, 128-, 256-, 512-, and 1K-entry predictors. Compared to the run-time increase without the predictors (on-demand gated bitline precharge), the proposed predictor reduces the run-time increases by 72%, 78%, 81%,

Figure 5: Run-time increase vs. predictor accuracy.



84%, and 87%, respectively for 64-, 128-, 256-, 512-, and 1K-entry predictors. For example, with a 64-entry predictor which was the smallest size predictor we studied, we could reduce the run-time increases of *bzip2*, *crafty*, and *vortex* from 16%, 5%, and 15% to ~0.0%, 2%, and 6%, respectively. For those workloads, we reduce the run-time increases by ~100%, 53%, 96%, and 61%, respectively. Those results prove that a small size predictor is also very effective to reduce the performance loss by the gated bitline precharge. Theoretically, this technique reduces the bitline leakage power by 88% in the 8×4KB sub-bank instruction cache because only one sub-bank is precharged among 8 sub-banks. However, it takes a finite time until the floated bit-line conditions are balanced [10]. Therefore, the actual bit-line leakage saving is somewhat less than 88%. In previously proposed technique [6], a *content addressable memory* (CAM), which is expensive, is used to store and match the full address bits of instructions for the next target sub-bank predictor. However, it turns out that we can achieve a similar accuracy with a much simpler structure needing less hardware.

5. Experiments

5.1 Simulation Methodology

The architectural simulator used in this study are derived from the SimpleScalar/Alpha 3.0 tool set, a suite of functional and timing simulation tools for the Alpha AXP ISA; see Table 1 for the processor simulation parameters. The processor microarchitectural parameters model a high-end microprocessor similar to an Alpha 21264. To perform our evaluation we collected results from all 25 of the SPEC2000 benchmarks†. For the circuit simulation parameters, we used the 70*nm* BPTM [11] models, whose V_{DD} and V_T are 1.0 and ~0.2V, respectively. Our memory cell consumes 0.0778μW per active bit at V_{DD} = 1V and 0.0167μW per drowsy bit at V_{DD} = 0.25V with 1-cycle and 115fJ wakeup latency and energy penalties; steady-state drowsy bit leakage power become 0.00387μW when bit-line precharge devices are *gated* (or turned off). In addition, we set the microprocessor clock cycle time to 12×FO4 or 395ps to calculate total energy consumption from total leakage power and cycles.

5.2 Leakage Reduction and Run-Time Increase

Table 2 shows the run-time increase and the normalized leakage power of the proposed techniques—the noaccess drowsy policy with the JITA, and the drowsy policy with the gated bitline precharge (GPB in Table 2). A 32KB, 2-way set associative instruction cache of 8 4KB sub-banks with a 32K-cycle update window size for the noaccess policy, an 1K-entry predictor, and a 16-cycle turning off delay period for the gated bitline precharge is used; although cache access pointer transits from one sub-bank to another, the precharge circuits of the previous sub-bank remains turned on for a specified number of cycles. The noaccess policy with the 32K-cycle update window shows only a 0.07% run-time increase with the 60% leakage power reduction assuming we reduces only cell leakage power by voltage scaling. In the worst case, *vortex* shows a 0.85% performance loss with a 21% leakage power saving. When we combine the gated precharge with the noaccess policy to reduce the bitline leakage power, the average run-time increase is 0.38% —less than 0.5% performance loss—while reducing leakage power by 72%. In the worst case, *vortex* shows the 3.84% performance loss with 31% leakage saving. In particular, *bzip2*, *gcc*, *mcf*, *parser*, *vpr*, *applu*, *art*, *galgel*, *lucas*, and *swim* — nearly the half of the entire SPEC2K workloads — show negligible or no performance losses while achieving more than 80% leakage power reduction for the combined technique. In terms of the area overhead by the 1K-entry next target sub-bank predictor, it is less than 1.2% in terms of the number of bits compared to the 32KB, 2-way set associative instruction cache.

Table 2: Run-time increase and leakage power.

	run-time increase (%)		normalized leakage (%)	
	noaccess	w/ GBP	noaccess	w/ GBP
bzip2	0.05	0.04	31	17
crafty	0.62	2.01	72	61
gcc	0.03	0.03	30	16
mcf	0.00	0.01	28	14
parser	0.01	0.09	31	17
vortex	0.85	3.84	79	69
vpr	0.01	0.02	31	16
ammp	0.07	0.07	34	19
applu	0.01	0.01	33	19
art	0.00	0.01	28	13
equake	0.00	1.10	41	30
facerec	0.01	0.17	30	16
galgel	0.00	0.00	28	13
swim	0.00	0.01	32	17
AVG†	0.07	0.38	41	28

Table 1: Processor simulation parameters.

Out of Order Execution	4-wide fetch / decode / issue / commit, 64 RUU, 32 LSQ, speculative scheduling
Functional Unit (latencies)	4 integer ALUs (1), 2 floating point ALUs (2), 1 integer MULT/DIV (3/20), 1 floating point MULT/DIV/SQRT (4/12/24), 2 general memory ports
Branch Prediction	combined bimodal (4K-entry) / gshare (4K-entry) w/ selector (4K-entry), 32-entry RAS, 512-entry 4-way BTB, 11-cycle misprediction recovery
Memory System (latencies)	32KB 2-way 32-byte block L1 inst (1) and data caches (1), 512KB 4-way 64-byte block unified L2 cache (12), 128-entry fully associative inst and data TLB (28/28) main memory (80/8)

6. Conclusion

In this paper, we present a single-V_{DD} and a single-V_T drowsy cache design that reduces both memory cell and bitline leakage power. The techniques are based on: 1) dynamic voltage scaling to reduce the cell leakage and 2) gated precharge to reduce the bitline leakage power. To control the cell leakage power of instruction cache lines, we investigated various policies and window sizes and we presented an optimal policy and window size derived from the comprehensive working-set and re-use analysis of an instruction cache. To reduce the bitline leakage power without using high-V_T transistors in memory cells, we proposed a gated bitline precharge technique. To minimize the run-time impact, a simple but effective next target sub-bank predictor with a delayed sub-bank precharge turn-off was proposed.

Reference

[1] T. Douseki, et al., “A 0.5-1V MTCMOS/SIMOX SRAM macro with multi-V_{TH} memory cells,” IEEE SOI Conf., 2000.

[2] K. Nii, et al., “A low power SRAM using auto-backgate-controlled MT-CMOS,” ISLPED, 1998.

[3] F. Hamzaoglu, et al., “Analysis of dual-V_T SRAM cells with full-swing single-ended bitline sensing for on-chip cache,” IEEE Trans. on VLSI Systems, Apr. 2002.

[4] S. Kaxiras, et al., “Cache decay: Exploiting generational behavior to reduce cache leakage power,” ISCA-28, 2001.

[5] K. Flautner, et al., “Drowsy caches,” ISCA, 2002.

[6] N. Kim, et al., “Drowsy instruction caches,” MICRO, 2002.

[7] N. Kim, et al., “Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power,” IEEE TVLSI, Feb., 2004.

[8] J. Hu, et al., “Exploiting program hotspots and code sequentiality for instruction cache leakage management,” ISLPED, 2003.

[9] C. Neau, et al., “Optimal body bias selection for leakage improvement and process compensation over different technology generations,” ISLPED, 2003.

[10] S. Yang, “Near-optimal precharging in high-performance nanoscale CMOS caches,” MICRO-36, 2003.

[11] http://www-device.eecs.berkeley.edu/~ptm