Using Branch Prediction Information for Near-Optimal I-Cache Leakage

Sung Woo Chung[1,*] and Kevin Skadron[2]

[1] Division of Computer and Communication Engineering, Korea University, Seoul 136-713, Korea
swchung@korea.ac.kr
[2] Department of Computer Science, University of Virginia
Charlottesville 22904, USA
skadron@cs.virginia.edu

Abstract. This paper describes a new on-demand wakeup prediction policy for instruction cache leakage control that achieves better leakage savings than prior policies, and avoids the performance overheads of prior policies. The proposed policy reduces leakage energy by more than 92% with only less than 0.3% performance overhead on average. The key to this new on-demand policy is to use branch prediction information for the wakeup prediction. In the proposed policy, inserting an extra stage for wakeup between branch prediction and fetch, allows the branch predictor to be also used as a wakeup predictor without any additional hardware. Thus, the extra stage hides the wakeup penalty, not affecting branch prediction accuracy. Though extra pipeline stages typically add to branch misprediction penalty, in this case, the extra wakeup stage on the normal fetch path can be overlapped with misprediction recovery. With such consistently accurate wakeup prediction, all cache lines except the next expected cache line are in the leakage saving mode, minimizing leakage energy.

Keywords: Instruction Cache, Low Power, Leakage, Drowsy Cache, Branch Prediction.

1 Introduction

As process technology scales down, leakage energy accounts for a significant part of total energy. The International Technology Roadmap for Semiconductor [23] predicts that by the 70nm technology, leakage may constitute as much as 50% of total energy dissipation. In particular, the leakage energy for on-chip caches is crucial, since they comprise a large portion of chip area. For instance, 30% of the Alpha 21264 and 60% of the StrongARM are devoted to cache and memory structures [13]. However, cache size can not be decreased to reduce leakage power since cache size is directly related to the performance.

There have been four major circuit techniques to reduce leakage energy dynamically: ABB (Adaptive-reverse Body Biasing) MTCMOS [16], DRG (Data-Retention Gated-ground) [1], Gated-Vdd [17], and DVS for Vdd (which is also called drowsy cache) [3]. In the ABB MTCMOS technique, threshold voltage is dynamically

[*] Corresponding Author.

changed but the wakeup penalty between the active mode and the leakage saving mode is long, which makes it difficult for use in L1 caches [4]. DRG retains the data while reducing leakage by gating ground and using remaining leakage to retain cell contents, but the wakeup penalty is long. Thus, this technique may be inappropriate for timing critical caches such as an L1 cache, even if it is effective for less timing critical caches such as L2 [10]. The gated-Vdd technique reduces the leakage power by breaking the connection from the supply voltage (Vdd) or ground (the difference compared to DRG is that a larger sleep transistor is used and cell contents are not preserved) when the cell is put to sleep. While this technique dramatically reduces the leakage, its main disadvantage is that it does not preserve the state of the data in the sleep mode [4]. When the line is needed after it has been put into the leakage saving mode, the line must be refetched from a lower-level memory, which leads not only to additional dynamic energy consumption but also to performance degradation. To prevent these costs, conservative prediction policies should be employed [5][20][21]. Gated-Vdd may, however, be suitable for some L1 data caches where re-fetch penalty is short [12]. Another leakage saving technique is to lower the supply voltage. In this technique, data is not lost when the cache line is in the leakage saving mode (called "drowsy" mode). In the drowsy mode, data is retained, although it can not be accessed for read or write operation. Fortunately, most cache lines are unused for long periods due to temporal locality. Thus, by putting infrequently used cache lines into drowsy mode and keeping frequently accessed cache lines in the active mode, much leakage power is reduced without significant performance degradation. Please note that there is a wakeup penalty to restore the voltage level of the Vdd from the drowsy mode into the active mode. However, the wakeup penalty is expected to be one cycle in 70nm process technology [3]. There has been concern that drowsy cache is more susceptible to soft errors than conventional caches [10]. Fortunately, instructions are read-only and must be protected by parity even in the absence of drowsy techniques. In the infrequent cases when an error is detected, the instruction only has to be refetched.

Among the above four techniques, drowsy technique is most suitable for L1 instruction caches, since it retains data and has short wakeup penalty. In order to prevent (or hide) the wakeup penalty of the drowsy cache, many prediction policies have been proposed. The easiest policy is "no prediction": to place all the cache lines into the drowsy mode periodically and restore the voltage level of Vdd of accessed cache lines, suffering the wakeup penalty. It performs well with data caches because they have high temporal locality, leading to little performance loss, and out-of-order processors can often tolerate extra latency from waking up lines [3]. For instruction caches, however, this "no prediction" technique does not perform well, because any wakeup penalty that stalls fetching directly impacts the performance. Many prediction policies have been proposed for instruction caches. (Details will be explained in the next section). None of them has simultaneously shown consistent leakage energy reduction with negligible performance degradation. In this paper, we propose a new *on-demand* wakeup prediction policy for an instruction cache. By on-demand, we mean that *only currently necessary cache line(s) needs to be awake*. This technique takes advantage of the fact that we can accurately predict the next cache line by using the branch predictor. Thus, the wakeup prediction accuracy capitalizes on branch predictors that have already proven to be very accurate [14]. A further advantage compared to previous policies is that the proposed policy does not require an additional predictor. To utilize the branch predictor for wakeup prediction, we can allow a pipeline stage between branch prediction and instruction cache fetch. Allowing the branch predictor to be accessed one cycle earlier permits the branch prediction outcome to be used for wakeup, without harming branch prediction accuracy or requiring additional wakeup prediction hardware. Please note that this approach does not suffer the traditional branch-misprediction overhead of inserting extra stage in the pipeline. On a branch misprediction, the extra wakeup stage is overlapped with misprediction recovery. For further details, see Section 3.

This work focuses on use of drowsy cache (actually super-drowsy cache [9], explained in Section 2) for the leakage saving circuit technique. In this paper, we distinguish the wakeup prediction *policy* from the leakage saving *circuit technique*. The wakeup prediction policy predicts which cache line will be woken up, while the leakage saving circuit technique is the mechanism for putting lines to sleep and waking them up, independent of the prediction policy.

2 Background

Kim et.al proposed a refinement of the drowsy technique, called *super-drowsy cache* [9]. A single-Vdd cache line voltage controller with Schmitt trigger inverter replaces multiple supply voltage sources in order to alleviate interconnect routing space. In addition, the on-demand gated bitline precharge technique [19] is employed to reduce the bitline leakage. We apply our prediction policy to the super-drowsy cache because it is the most advanced circuit technique for instruction cache leakage control as far as we know.

The success of the drowsy-style cache depends on how accurately the next cache line can be predicted and woken up. Especially for an instruction cache, accuracy is crucial since the accuracy directly affects performance degradation. A simple policy is *noaccess* [3]: This uses per-line access history and puts all the unused lines into drowsy mode periodically. For more accurate wakeup prediction, two prediction policies were proposed for a drowsy instruction cache [8] – *NSPB* (Next Subcache Prediction Buffer) and *NSPCT* (Next Subcache Predictor in Cache Tags). Additional storage is required to predict the next subbank (not a cache line) using NSPB, whereas cache tags are extended to provide the subbank predictor in NSPCT. Therefore, NSPCT requires less hardware overhead but is comparable to NSPB in accuracy (performance loss is 0.79%). However, leakage reduction is weak [8] due to large subbank turn-on energy. Zhang et.al. proposed the *Loop* policy [21] where all cache lines are put into the drowsy mode after each loop was executed. This bears some similarity to the *DHS* (Dynamic HotSpot Based Leakage Management) policy, which was proposed in [5]. DHS makes use of the branch target buffer (BTB), since branch behavior is an important factor in shaping the instruction access behavior. In the DHS policy, the global turn-off (drowsy) signal is issued when a new loop-based hotspot is detected. Thus this policy can lower the supply voltage of unused cache lines before the update window expires by detecting that execution will remain in a new loop-based hotspot. The *DHS-PA* (DHS-Per Access) policy employs a Just-In-Time-Activation (JITA) strategy on top of the DHS policy [5]. The JITA strategy is to wake up the next sequential line, exploiting the sequential nature of code. However, this is not successful when a taken branch is encountered. The *DHS-Bank-PA* policy [5] issues the global turn-off signal at fixed periods, when the execution shifts to a new bank, or when a new loop hotspot is detected. It attempts to identify both spatial and temporal locality changes. It also employs hotspot detection to protect active cache lines and the JITA policy for predictive cache line wakeup. As shown in [5], although the DHS-Bank-PA reduced leakage energy significantly, performance degradation is severe.

The super-drowsy cache deploys the noaccess-JITA policy with as large as a 32K-cycle update window size for next cache line prediction to achieve high accuracy [9]. The noaccess-JITA puts only lines that have not been accessed during a fixed time period into drowsy mode and activates the first sequential cache line. The super-drowsy cache also deploys an additional *NTSBP* (Next Target Sub-Bank Predictor) that predicts next sub-bank to be bitline precharged in advance, since the on-demand gated precharge incurs extra penalty to enable an inactive sub-bank, and this can result in significant execution time increase. The noaccess-JITA/NTSBP with 32K cycle update window size is a leakage energy reduction policy with the most accurate wakeup prediction but with modest leakage energy reduction. However, the accuracy of the noaccess-JITA/NTSBP is so dependent on program behavior, especially locality, that the accuracy of no-access-JITA/NTSBP is poor in some applications.

3 Novel Wakeup Prediction Policy: Utilizing Branch Prediction Information

In previous wakeup prediction policies, additional predictors are required in order to wake up a cache line, and accessed cache lines remain active for a fixed time period. Accordingly, the accuracy of the previous policies is highly dependent on the locality. As shown in Figure 1(a), the additional predictors, such as JITA [5], NSPB [8],

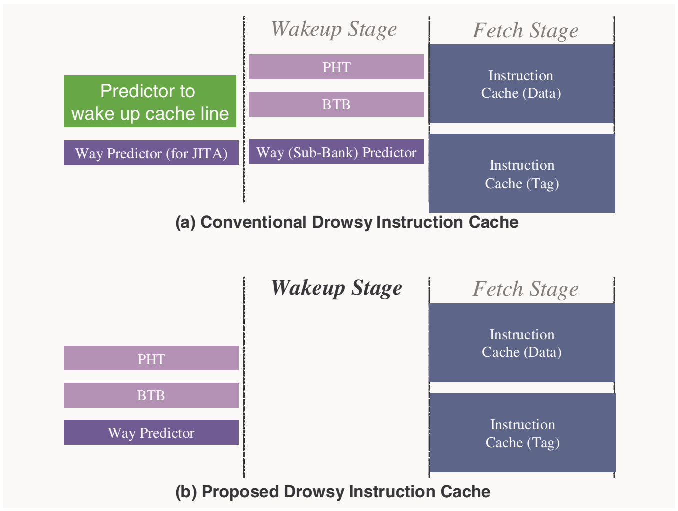


Fig. 1. Pipeline stage comparison

NSPCT [8] and NTSBP [9], are accessed before looking up the branch predictor in order to hide the wakeup penalty. However, the accuracy of additional predictors was not satisfactory. For near-optimal leakage energy reduction and performance, we propose a new wakeup prediction policy which enables on-demand wakeup. In the proposed policy, as shown in Figure 1(b), the branch predictor, consisting of Prediction History Table (PHT) and Branch Target Buffer (BTB), is accessed one cycle earlier than in conventional policies.

There are two architectural options in branch resolution. When a branch turns out to be mispredicted in the execution stage, some time is usually required to clean up mis-speculated state and generate the next address (Figure 2(a)), but depending on exactly where during the branch-resolution cycle the misprediction is detected, it may be possible to complete this without any extra overhead (Figure 3(a)). Requiring at least one cycle for cleanup and fetch-address generation, as shown in Figure 2 (a), appears to be common [22].

- **Additional penalty for recovery after the execution stage**

As shown in Figure 2 (b), after the execution/branch-resolution stage of the instructtion *n*, cleanup, effective address calculation, and wakeup occur simultaneously. Thus there is always only one active cache line.

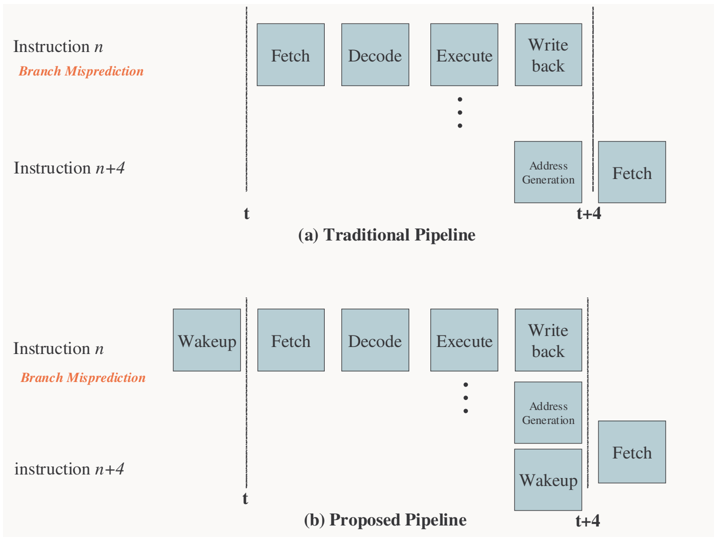


Fig. 2. Pipeline structure (when there is one-cycle penalty for effective address calculation)

- **No penalty for recovery after the execution stage**

In Figure 3 (b), it is impossible to wake up only one correct cache line after a misprediction without incurring a one-stage penalty, because cleanup and address generation occur in the same stage as misprediction detection. Instead, the potential alternative path should be woken up speculatively in parallel with branch resolution. This means that during some cycles, two lines are awake.

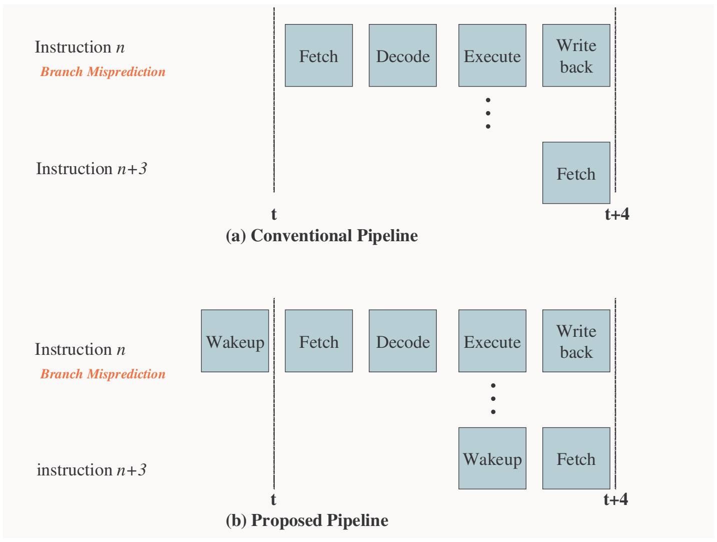


Fig. 3. Pipeline structure (when there is no penalty for effective address calculation)

It is possible to determine the alternative path in parallel with branch resolution. For predicted-taken branches, the not-taken path must be woken up and the branch address itself (usually carried with the instruction) can be used. For predicted not-taken branches, the taken target is needed. This can either be carried with the instruction or reside in some dedicated storage. This capability must exist anyway in current microprocessors because every taken branch in flight must be able to check whether the target address obtained from the BTB is correct or not. Note that the taken target is available at branch-prediction time regardless of predicted direction, because the direction predictor and target predictor are usually consulted in parallel.

In both options of Figure 2 and Figure 3, there is no penalty when there is no branch misprediction. In case of branch target misprediction, the penalty is inevitable. However, there is only one case that we can not hide the penalty in case of branch direction misprediction. Since the stored cache line address woken up is not that of (mispredicted branch instruction address + 4), but the mispredicted branch instruction address itself, there is a penalty when the resolved branch instruction is at the end of the cache line and the correct next instruction is sequential. It is possible to make use of the instruction address +4, but it requires extra adder or storage for the instruction address + 4. Even though this cost may be minor, in this paper we do not use an extra adder or extra storage, since the probability that a mispredicted instruction is at the end of the cache line is rare.

In the proposed policy, only one cache line (or two cache lines in Figure 3) expected to be accessed exists in the active mode and all the other cache lines are in the drowsy mode. For a set-associative cache, only one way should woken up to save the energy. We adopt a way predictor [18] that employs MRU (Most Recently Used) bit and integrates a way predictor and a BTB for high accuracy, which is known as one of the most accurate way predictors. In the noaccess-JITA/NTSBP, the way predictor is used for cache line wakeup prediction, while for NTSBP it is used for precharging and way prediction of cache line to be read. When the way predictor can have 2-read ports in order to predict the next cache line that will be actually read as well, the prediction accuracy for precharging is higher and the NTSBP is unnecessary (In this paper, we call this policy as *Noaccess-JITA utilizing w.p. (Way Predictor)*). Both options (noaccess-JITA/NTSBP and noaccess-JITA (utilizing w.p.) are evaluated in this paper. In DHS-Bank-PA, way prediction is not required in case of actual cache read, since the whole sub-bank is put in the sleep mode when execution jumps from one sub-bank to another, resulting in overlapping of wakeup penalty and precharging penalty. In the proposed policy, the PHT and the BTB are accessed one cycle earlier, which leads to one cycle earlier way prediction. There is no need for another way prediction to read the instruction cache, since only one woken up cache line can be read in the proposed on-demand policy. In case of Figure 3, however, a two-port way predictor is required to support concurrent two accesses: one is to wake up the next cache line in case of correct branch prediction (to wake up instruction n+3, when instruction n is predicted correctly in Figure 3 (b)) and the other is to wake up a probable cache line recovered from branch misprediction (to wake up instruction n+3, when instruction n is recovered from branch misprediction in Figure 3 (b)).

4 Experimental Methodology

We extended Simplescalar 3.0 [2] to evaluate energy and performance. The processor parameters model a high-performance microprocessor similar to Alpha 21264 [7], as shown in Table 1. The power/energy parameters are based on the 70nm/1.0V

Table 1. Architecture/circuit parameters

Processor Parameters	
Branch Predictor	Gshare/4K, 1024-entry 4-way BTB
L1 I-Cache	32 KB, 4 way, 32B blocks, 1 cycle latency, 4KB sub-bank size
L1 D-Cache	32 KB, 4 ways, 32B blocks, 1 cycle latency
Power/Energy Parameters	
Process Technology	70 nm
Threshold Voltage	0.2 V
Supply Voltage	1.0 V (active mode), 0.25 V (drowsy mode)
Leakage Power/Bit in Active Mode w/o Gated Precharging (1 cycle)	0.0778 μW
Leakage Power/Bit in Active Mode w/ Gated Precharging (1 cycle)	0.0647 μW
Leakage Power/Bit in Drowsy Mode w/o Gated Precharging (1 cycle)	0.0167 μW
Leakage Power/Bit in Drowsy Mode w/ Gated Precharging (1 cycle)	0.00387 μW
Turn-on (drowsy to active) Energy/Bit	115fJ
Turn-on (drowsy to active) Latency	1 cycle
Clock Cycle Time	12 * FO4 (395ps)

technology [9]. We use all integer and floating point applications from the SPEC2000 benchmark suite [24]. Each benchmark is first fast-forwarded half a billion instructions and then simulated the next half a billion instructions.

We selected three previous prediction policies (noaccess-JITA/NTSBP, noaccess-JITA (utilizing w.p.), and DHS-Bank-PA, described in Section 2 and Section 3) for comparison. We use same details of the policies as proposed in [5][9]. The noaccess-JITA/NTSBP has a 32 K cycle update window to periodically update mode of each cache line. Although execution moves from one sub-bank to another sub-bank, the precharge circuits of the previous sub-bank remain on for 16 cycles to prevent the misprediction of sub-bank. After 16 cycles, the bitline of the sub-bank is isolated. The DHS-Bank-PA has 2 K cycle update window and its hotness threshold is 16.

5 Experimental Methodology

This section presents our simulation results and compares the proposed policy to other policies. We analyze each policy's energy reduction and execution time increases.

5.1 Drowsy Fraction and Gated Bitline Precharging Fraction

Figure 4 shows the drowsy fraction in the 4-way set-associative cache. Since the update window size of the noaccess-JITA/NTSBP is as large as 32K, the drowsy

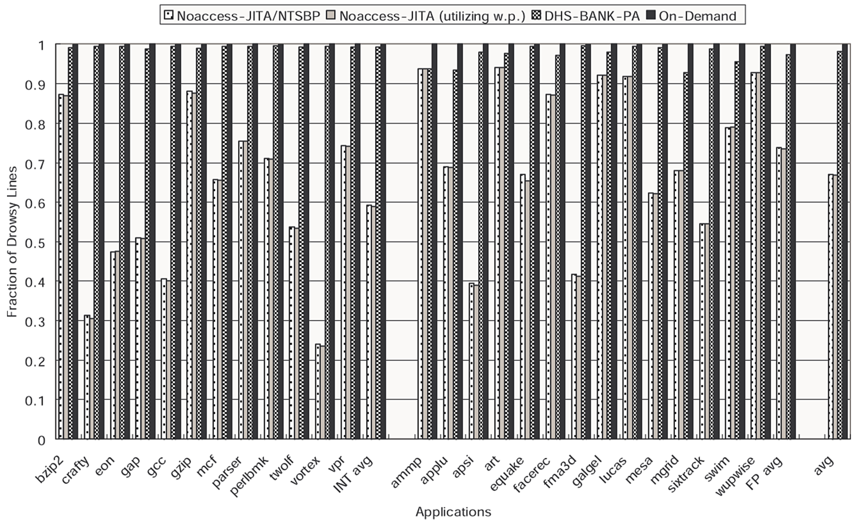


Fig. 4. Average drowsy fraction in instruction cache

fraction is 66.9%, on average. In the DHS-Bank-PA, the drowsy fraction is 98.2%, on average. The reason is that the update window size is as small as 2K and additionally cache lines are put into the drowsy mode when a new hotspot is detected. In the proposed on-demand policy, only one (or two in the proposed policy of Figure 3) cache line is in the active mode and the others are in the drowsy mode, resulting in 99.9% (or 99.8% in the proposed policy of Figure 3) drowsy fraction, on average. There is little difference between the noaccess-JITA/NTSBP and the noaccess-JITA (utilizing w.p.), since the NTSBP and the 2-read port way predictor are not related to the drowsy fraction but related to the precharging fraction.

Figure 5 shows the fraction of isolated bitines in the 4-way set associative cache. In case of bitline precharging prediction, there is no energy penalty but there is one cycle timing penalty when mispredicted. In the noaccess-JITA/NTSBP, on average 75.7% of the sub-banks are bitline gated. The fraction is relatively small, because a sub-bank should be remained bitline precharged for 16 cycles to prevent bitline precharging mispredictions when execution moves to another sub-bank. However, the noaccess-JITA (utilizing w.p.) always has 87.5% since way predictor is used for subbank prediction. In the other two techniques, only one sub-bank is bitline percharged. Thus, the portion of gated bitline precharging is always 87.5% (1 sub-bank/8 sub-banks).

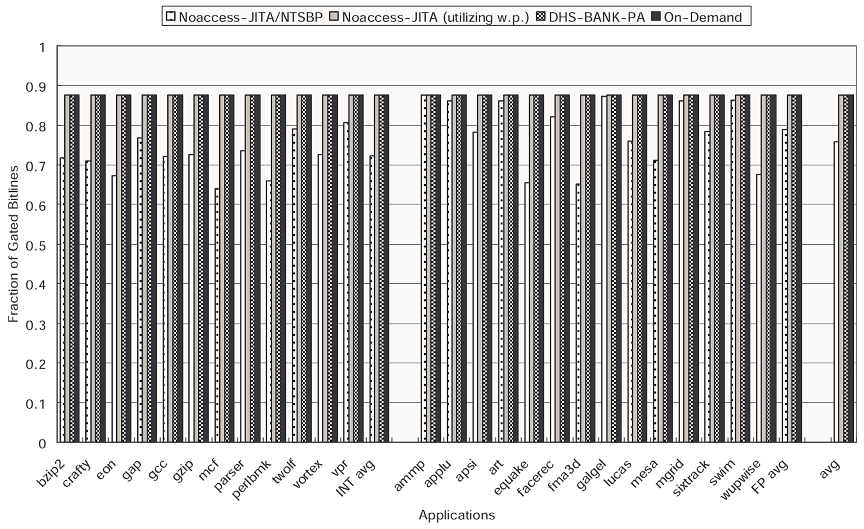
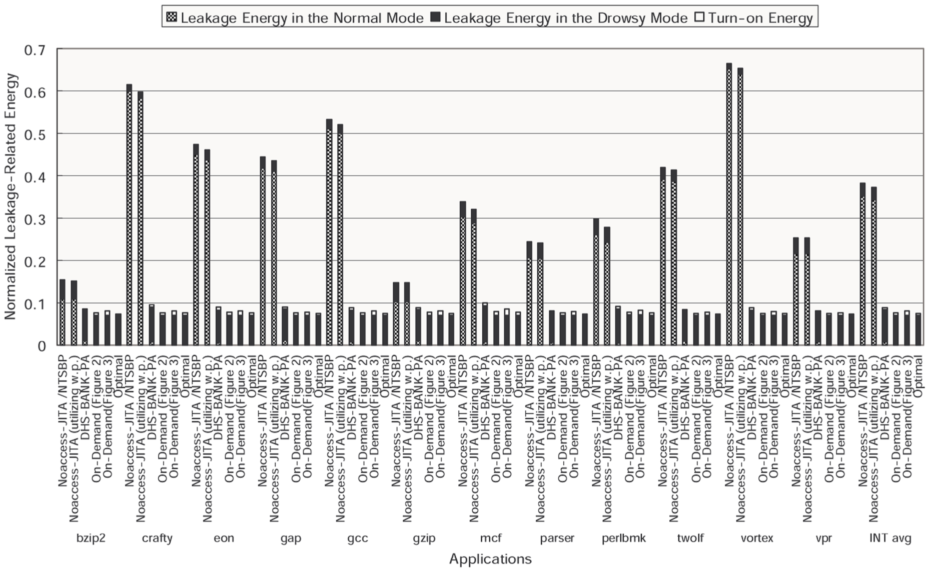


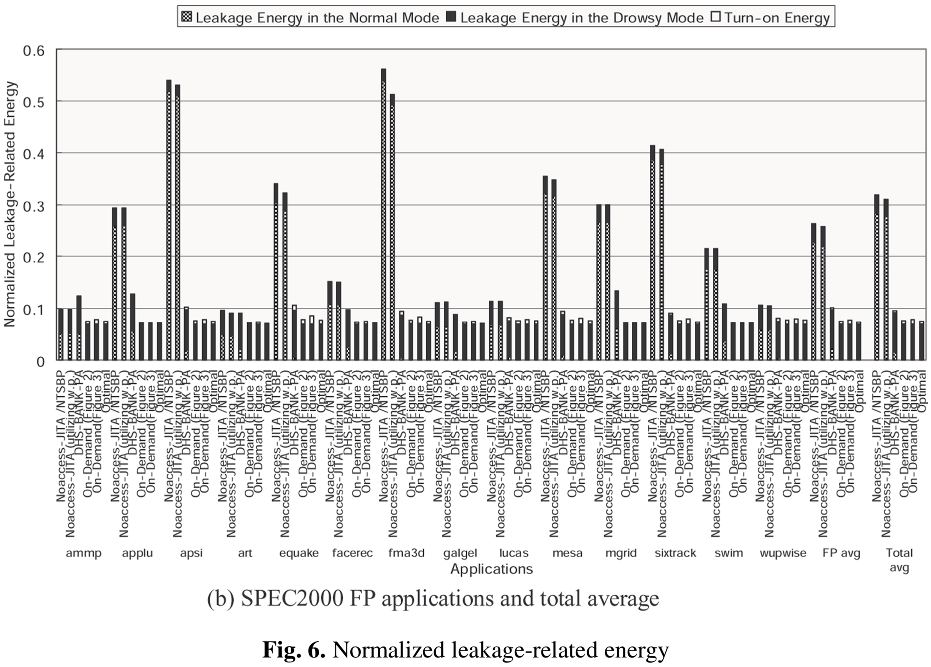
Fig. 5. Average isolated bitline fraction in instruction cache

5.2 Total Leakage-Related Energy

In the proposed policy, the next cache line is woken up on-demand. Thus, the leakage energy in the active mode is minimized, whereas turn-on energy by prediction is expected to be larger due to more frequent sleep/activation round-trips compared to the other previous policies. However, turn-on energy in the proposed policy still accounts for a small portion of total leakage-related energy. Figure 6 shows normalized leakage-related energy to the base model. The base model does not use any leakage-saving policy but it has the way predictor. Average leakage-related energy reduction is 68.1%, 69.8%, 90.4%, 92.5%, 92.2%, and 92.6% in the noaccess-JITA/NTSBP, noaccess-JITA (utilizing w.p.), DHS-Bank-PA, on-demand of Figure 2, on-demand of Figure 3, and optimal policies, respectively.



(a) SPEC2000 INT applications



(b) SPEC2000 FP applications and total average

Fig. 6. Normalized leakage-related energy

5.3 Wakeup Prediction Accuracy

On average, the branch prediction accuracy is 94.3% and the branch instruction ratio is 8.7% for SPEC applications. Recall that wakeup misprediction is mainly caused by branch misprediction by incorrect target address. As the number of branch instructions gets smaller, the branch prediction accuracy affects wakeup prediction accuracy less. For example, gcc and gzip shows similar branch prediction accuracy but the branch instruction ratio of gzip is much less than that of gcc, resulting in higher wakeup prediction accuracy of gzip in Figure 7.

As explained in Section 2.2, correct cache line prediction for drowsy cache does not always mean correct sub-bank prediction for bitline precharging in the noaccess-JITA/ NTSBP, since the cache line is predicted by noaccess-JITA and the sub-bank is predicted by NTSBP (In other words, cache lines in the active mode are spread across sub-banks). The same is applied to the noaccess-JITA (utilizing w.p.) in the set-associative cache. In the other policies, cache lines in the active mode are in one sub-bank.

Figure 7 shows the wakeup prediction accuracy, including bitline precharging and way prediction accuracy in the 4-way set-associative cache. The accuracy of the optimal policy implies the way prediction accuracy. Please note that the results are not per instruction but per fetch. Average accuracy of the noaccess-JITA/NTSBP is 71.9% since a set-associative cache make it more difficult to predict sub-bank precharging. However, the noaccess-JITA (utilizing w.p.) and the proposed on-demand policy shows 87.3% and 87.6% accuracy, respectively which is close to the accuracy (way prediction accuracy) of the optimal policy. The accuracy of DHS-Bank-PA is as low as 57.6%, on average, which might result in severe performance degradation. This is caused by flushing the previous sub-bank when execution jumps

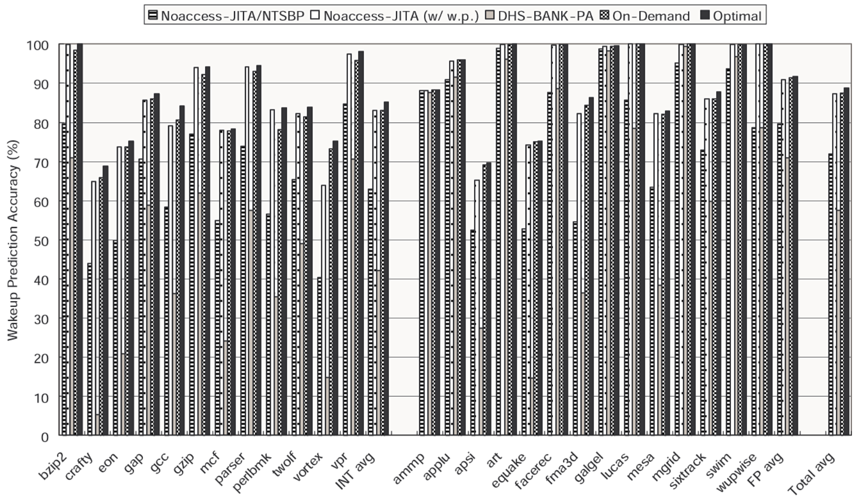


Fig. 7. Wakeup prediction accuracy per fetch, Including bitline precharging and way prediction accuracy

from one sub-bank to another, since the sub-bank hoppings are much more frequent in a set-associative cache.

5.4 Execution Time

Even one percent increase of execution time leads to substantial increase of the total processor energy, which might counterbalance the reduced L1 instruction cache leakage. Thus, it is crucial to maintain execution time close to the base model. We only show the proposed policy of Figure 2, since there is negligible difference from that of Figure 3.

When a wakeup misprediction (including precharging misprediction and way misprediction) and an instruction cache miss occur at the same time, the wakeup penalty is hidden by the cache miss penalty. Thus, the wakeup prediction accuracy is related to the execution time but this is not always exactly proportional.

Figure 8 shows the execution time normalized to the base model in the 4-way set-associative cache. The increases of execution time are 2.09%, 0.15%, 5.36%, and 0.27% for noaccess-JITA/NTSBP, noaccess-JITA (utilizing w.p.), DHS-Bank-PA, and the proposed on-demand policy. Though the noaccess-JITA/NTSBP increases the execution time by inaccurate next sub-bank prediction, the noaccess-JITA (utilizing w.p.) does not since it utilizes the 2-read port way predictor which is more accurate than the NTSBP. In equake, The DHS-Bank-PA degrades the performance as much as 30.1%, which is too severe to be tolerated.

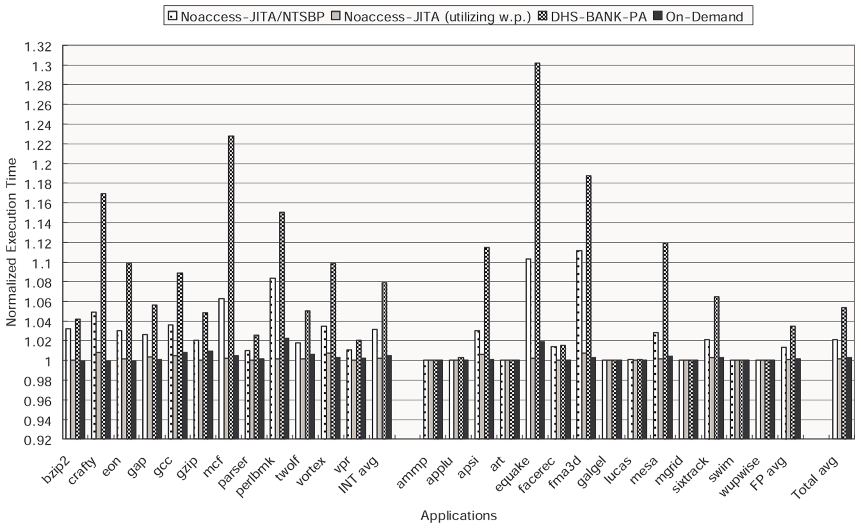


Fig. 8. Normalized execution time

5.5 Comparison of Hardware Overhead

For a wakeup prediction policy, hardware overhead is inevitable in additional to the DVS control circuitry. We compare the hardware overhead of each policy. In the noaccess-JITA/NTSBP, one bit per cache line is required in order to detect whether the cache line is accessed or not in the fixed time period. In addition, the NTSBP has 1K entries (3 bits/entry). The noaccess-JITA (utilizing w.p.) requires one bit per cache line same as the noaccess-JITA. In addition, it needs the 2-read port way predictor for bitline precharging (sub-bank) prediction. In the DHS-Bank-PA, one bit per cache line is also required to store the access history. Additionally, ten bits (half for the target basic block counter and the other half for the fall-through basic block counter) are required to locate a hotspot [5]. Since the BTB has 1024 entries, the total storage overhead is 10K. For the proposed policy, only a small register (ex. 10 bit for our 1024-entry cache) is needed to record the most recently accessed cache line.

6 Conclusions

In this paper, we propose an on-demand wakeup prediction policy using the branch prediction information. Our goal is not only less energy consumption but also consistent near-optimal performance. The noaccess-JITA/NTSBP and the noaccess-JITA (w/ w.p.) show competitive performance consistently but their energy consumption is more than four times of the proposed policy, on average. The DHS-Bank-PA reduces leakage-related energy significantly but it increases the execution time by more than 10% in many cases. In several cases, the increase is more than 20%. The proposed policy degrades the performance by only 0.27%, on average, and 2.1% for the worst case. At the same time, leakage energy is almost eliminated since only one (or two) cache line is active while all the other lines are in the drowsy mode. This is especially beneficial for controlling leakage in future instruction caches which might be much larger. The leakage energy reduction by the proposed policy is on average 92.2~92.5%, almost identical to the reduction by the optimal policy (92.6%). Therefore, we conclude that the proposed on-demand wakeup prediction policy is near-optimal.

We believe that there is no reason to try to reduce remaining leakage by adopting non-state-preserving techniques, at the risk of severe performance degradation. The proposed policy can be adopted for other state-preserving leakage saving circuit techniques as long as the wakeup penalty is at most one cycle.

Acknowledgments. This work was funded by National Science Foundation under grant no. CAREER CCR-0133634, Army Research Office under grant no. W911NF-04-1-0288, a grant from Intel MRL and the IT National Scholarship Program from IITA & MIC, Korea.

References

1. A. Agarwal, L. Hai, and K. Roy. A Single-Vt Low-Leakage Gated-Ground Cache for Deep Submicron. IEEE Journal of Solid-State Circuits. Vol. 38, Feb, 2003, pp. 319-328.
2. T. Austin, E. Larson, and D. Ernst. Simplescalar: An Infrastructure for Computer System Modeling. IEEE Computer Magazine. vol. 35, 2002, pp. 59-67.
3. K. Flautner, N. S. Kim, S. Martin, D. Blaauw, T. Mudge. Drowsy Caches : Simple Techniques for Reducing Leakage Power. Proc. of Int. Symp. on Computer Architecture, 2002, pp. 148-157.

4. F. Hamzaoglu, Y. Ye, A. Keshavarzi, K. Zhang, S. Narendra, S. Borkar, M. Stan, and V. De. Analysis of Dual-VT SRAM cells with Full-Swing Single-Ended Bit Line Sensing for On-Chip Cache. IEEE Transaction on VLSI Systems, vol. 10, April 2002, pp. 91-95.
5. J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, M. Kandemir. Exploiting Program Hotspots and Code Sequentiality for Instruction Caches Leakage Management. Proc. of Int. Symp. on Low Power Electronics and Design, 2003, pp. 593-601.
6. S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. Proc. of Int. Symp. on Computer Architecture, 2001, pp 240-251.
7. R. Kessler. The Alpha 21264 Microprocessor. IEEE Micro Magazine. 1999, pp.24-36.
8. N. S. Kim, K Flautner, D. Blaauw, and T. Mudge. Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power. IEEE Transaction on VLSI Systems, vol.12, no. 2, Feb. 2004, pp 167-184.
9. N. S. Kim, K. Flautner, D. Blaauw, T. Mudge. Single-Vdd and Single-Vt Super-Drowsy Techniques for Low-Leakage High-Performance Instruction Caches, Proc. of Int. Symp. on Low Power Electronics and Design, 2004, pp.54-57.
10. L. Li, V. Degalahal, N. Vojaykrishnan, M. Kandemir, and M. J. Irwin. Soft Error and Energy Consumption Interactions: A Data Cache Perspective. Proc. of Int. Symp. on Low Power Electronics and Design, 2004, pp. 132-137.
11. L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin and A. Sivasubramaniam. Leakage Energy Management in Cache Hierarchies. Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques, 2002, pp.131-140.
12. Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron. State-Preserving vs. Non-State-Preserving Leakage Control in Caches. Proc. of the Design Automation and Test in Europe Conference. 2004, pp. 22-27.
13. S. Manne, A. Klauser, and D. Grunwald, Pipeline Gating : Speculation Control for Energy Reduction. Proc. of Int. Symp. on Computer Architecture, 1998, pp.132-141.
14. S. McFaring. Combining Branch Predictors. Technical Note TN-36. DEC June 1993.
15. K. Nii et. al. A Low Power SRAM Using Auto-Backgate-Controlled MT-CMOS. Proc. of Int. Symp. on Low Power Electronics and Design, 1998, pp. 293-298.
16. M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd : A circuit technique to reduce leakage in deep-submicron cache memories. Proc. of Int. Symp. on Low Power Electronics and Design, 2000, pp 90-95.
17. G. Reinman and B. Calder. Using a Serial Cache for Energy Efficient Instruction Fetching. Journal of Systems Architecture. vol. 50 , issue 11, 2004, pp.675-685.
18. S. Yang and B. Falsafi. Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches. Proc. of Int. Symp. on Microarchitecture, 2003.
19. S. Yang, M. Powell, B. Falsafi, K. Roy, and T. Vijaykumar. An Integrated Circuit/ Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. Proc. of Int. Symp. on High-Performance Computer Architecture, 2001, pp.147-157.
20. W. Zhang, J. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Compiler-Directed Instruction Cache Leakage Optimization. Proc. of Int. Symp. on Microarchitecture, 2002, pp.208-218.
21. ARM. ARM 1136 Technical Reference Manual. Available in http://www.arm.com
22. ITRS (International Technology Roadmap for Semiconductor). Available in http:// public.itrs.net.
23. Standard Performance Evaluation Corp.. Available in http://www.specbench.org.