On-Demand Solution to Minimize I-Cache Leakage Energy with Maintaining Performance

Sung Woo Chung, *Member*, *IEEE Computer Society*, and Kevin Skadron, *Senior Member*, *IEEE*

Abstract—This paper describes a new *on-demand* wake-up prediction policy for reducing leakage power. The key insight is that branch prediction can be used to selectively wake up only the needed cache line. This achieves better leakage savings than the best prior policies while avoiding the performance overheads of those policies, without needing an extra prediction structure. The proposed policy reduces leakage energy by 92.7 percent with only 0.08 percent performance overhead on average. The branch-prediction-based approach requires an extra pipeline stage for wake up, which adds to the branch misprediction penalty. Fortunately, this cost is mitigated because the extra wake-up stage is overlapped with misprediction recovery. This paper assumes the superdrowsy leakage control technique using reduced supply voltage because it is well suited to the instruction cache's criticality. However, the proposed policy can be also applied to other leakage-saving circuit techniques.

Index Terms—Microprocessor, instruction cache, leakage, branch predictor, wake-up policy.

1 Introduction

Power dissipation has emerged as a major concern both for high-end processors and for embedded processors since higher power incurs higher packaging, power delivery, and cooling costs. Recently, power dissipations have become high enough to cause serious thermal challenges, possibly even resulting in a project cancellation [31]. As process technology scales down, leakage energy accounts for a significant part of the total energy. The 2001 International Technology Roadmap for Semiconductor [29] predicts that, for the 70 nm technology, leakage may constitute as much as 50 percent of the total energy dissipation. In particular, the leakage energy for on-chip caches is crucial since they comprise a large portion of the chip area. For instance, 30 percent of the Alpha 21264 and 60 percent of the StrongARM are devoted to cache and memory structures [15]. However, cache size cannot 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: Adaptive-reverse Body Biasing (ABB) MTCMOS [19], Data-Retention Gated-ground (DRG) [1], Gated-Vdd [20], and DVS for Vdd (which is also called drowsy cache) [4]. In the ABB MTCMOS technique, the threshold voltage is dynamically changed, but the wake-up penalty between the active mode and the leakage saving mode is long, making it difficult to use in L1 caches [5]. DRG retains the data while reducing leakage by gating ground and using the remaining leakage to operate the cells in subthreshold mode and to retain cell contents. It is promising for less timing-critical caches such as L2 [12], but, again, the wake-up penalty is long, making it difficult to use in L1 caches. 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. Although this technique dramatically reduces the leakage, its main disadvantage is that it does not preserve the state of the data in the sleep mode [5]. If the line is put to sleep prematurely and is needed later, it must be refetched from a lower level memory, which leads not only to additional dynamic energy consumption but also to performance degradation [7]. To prevent these costs, conservative prediction policies must be employed [26], [27]. Gated-Vdd may, however, be suitable for some L1 data caches where the refetch penalty is short [13]. Another leakage saving technique is to lower the supply voltage to a level near the threshold voltage. In this technique, data is not lost when the cache line is in the leakage saving mode (called the "drowsy" mode). In the drowsy mode, data is retained, although it cannot be accessed for a 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 the drowsy mode and keeping frequently accessed cache lines in the active mode, much power leakage is reduced without significant performance degradation. There is a wake-up penalty to restore the voltage level of the Vdd from the drowsy mode into the active mode. However, the wake-up penalty is expected to be one cycle in the 70 nm process technology [4].

Among the above four techniques, the drowsy technique appears most suitable for L1 instruction caches since it retains data and has a short wake-up penalty. In order to alleviate the wake-up penalty of the drowsy cache, many

- *S.W. Chung is with the Division of Computer and Communication Engineering, Korea University, Anam-Dong, Seongbuk-Gu, Seoul 136-713, Korea. E-mail: swchung@korea.ac.kr.*
- *K. Skadron is with the Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 151 Engineer's Way, PO Box 400740, Charlottesville, VA 22904-4740. E-mail: skadron@cs.virginia.edu.*

Manuscript received 17 Mar. 2006; revised 7 Jan. 2007; accepted 13 June 2007; published online 18 July 2007.
Recommended for acceptance by M. Dubois.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0105-0306.
Digital Object Identifier no. 10.1109/TC.2007.70770.

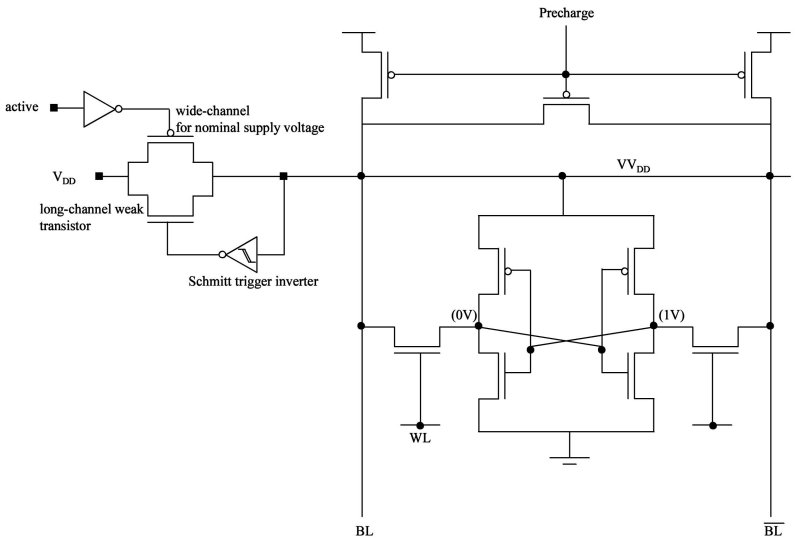

Fig. 1. Implementation of the superdrowsy cache line.

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 the Vdd of the accessed cache lines, suffering the wake-up 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 the extra latency from waking up lines [4]. For instruction caches, however, this "no-prediction" technique does not perform well because any wake-up 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 and negligible performance degradation. In this paper, we propose and evaluate a new *on-demand* wake-up prediction policy [3] for an instruction cache. By on-demand, we mean that *only the cache lines currently in use need to be awake*. This technique takes advantage of the fact that we can accurately predict the next cache line by using the branch predictor. Good wake-up prediction accuracy is therefore achieved using branch predictors, which have already proven highly accurate [16]. A further advantage compared to previous policies is that the proposed policy does not require an additional predictor. To utilize the branch predictor for wake-up prediction, we can allow a new pipeline stage between branch prediction and instruction cache fetch. On most branch mispredictions, the extra wake-up stage is overlapped with misprediction recovery, so performance is only affected on target-address mispredictions. Note that the extra stage does not affect branch predictor accuracy. For further details, see Section 3.

This work focuses on the use of the drowsy mode (actually, the superdrowsy mode [10], explained in Section 2) as the leakage-saving circuit technique. In this paper, we distinguish the wake-up prediction *policy* from the leakage-saving *circuit technique*. The wake-up prediction policy predicts which cache line will be woken up, whereas the leakage-saving circuit technique is the mechanism for putting lines to sleep and waking them up, independent of the prediction policy. Although not evaluated here, the on-demand policy would work well with other leakage-saving techniques.

The rest of this paper is organized as follows: Section 2 explains the concept of the drowsy/superdrowsy cache, the reliability of the drowsy instruction cache, and previously proposed prediction policies. Section 3 proposes a new on-demand wake-up prediction policy by using branch prediction information. Section 4 presents the analytical model for evaluation and simulation environments. Section 5 evaluates the energy/performance for the proposed policy. Finally, Section 6 concludes the paper.

2 Background Work

2.1 Drowsy/Superdrowsy Cache Circuit Technique

The drowsy cache technique [4] has received a great deal of attention because it retains data while providing a short wake-up penalty. When the cache line is not expected to be used in the near future, the supply voltage of the cache line is reduced to a lower value typically close to the threshold voltage, leading to lower leakage power. In the active mode with nominal voltage, the cache line operates the same as in a conventional cache. In the drowsy mode, however, the cache line cannot be accessed even though data is retained. After being awakened, the cache line can be accessed. Since the drowsy mode does not fully turn off the supply voltage, the drowsy cache does not reduce the leakage power as much as the gated-Vdd, but data retention implies that the drowsy cache does not need to refetch instructions and allows a more aggressive sleep policy. Moreover, the wake-up penalty is short: One cycle is expected in the 70 nm technology [4].

Kim et al. proposed a refinement of this technique, called the *superdrowsy cache* [10]. As shown in Fig. 1 [10], a single-Vdd cache line voltage controller with a Schmitt trigger inverter replaces multiple supply voltage sources in order to alleviate the interconnect routing space. In addition, the on-demand gated bitline precharge technique [25] is

employed to reduce the bitline leakage. We apply our prediction policy to the superdrowsy cache because it is the most advanced circuit technique for instruction cache leakage control as far as we know.

2.2 Reliability of the Drowsy Instruction Cache

There has been concern that the drowsy cache is more susceptible to soft errors than conventional caches [11]. Some researchers are reluctant to adopt the drowsy (or superdrowsy) cache circuit technique since the error rate is exponentially dependent on the decrease of the supply voltage. Without a special error detection/correction technique, the reliability of the drowsy cache is lower than that of a conventional cache. However, this is true only in the case of write-back data caches, where even single-bit soft errors in the modified data of the data cache are critical. Instruction caches usually are not write-back. Errors can be detected using per-line parity and faulty lines can be refetched [24]. Double-bit errors are relatively rare: Li et al. reported one order of magnitude difference between the rates of single and double-bit errors [11]. Parity protection is implemented in any case and the overhead is fairly small.

2.3 Previous Wake-Up Prediction Policies

The success of the drowsy-style cache depends on how accurately the next cache line can be predicted and awakened. Especially for an instruction cache, accuracy is crucial since the accuracy directly affects performance degradation. A simple policy is *noaccess* [4]: This uses per-line access history and puts all of the unused lines into the drowsy mode periodically. For a more accurate wake-up prediction, two prediction policies were proposed for a drowsy instruction cache [9]—Next Subcache Prediction Buffer (*NSPB*) and Next Subcache Predictor in Cache Tags (*NSPCT*). 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 percent). However, leakage reduction is weak [9] due to large subbank turn-on energy. Zhang et al. proposed the *Loop* policy [27], where all cache lines are put into the drowsy mode after each loop is executed. This bears some similarity to the Dynamic HotSpot Based Leakage Management (*DHS*) policy, which was proposed in [6]. 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-Per Access (*DHS-PA*) policy employs a Just-In-Time-Activation (JITA) strategy on top of the DHS policy [6]. The JITA strategy is to wake up the next sequential line, exploiting the sequential nature of the code. However, this is not successful when a taken branch is encountered. The *DHS-Bank-PA* policy [6] 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 wake up. As shown in [6], although the DHS-Bank-PA reduced leakage energy significantly, the average performance degradation is as much as 2.3 percent, which incurs extra dynamic and leakage power.

The conventional superdrowsy cache deploys the noaccess-JITA policy and, in order to achieve high accuracy [10], up to a 32,000-cycle update window for next cache line prediction. The noaccess-JITA puts only lines that have not been accessed during a fixed time period into the drowsy mode and activates the first sequential cache line. The superdrowsy cache also deploys an additional Next Target SubBank Predictor (*NTSBP*) that predicts the next subbank whose bitlines are to be precharged in advance since the on-demand gated precharge incurs an extra penalty to enable an inactive subbank and this can result in a significant runtime increase. The noaccess-JITA/NTSBP with a 32,000 cycle update window size is a leakage energy reduction policy with the most accurate wake-up 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 noaccess-JITA/NTSBP is poor in some applications. By slightly modifying noaccess-JITA/BTSBP, wake-up prediction can be adopted for more accuracy. When the way predictor can have two read ports in order to predict the next cache line that will be awakened as well, the prediction accuracy is higher and the NTSBP is unnecessary (in this paper, we call this policy *noaccess-JITA utilizing way predictor*).

In this paper, we compare the proposed on-demand policy to noaccess-JITA (utilizing way predictor) and DHS-Bank-PA since the former is the most accurate and the latter is known to reduce leakage most. For a fair comparison, we apply the gated bitline precharging technique to all of the policies. All of these policies are explained in Table 1 to prevent possible confusion.

3 Novel Wake-Up Prediction Policy: Utilizing Branch Prediction Information

In conventional drowsy (including superdrowsy) instruction caches, branch predictors are only used for conventional branch direction/target prediction. In previous wake-up prediction policies for leakage savings, additional predictor structures are required in order to wake up a cache line and accessed cache lines usually remain active for a long time. The accuracy of the previous policies is therefore highly dependent on the locality and the portion of active leakage is still large. As shown in Fig. 2a, additional predictors such as JITA [6], NSPB [9], NSPCT [9], and NTSBP [10] are accessed before looking up the branch predictor in order to hide the wake-up penalty. The accuracy of these predictors has generally been problematic.

Instead, for near-optimal leakage energy reduction and performance, we propose a new wake-up prediction policy that enables on-demand wake up. The key is that the branch predictor already identifies which line needs to be awakened. No additional wake-up-prediction structure is needed. In the proposed policy, as shown in Fig. 2b, the instruction cache is therefore accessed one or more cycles

TABLE 1
Prediction Policies, Including the Policies for Comparison

Policy	Description
Noaccess	All the unused cache line is put into drowsy mode periodically by referencing to per-line access history.
NSPB (Next Subcache Prediction Buffer)	Additional predictor to wake up the next subbank
NSPCT (Next Subcache Predictor in Cache Tags)	Additional bits in cache tags to wake up the next subbank
NTSBP (Next Target Sub-Bank Predictor)	Additional predictor to precharge the bitline of a sub-bank
Loop	All cache lines are put into the drowsy mode after each loop was executed
DHS (Dynamic HotSpot based leakage management)	All cache lines are put into the drowsy mode when a new loop-based hotspot is detected
DHS-PA (Per Access)	In addition to DHS, next cache line activation is supported
Noaccess-JITA (Just In Time Activation)/NTSBP	For a drowsy cache, Noaccess is used, while activating sequentially next cache line. For bitline precharging prediction, NTSBP is used
Noaccess-JITA (utilizing w.p.)	For a drowsy cache, Noaccess is used, while activating sequentially next cache line. For wakeup prediction as well as precharging and way prediction, 2-read port way predictor is used instead of NTSBP.
DHS-Bank-PA	For a drowsy cache, DHS-Bank is used while activating sequentially next cache line. Automatically DHS-Bank-PA predicts which bitlines of a sub-bank to be precharged

later in order to provide time to wake up the line identified by the branch predictor. The data part and tag part of the instruction cache are accessed at the same time and the leakage-saving technique can be applied to both. During the wake-up stage, the next branch prediction is made. It is important to point out that the critical path of the branch predictor is unaffected and this approach has no impact on branch predictor accuracy.

The performance cost of the on-demand policy is therefore limited to control-flow mispredictions. For the case of direction misprediction (with correct target prediction), there are two wake-up prediction policies (Figs. 3b and 4b), depending on the architectural options in the branch resolution. When a branch turns out to be mispredicted in the execution stage, some time is usually required to clean up the misspeculated state and generate the next fetch address (Fig. 3a), 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, for example, Fig. 4a. This makes the impact of the wake-up penalty essentially negligible. As we will show later, in this case, performance degradation is mostly caused by target address misprediction since there is no way to identify a mispredicted target address before the actual address calculation.

Additional penalty for recovery after the execution stage. In some pipeline designs, recovering from a branch misprediction may require an extra clock cycle. As shown in

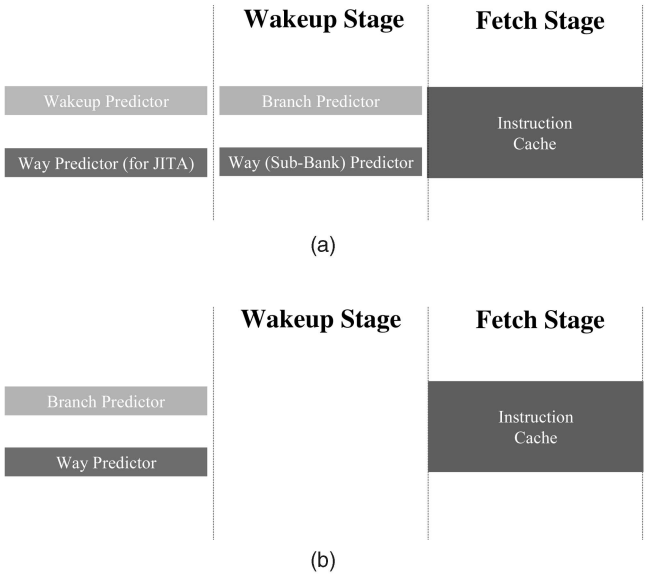

Fig. 2. Pipeline stage comparison. (a) Conventional drowsy instruction cache. (b) Proposed drowsy instruction cache.

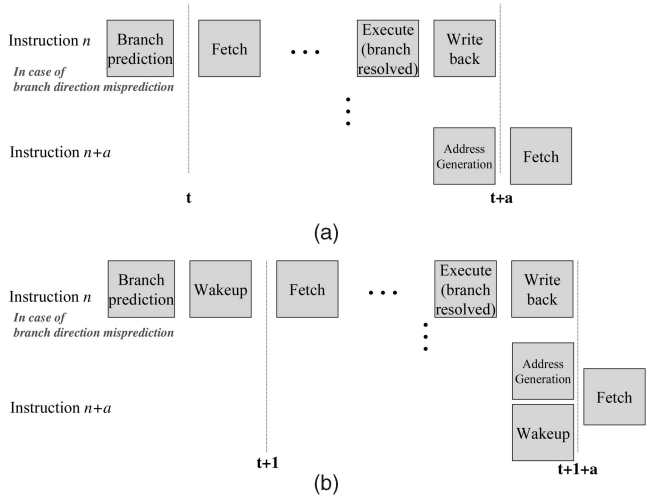

Fig. 3. Pipeline structure (when there is a one-cycle penalty for effective address calculation). "a" represents the branch-resolution latency. (a) Conventional pipeline. (b) Proposed pipeline.

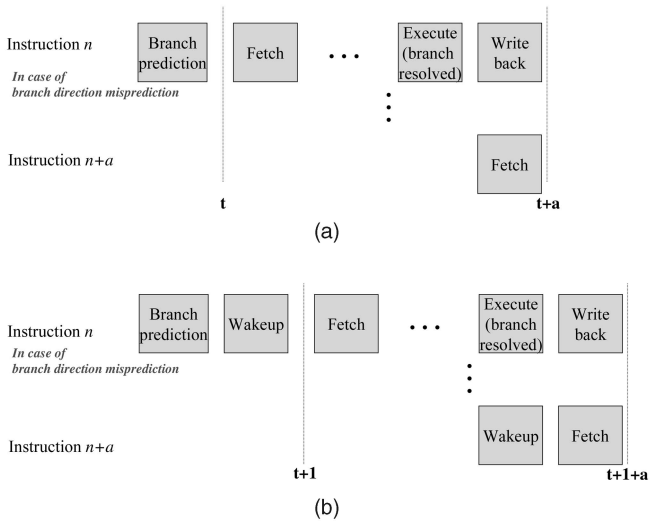

Fig. 4. Pipeline structure (when there is no penalty for effective address calculation). (a) Conventional pipeline. (b) Proposed pipeline.

Fig. 3, in the next pipeline stage after the execution/branch-resolution stage of some instruction n, cleanup, effective address calculation, and (in the on-demand scheme) wake up occur simultaneously. For recovering from incorrectly predicted taken branches, the branch instruction address to wake up the not-taken path (sequentially next address) is usually carried with the instruction. For recovering from incorrectly predicted not-taken branches, the target address 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 and quickly redirect fetch after a misprediction. Because the wake-up cycle is overlapped with the other misprediction-recovery tasks, the on-demand scheme does not increase the misprediction penalty.

No penalty for recovery after the execution stage. If branches are resolved sufficiently early in the branch-resolution stage to allow recovery to be completed by the end of the stage, we want to avoid introducing an extra cycle of misprediction penalty. As shown in Fig. 4, this can be achieved if branches are resolved sufficiently early to allow wake up to complete by the end of the branch-resolution stage or (the more likely case) if we are willing to speculatively wake up the alternative path. This means that sometimes two lines (current instruction path and the other path) are awake. Since branch instructions account for a small portion of the total instructions and turn-on energy is negligible, the turn-on energy of one more cache line for branch instructions is negligible. In addition, requiring at least one cycle for cleanup and fetch-address generation (Fig. 3a) appears to be common [28]. For this reason, we selected the on-demand policy in Fig. 3 for our evaluations in the remainder of this paper.

We have now established that branch direction mispredictions generally need not incur an additional misprediction penalty due to the on-demand policy. There is only one case in which the on-demand scheme encounters an additional penalty *in case of a branch direction misprediction* (as previously explained, we cannot hide the penalty *in case of a branch target misprediction*). Since the stored cache line address awakened is not that of the 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 in the sequentially next cache line. It is possible to make use of the instruction address $+4$, but it requires an extra adder or storage for the instruction address $+4$. Even though this hardware 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 not so high (for example, in the case of a 32-byte cache line and a 32-bit instruction, the probability is 12.5 percent).

In the on-demand policy, only the one cache line (or at most two cache lines in Fig. 4) expected to be accessed exists in the active mode and all the other cache lines are in the drowsy mode. After a new line is selected for wake up, the currently awake line is put to sleep. For a set-associative cache, the entire set could be awakened, but, to save energy, only one way should be awakened. We adopt a way predictor [21] that employs an MRU bit and integrates the way predictor and BTB for high accuracy, which is known as one of the most accurate way predictors [21]. For conventional drowsy instruction caches, the way predictor is also used to predict the way that will be awakened. The way predictors for conventional drowsy instruction caches only wake up the cache lines that are sequential. Non-sequential cache lines are expected to still be awake based on cache lines that were previously awakened and remain active. In the noaccess-JITA (utilizing way predictor), the way predictor can have two read ports for wake-up prediction, as well as precharging and fetching. In DHS-Bank-PA, way prediction is not required in case of an actual cache read since the whole subbank is put in sleep mode when execution jumps from one subbank to another, resulting in the overlap of the wake-up penalty and precharging penalty. In the proposed policy, the branch predictor and the way predictor are accessed simultaneously, which results in no need for another way prediction to read the instruction cache since only one awakened cache line can be read in the proposed on-demand policy. Note also that, in this paper, we apply the on-demand wake-up policy to the tag part of the instruction cache. However, when the tag part is always in the active mode, the tag access can be moved to the wake-up stage, leading to tag matching in the wake-up stage. Although this provides a 100 percent accurate way prediction without any way predictor, we take the more pessimistic approach and assume that a way predictor is needed for set-associative caches.

Fig. 5 shows one example of the proposed policy for the Fig. 3 option. After a misprediction by incorrect target address, the recovered target address (0x00182f10) is awakened in cycle n. At the same time, the branch predictor is looked up to predict the next fetch block. In the conventional pipeline, the first branch predictor access is done in cycle $\mathrm{n}+1$. There is no predicted taken branch in the fetch block (0x00182f10), leading to waking up the next

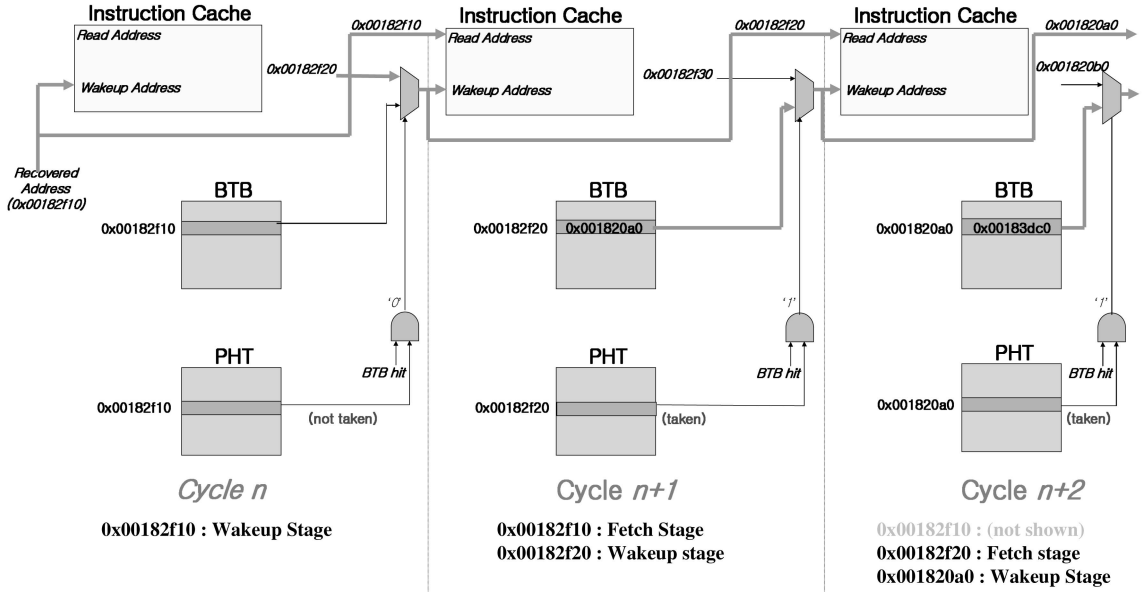

Fig. 5. Example operation of the on-demand wake-up prediction policy in which fetch resumes at address 0x0018sf10 after a misprediction.

TABLE 2
Notation Description

Notation	Description
P_i	Leakage power consumed for a cache line in the *i* mode
$P_{\text{turn-on}}$	Transition power consumed for a cache line to be changed to the active mode
AN_i	Average Number of cache lines in the *i* mode
$TN_{\text{prediction}}$	Total Number of cache lines turned on by wakeup prediction
$TN_{\text{correction}}$	Total Number of cache lines turned on for correction (by wakeup misprediction)
T	Total run-time, including the increased run-time for wakeup prediction policies

sequential fetch block (0x00182f20) in cycle $n+1$. In cycle $n+1$, the branch predictor is accessed for the block (0x00182f20), which should be accessed in cycle $n+2$ in the conventional pipeline. In this case, the fetch block (0x00182f20) has a predicted taken branch. Thus, the target address from the BTB is used for the wake-up address. Accordingly, the block (0x001820a0) is awakened in cycle $n+2$ and fetched in cycle $n+3$. Please note that the proposed policy does not affect the branch prediction accuracy.

4 Experimental Methodology

4.1 Analytical Models

Leakage-related energy includes the leakage energy in the active mode, leakage energy in the drowsy mode (including the super-drowsy mode), turn-on energy for prediction, turn-on energy for the correction of a wake-up misprediction (turn on means changing the cache line from the drowsy mode into the active mode and is due to the need to charge the capacitance of the line back to Vdd), and total extra energy expended due to any change in runtime. Equation (1) provides an expression for leakage-related energy, and Table 2 summarizes the notation used there:

$$E_{\text{leakage\_related}} = \Sigma P_i{}^{*} AN_i{}^{*} T + P_{\text{turn-on}}{}^{*} (TN_{\text{prediction}} + TN_{\text{correction}}), \quad (1)$$

where $i = \{\text{active mode}, \text{drowsy mode}, \text{bitline gating mode}, \text{super-drowsy mode (drowsy mode + bitline gating mode)}\}$.

In the base model, all of the cache lines are always in the active mode. The policies use predictions to reduce the leakage-related energy by decreasing the number of cache lines in the active mode. In the on-demand policy, all of the cache lines except the next fetch cache line are in the drowsy mode in which some of them are bitline-gated and others are not.

In order to compare the on-demand policy to the theoretically best policy, the *optimal policy* is presented in this paper. The optimal policy is assumed to have perfect knowledge of the future address trace. Thus, its performance is the same as the base model and its leakage-related energy is least. Not only minimizing the number of cache lines in the active mode but also reducing the unnecessary turn-on energy is the goal of the optimal policy. If the turn-on energy is more than the active leakage energy (in other words, if the time until the cache line is reused is short enough), it is more efficient to leave the cache line in the active mode instead of putting it in the drowsy mode. Accordingly, the optimal policy can save more energy than the on-demand policy. Using (2), extended from [17], the

TABLE 3
Architecture/Circuit Parameters

Processor Parameters	Specifications
Pipeline	Out-of-order
Fetch/Decode/Issue/Commit Width	4 instructions/cycle
Branch Predictor	Gshare/4K, 9-cycle penalty, 1024-entry 4-way BTB
Integer ALUs/Multi-divs/memory ports	4/1/2
FP ALUS/multi-divs	4/1
FU Latencies	Int : mul 3, div 20, all others 1 FP : adder 2, mul 4, div 12, sqrt 24
Memory Bus Width/Latency	4 Bytes/255 and 2 cycles for the first and inter chunks
Instruction/Data TLB	128 entry/32 entry in each way, 8KB page size, fully associative, LRU, 28-cycle latency
L1 I-Cache	32 KB, 1-/4-way, 32B blocks, 1 cycle latency, 4KB sub-bank size
L1 D-Cache	32 KB, 4 way, 32B blocks, 1 cycle latency
L2 Unified Cache	512 KB, 4 ways, 64B blocks, LRU, 12 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	0.0778 μW
Leakage Power/Bit in Active Mode w/ Gated Precharging	0.0647 μW
Leakage Power/Bit in Drowsy Mode w/o Gated Precharging	0.0167 μW
Leakage Power/Bit in Drowsy Mode w/ Gated Precharging	0.00387 μW
Turn-on (drowsy to active) Energy	115fJ
Turn-on (drowsy to active) Latency	1 cycle
Clock Cycle Time	12 * FO4 (395ps)

optimal point can be found. If the reuse interval (I_{reuse}) for a cache line satisfies the following formula, it is more energy efficient to leave the cache line in the active mode:

$$I_{reuse}{}^{*}P_{active\_mode} < I_{reuse}{}^{*}P_{drowsy\_mode} + P_{turn\text{-}on}. \quad (2)$$

4.2 Simulation Environment

We extended SimpleScalar 3.0 [2] to evaluate energy and performance. The processor parameters model a high-performance microprocessor similar to Alpha 21264 [8], as shown in Table 3. Table 3 also gives the technology and power/energy parameters used in this paper for modeling the instruction cache. The power/energy parameters are based on the 70 nm/1.0 V technology [10]. We only specifically model power in the cache and omit a detailed cycle-by-cycle whole-processor power model with Simple Scalar to make it easier to generalize conclusions about cache leakage to other processor configurations. We therefore consider whole-processor energy efficiency and the impact of extra runtime in terms of the ratio of leakage-related power in the instruction cache to total processor power.

We use all integer and floating-point applications from the SPEC2000 benchmark suite [30] and use their alpha binaries and reference inputs for execution. SimPoint [23] is used to find representative samples of program execution. Each benchmark is first fast-forwarded 300 million instructions and then simulated for the next billion instructions.

We selected gshare for a branch predictor since gshare performs fairly well and it is suspected to be used for commercial microprocessors [18]. However, if a more accurate branch predictor was selected for evaluation, the proposed policy would perform even better, whereas it would not help wake-up prediction accuracy for other policies. Using gshare instead of a more accurate branch predictor actually sets a higher bar for evaluating our proposed policy.

We selected two prediction policies (noaccess-JITA (utilizing way predictor) and DHS-Bank-PA) for comparison. We use the same details of the policies as proposed in [6], [10]. The noaccess-JITA (utilizing w.p.) has a 32 K cycle update window to periodically update the mode of each cache line. Although execution moves from one subbank to another subbank, the precharge circuits of the previous subbank remain on for 16 cycles to prevent the misprediction of the subbank. After 16 cycles, the bitline of the subbank is isolated. The DHS-Bank-PA has 2 K cycle update window and its hotness threshold (when the access count exceeds this value, the cache line is considered to be a hotspot and all other cache lines except this hotspot basic block are turned off) is 16.

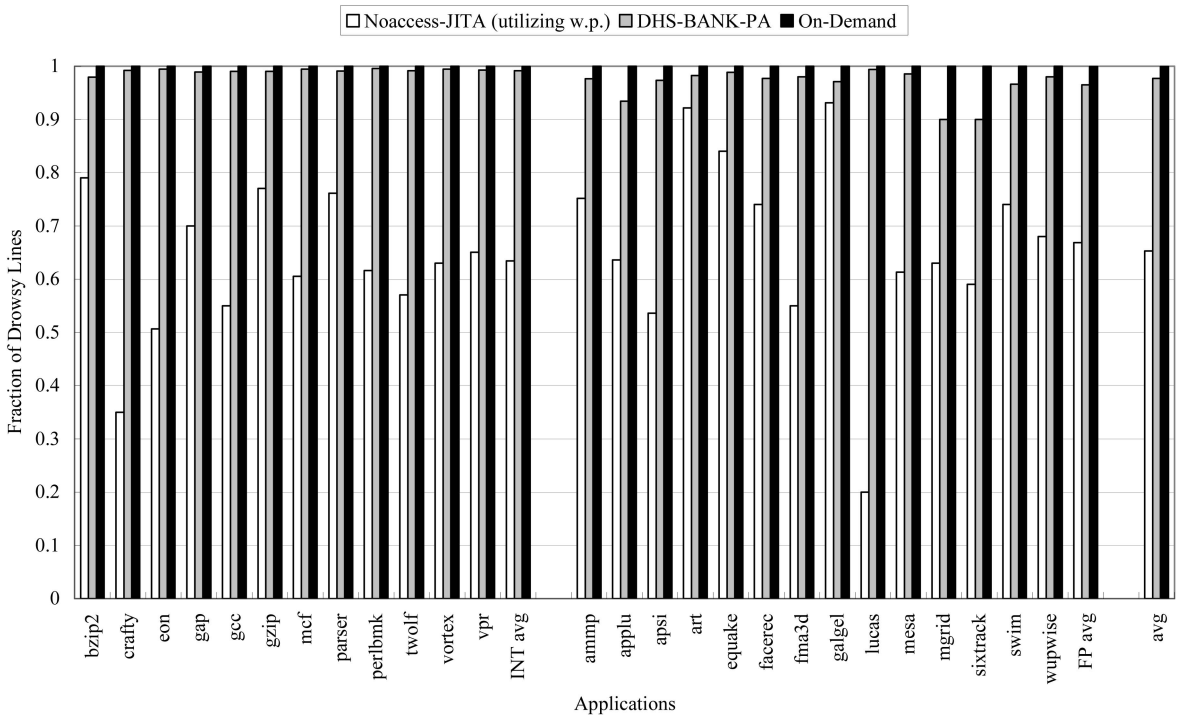

Fig. 6. Average drowsy fraction in the instruction cache (direct-mapped).

5 Simulation Results

This section presents simulation results comparing the proposed on-demand policy to other policies in terms of energy reduction and runtime impact. Then, we explore the effects of the proposed policy on the total processor energy and the policy's potential effect on other leakage-saving circuit techniques.

5.1 Drowsy Fraction and Gated Bitline Precharging Fraction

Fig. 6 shows the drowsy fraction—the percentage of cache lines in sleep mode—in the direct-mapped instruction cache. In the on-demand policy, only one cache line is in the active mode and the others are in the drowsy mode, resulting in a 99.9 percent drowsy fraction on the average. Since the update window size of the noaccess-JITA (utilizing way predictor) is as large as 32 K, the drowsy fraction is relatively small (on the average, 65.3 percent). In the DHS-Bank-PA, the average drowsy fraction is 97.7 percent. The reason is that the update window size is as small as 2 K and, additionally, cache lines are put into the drowsy mode when a new hotspot is detected.

In the case of the four-way set-associative instruction cache, the drowsy fraction of the on-demand policy is 99.9 percent, but the drowsy fraction of the noaccess-JITA (utilizing way predictor) is 69.6 percent, which is again less than that of other policies. In general, the drowsy fraction is not sensitive to the cache associativity. We do not show the average drowsy fraction for the 4-way set-associative cache since it looks almost identical to that in Fig. 6.

In case of bitline precharging prediction, there is no energy penalty, but there is a one cycle timing penalty when mispredicted. In the 4-way set-associative caches, the bitline precharging prediction is the same as the way prediction for noaccess-JITA (utilizing way predictor) and on-demand policy (or next subbank prediction for DHS-Bank-PA). Consequently, the fraction of isolated bitlines in the 4-way set-associative cache is always 87.5 percent (one subbank/eight subbanks). Since a way predictor is used for subbank prediction for the direct-mapped cache, the fraction of gated bitline precharging is always 87.5 percent (one subbank/eight subbanks), even in the direct-mapped cache.

5.2 Total Leakage-Related Energy

Figs. 7 and 8 show normalized leakage-related energy to the base model in the direct-mapped cache (as explained in Section 4.1, the base model is a conventional cache that does not perform leakage control), reflecting the increased runtime. The noaccess-JITA (utilizing way predictor) shows inconsistent and relatively small reductions depending on applications, whereas the other policies show very consistent reductions. The average leakage-related energy reduction is 70.0 percent, 90.6 percent, 92.66 percent, and 92.71 percent in the noaccess-JITA (utilizing way predictor), DHS-Bank-PA, on-demand, and optimal policies, respectively.

In the on-demand policy, the next cache line is awakened on demand. Thus, the leakage energy in the active mode is minimized, whereas the turn-on energy by prediction is expected to be larger due to more frequent sleep/activation round-trips compared to the other previous policies, such as the noaccess-JITA (utilizing way predictor) and the DHS-Bank-PA. However, the turn-on energy in the on-demand policy still accounts for a small portion of the total leakage-related energy. Consequently, the average difference in leakage-related energy between the on-demand policy and the optimal policy is only 0.05 percent. In contrast, the average leakage-related energy difference between the JITA

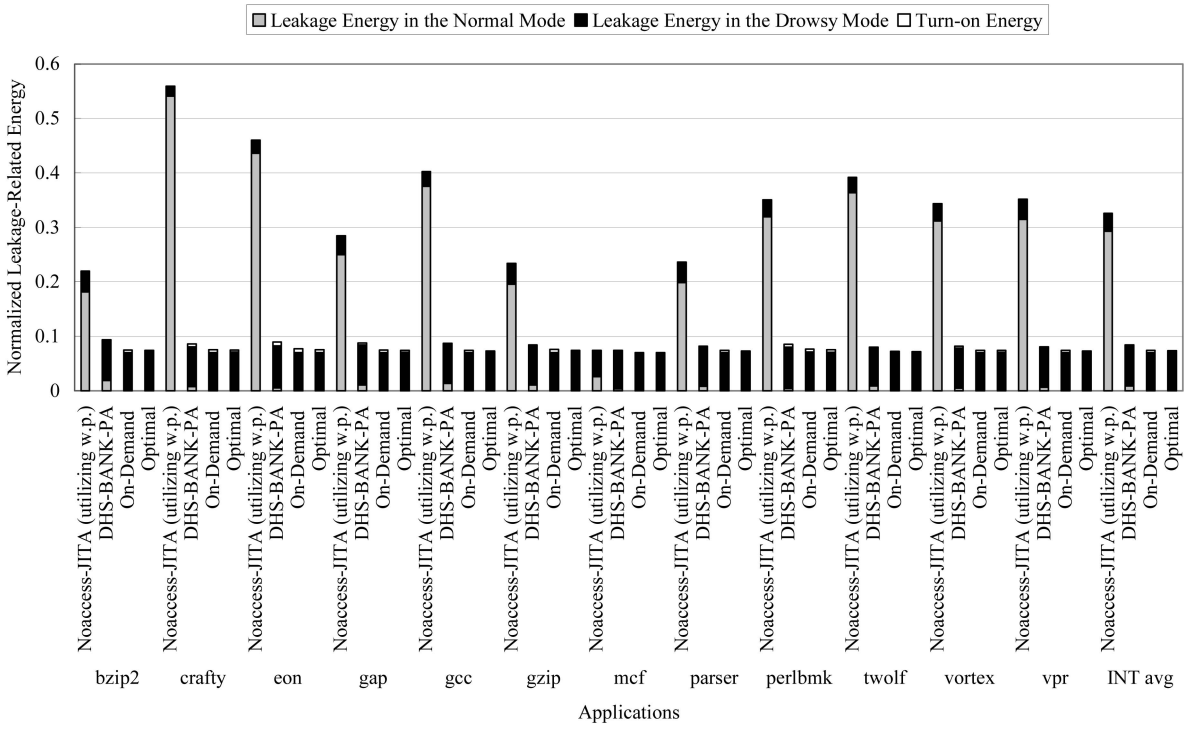

Fig. 7. Normalized leakage-related energy for SPEC2000 INT applications (direct-mapped).

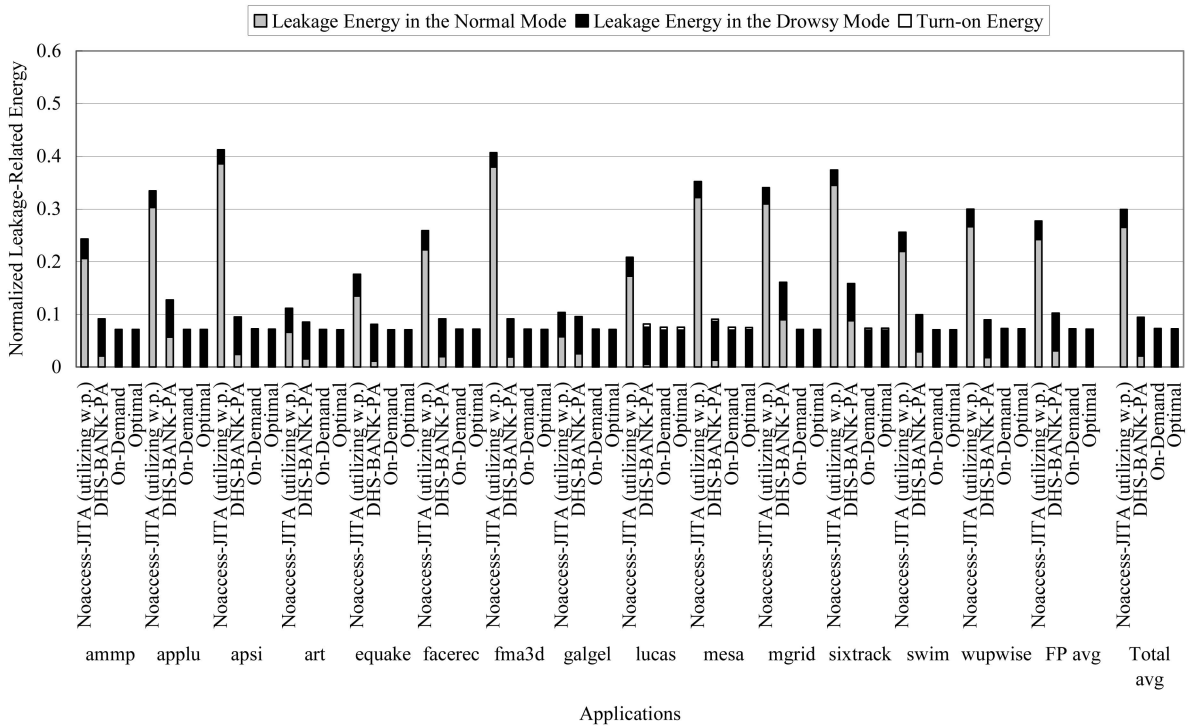

Fig. 8. Normalized leakage-related energy for SPEC2000 FP applications and total average (direct-mapped).

(utilizing way predictor) and the optimal policy is 22.7 percent and the difference between DHS-Bank-PA and the optimal policy is 2.2 percent (2.2 percent seems acceptable, but we should consider the extra runtime shown in Section 5.4).

Figs. 9 and 10 show normalized leakage-related energy to the base model in the 4-way set-associative cache. The base model does not use any leakage-saving policy, but it has a way predictor. The average leakage-related energy reduction is 72.3 percent, 90.8 percent, 92.65 percent, and 92.70 percent in the noaccess-JITA (utilizing way predictor), DHS-Bank-PA, on-demand, and optimal policies, respectively. These results are similar to the results in the direct-mapped cache.

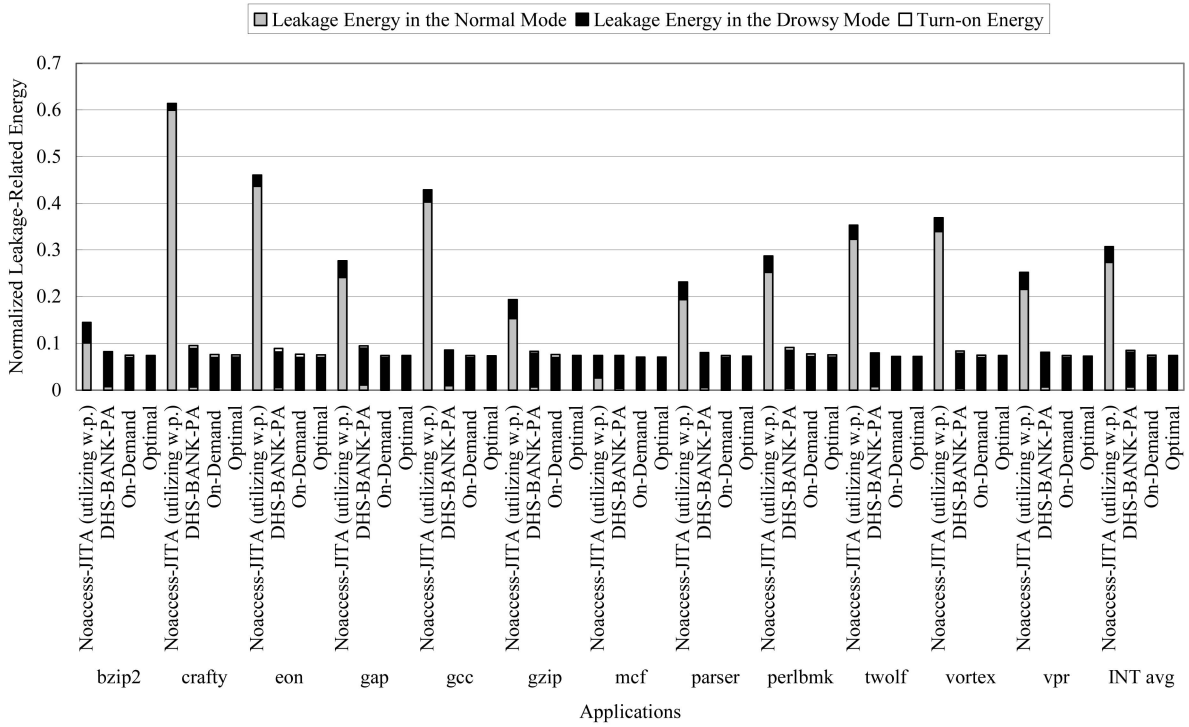

Fig. 9. Normalized leakage-related energy for SPEC2000 INT applications (4-way set-associative).

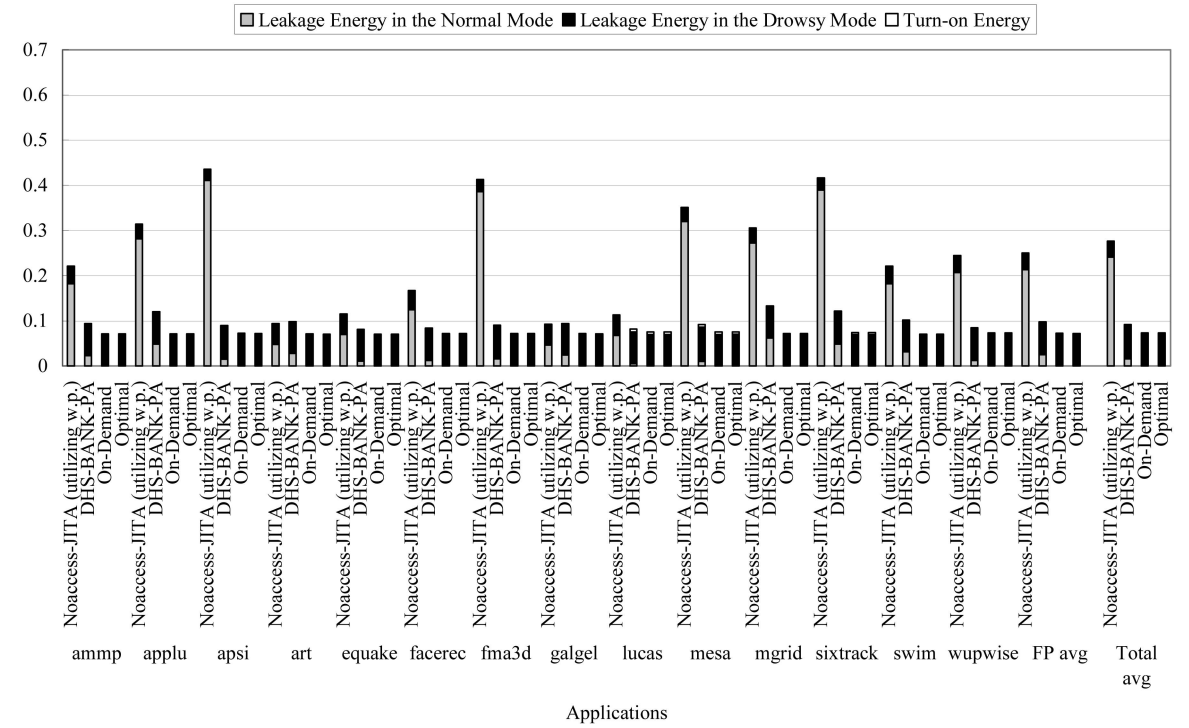

Fig. 10. Normalized leakage-related energy for SPEC2000 FP applications and total average (4-way set-associative).

5.3 Wake-Up Prediction Accuracy

Table 4 shows the branch prediction accuracy and the branch instruction ratio (the number of branch instructions/the number of total instructions) for SPEC2000 applications. On average, the branch prediction accuracy is 94.7 percent and the branch instruction ratio is 8.9 percent. Recall that the wake-up misprediction in the on-demand policy is mainly caused by a branch misprediction by incorrect target address. As the number of branch instructions gets smaller, the branch prediction accuracy affects the wake-up prediction accuracy less. For example, vortex and applu show similar branch prediction accuracy, but the branch instruction ratio of applu is much less than that of

TABLE 4
Branch Prediction Accuracy and Branch Instruction Ratio

INT Application	Branch Prediction Accuracy (%) (Branch Instruction Ratio (%))	FP Application	Branch Prediction Accuracy (%) (Branch Instruction Ratio (%))
bzip2	92.09 (13.00)	ammp	98.14 (6.09)
crafty	88.25 (11.22)	applu	98.49 (0.34)
eon	92.39 (13.21)	apsi	96.91 (3.29)
gap	89.35 (12.84)	art	96.04 (13.46)
gcc	91.10 (13.96)	equake	97.21 (3.83)
gzip	92.40 (12.43)	facerec	97.54 (6.75)
mcf	97.39 (19.38)	fma3d	98.76 (2.81)
parser	93.87 (15.30)	galgel	99.54 (5.65)
perlbmk	79.99 (13.42)	lucas	99.95 (1.35)
twolf	86.31 (12.00)	mesa	94.22 (8.83)
vortex	98.33 (17.46)	mgrid	95.85 (0.32)
vpr	93.11 (10.55)	sixtrack	98.18 (2.25)
		swim	99.66 (1.40)
		wupwise	97.36 (9.64)
INT avg	**91.21 (13.73)**	**FP avg**	**97.70 (4.72)**
		Total Avg	**94.70 (8.88)**

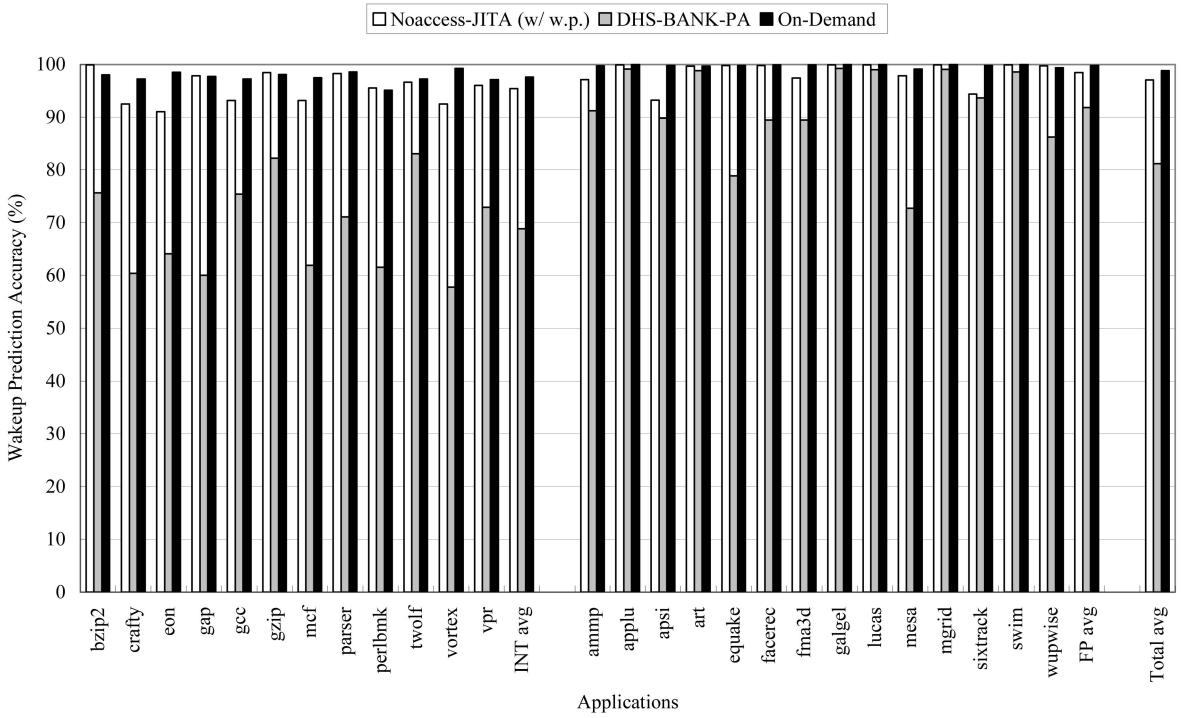

Fig. 11. Wake-up prediction accuracy per fetch, including the bitline precharging accuracy (direct-mapped).

vortex, resulting in the higher wake-up prediction accuracy of applu in Figs. 11 and 12.

Fig. 11 shows the wake-up prediction accuracy (the number of fetches with a target-mispredicted branch/the number of fetches) in the direct-mapped cache. The average wake-up prediction accuracy of noaccess-JITA (utilizing way predictor) is as high as 97.1 percent, although, for eon, the prediction accuracy is only 91.0 percent. On the other hand, the average wake-up prediction accuracy of the DHS-Bank-PA is 81.2 percent. In the proposed on-demand policy, the average wake-up prediction accuracy is as high as 98.8 percent. Even in the worst case, the wake-up prediction accuracy is no worse than 95 percent.

Fig. 12 shows the wake-up prediction accuracy, including bitline precharging and way prediction accuracy, in the four-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. The average accuracy of the noaccess-JITA (utilizing way predictor) is 97.5 percent since a set-associative cache makes it more difficult to predict subbank precharging. The on-demand and optimal policies show 87.5 percent and 88.7 percent accuracy, respectively, where there is little difference. The accuracy of DHS-Bank-PA is as low as 59.5 percent on the average, which might result in severe performance degradation. This is caused by flushing the previous subbank

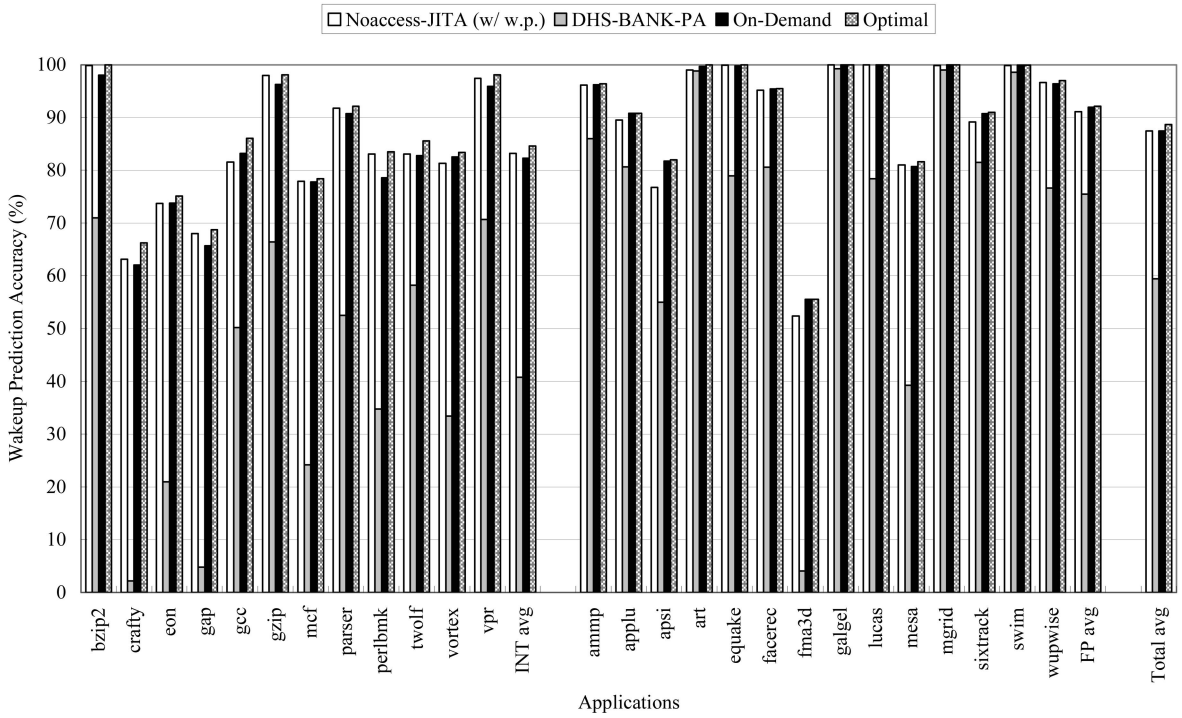

Fig. 12. Wake-up prediction accuracy per fetch, Including bitline precharging and way prediction accuracy (4-way set-associative).

when execution jumps from one subbank to another since the subbank hoppings are much more frequent in a set-associative cache.

5.4 Runtime Increase

Even a small increase in runtime leads to a substantial increase in the total processor energy consumption. This might outweigh the reduced L1 instruction cache leakage because the increased runtime extends the power dissipation of the entire processor. Note that, when a wake-up misprediction (including precharging misprediction and way misprediction) and an instruction cache miss occur at the same time, the wake-up penalty is hidden by the cache miss penalty. This means that the wake-up prediction accuracy is not quite proportional to the runtime.

Fig. 13 shows the runtime normalized to the base model in the direct-mapped cache. The noaccess-JITA (utilizing way predictor) increases runtime by 0.08 percent on average, but recall that leakage reduction is only 70.0 percent—much smaller than the other policies. The DHS-Bank-PA shows a 1.5 percent runtime increase on average. Even worse, in some applications, runtime is increased by more than 8 percent. The proposed on-demand policy increases the runtime by only 0.08 percent on the average. In perlbmk, the runtime of the on-demand policy is increased by 1.7 percent. This means that wake-up mispredictions are not overlapped with cache misses in the on-demand policy.

Fig. 14 shows the runtime normalized to the base model in the four-way set-associative cache. The increases of the average runtime are 0.11 percent, 3.33 percent, and 0.07 percent for noaccess-JITA (utilizing way predictor), DHS-Bank-PA, and the proposed on-demand policy, respectively. In crafty, the DHS-Bank-PA degrades the performance as much as 17.3 percent, which is especially severe.

5.5 Total Processor Energy and ED^2

Figs. 15, 16, 17, and 18 show the average total processor energy and average ED^2 $(\mathrm{Energy}^*\mathrm{Delay}^2)$ for all SPEC2000 applications in the direct-mapped cache, including the cost of increased runtime on whole-processor energy. Note that the "on-demand" graphs are almost overlapped with the "optimal" graph in the figures. The leakage-related energy of the x-axis in the above figures represents the leakage-related energy (leakage in the active mode, leakage in the energy-saving mode, and turn-on energy) in the instruction cache, as explained in (1) in Section 4.1. The total processor energy includes the dynamic leakage energy of the instruction cache, as well as the dynamic leakage energy consumed in other processor components except for the instruction cache. For example, supposing that the instruction cache accounts for 10 percent of the total processor energy and leakage comprises 50 percent of the instruction cache energy, the ratio (leakage-related energy)/(total processor energy) is 5 percent.

Since the on-demand policy and the optimal policy show similar performance and leakage-related energy reduction as shown in the previous sections, the total energy and ED^2 are also similar, regardless of the leakage-related energy ratio. Only in the case of perlbmk can we differentiate the on-demand policy and the optimal policy. This is due to the significantly increased runtime (1.7 percent) of the on-demand policy. However, even in this case, the on-demand policy generally shows the least total processor energy and ED^2. The total energy difference between the on-demand policy and the noaccess-JITA (utilizing way predictor) is only 0.5 percent when (leakage-related energy)/(total

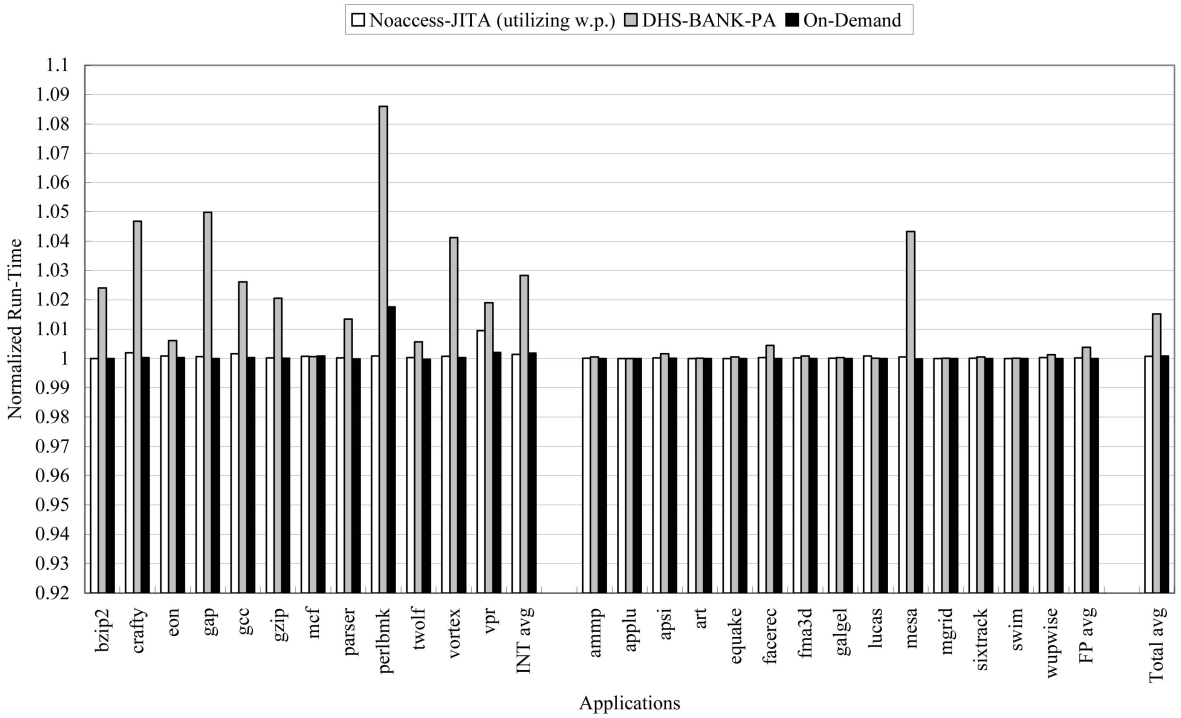

Fig. 13. Normalized runtime (direct-mapped).

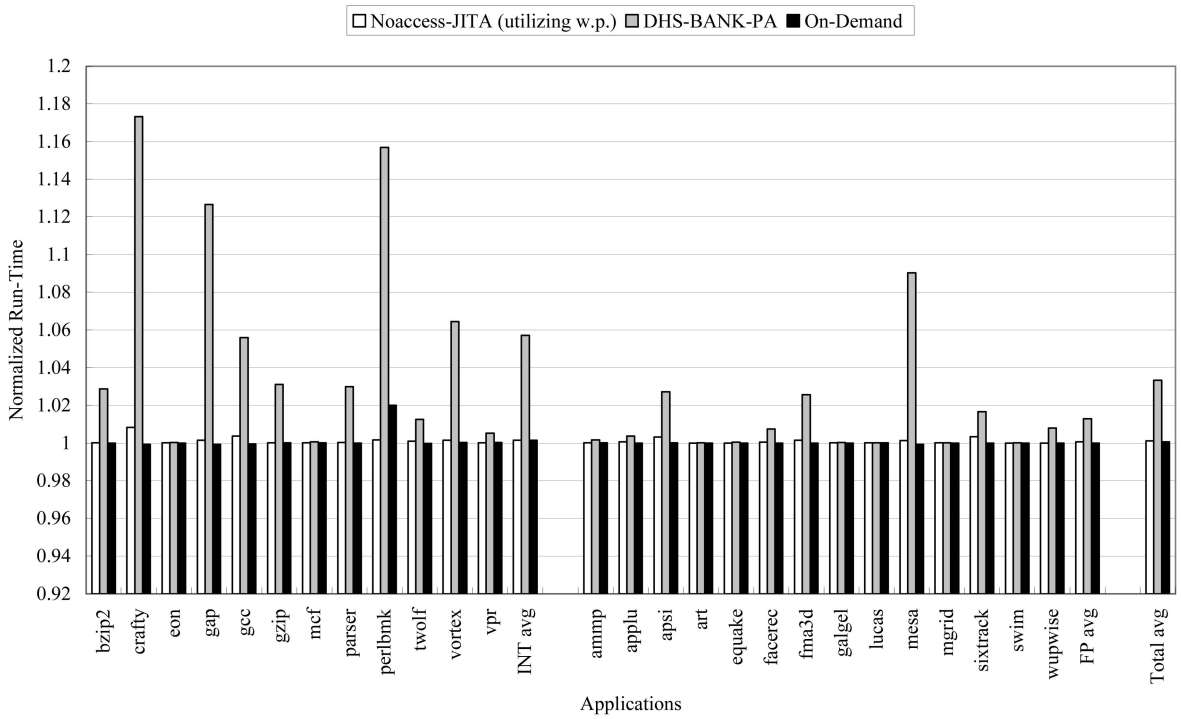

Fig. 14. Normalized runtime (4-way set-associative).

processor energy) is 2 percent. However, if the leakage-related energy accounts for a larger portion of the total processor energy, the difference becomes larger. Thus, in the embedded processor that may not have an L2 cache, the proposed on-demand policy is much more energy efficient.

Although Section 5.2 indicated that DHS-Bank-PA reduces the leakage-related energy about as well as the on-demand policy, the total processor energy of the DHS-Bank-PA is largest among the three policies due to its severely increased runtime. Even worse, for a (leakage-related energy)/(total processor energy) ratio of 2 percent, the ED^2 of the DHS-Bank-PA is worse than the base model with no leakage-saving mode.

In the four-way set-associative cache, the DHS-Bank-PA has larger total processor energy and ED^2 due to the more increased runtime compared to the direct-mapped cache.

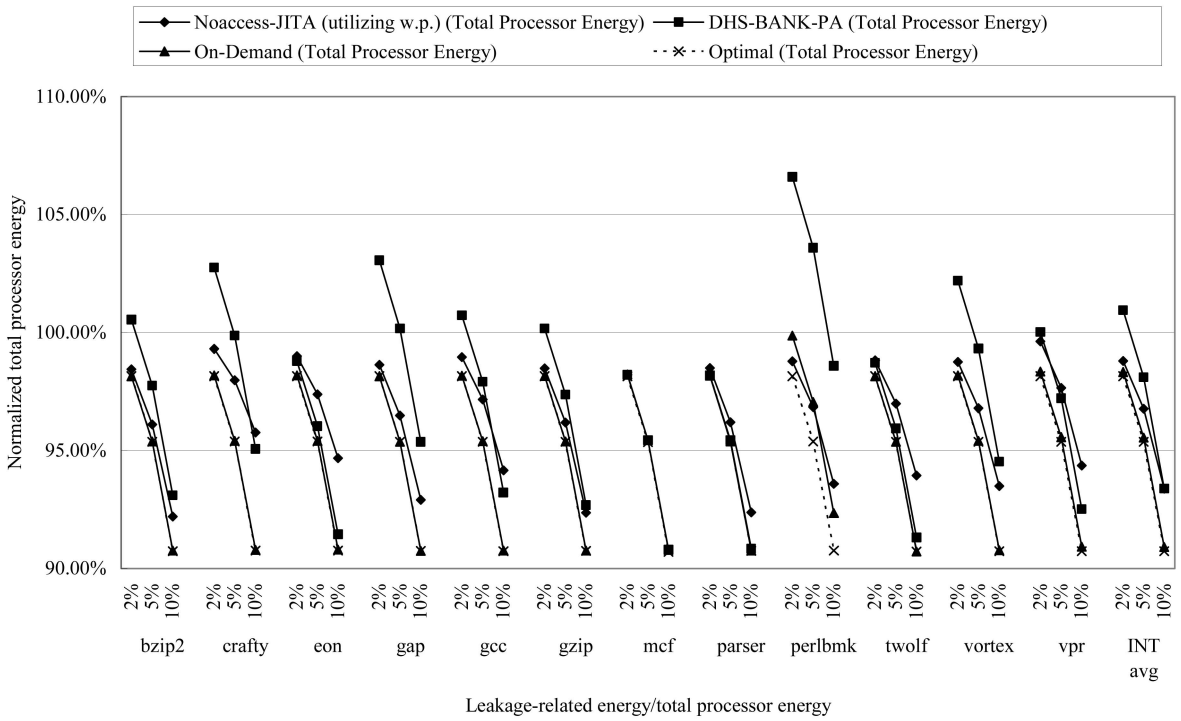

Fig. 15. Total processor energy for SPEC INT applications (direct-mapped). Note that the "on-demand" graphs are almost overlapped with the "optimal" graph.

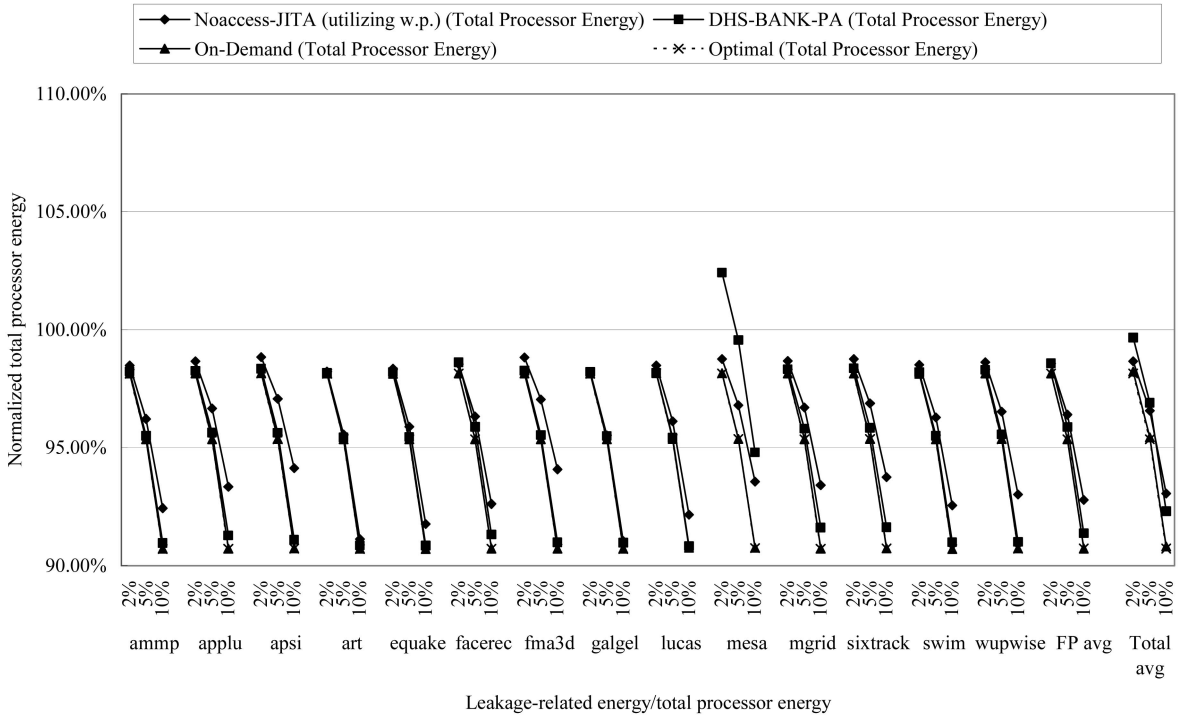

Fig. 16. Total processor energy for SPEC FP applications (direct-mapped). Note that the "on-demand" graphs are almost overlapped with the "optimal" graph.

The other graphs are almost identical to the graphs in Figs. 15, 16, 17, and 18 (we do not show the total processor energy and ED^2 for the four-way set-associative cache).

The on-demand policy does incur the extra energy of an extra pipeline register due to the additional wake-up stage. This is not included in the above total processor energy and ED^2 because its magnitude is small and because we can only approximately estimate its precise value. We estimate the energy cost of the extra pipeline register (approximately, $100\ bits * instruction\ width$ (instruction, PC, potential target, and a few control bits)—400 bits for the 4-wide processor studied here) to be about 0.1 W at 1.0 V and

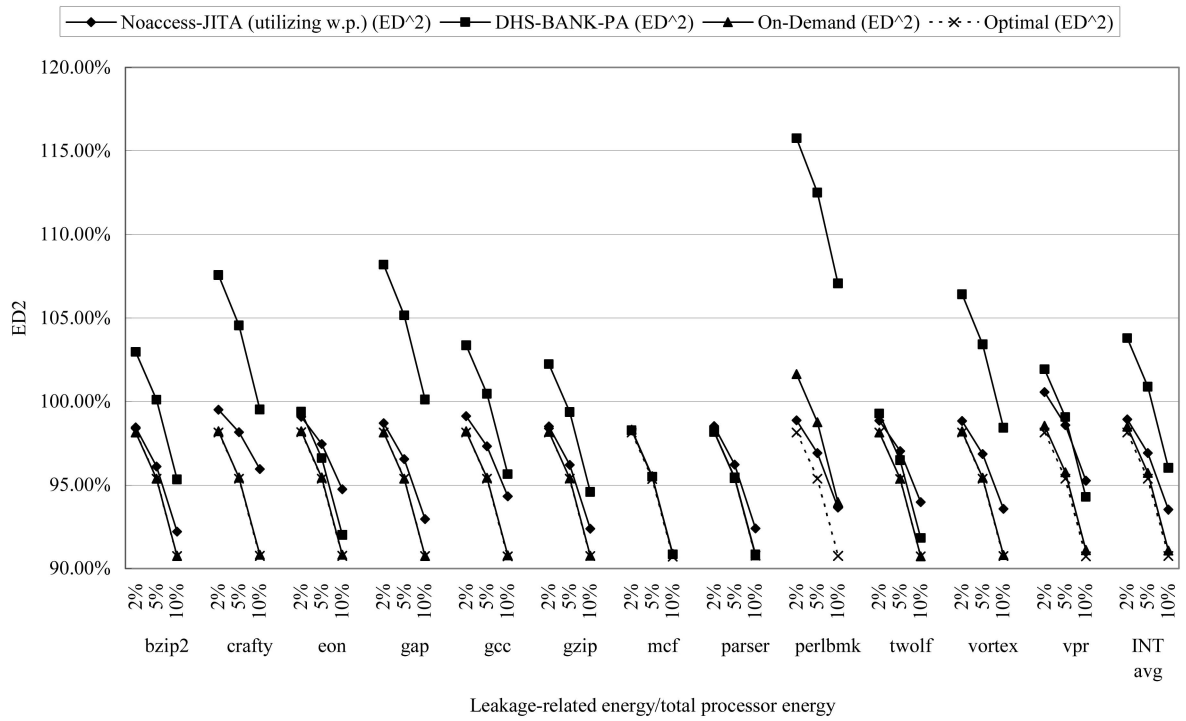

Fig. 17. ED^2 for SPEC INT applications (direct-mapped). Note that the "on-demand" graphs are almost overlapped with the "optimal" graph.

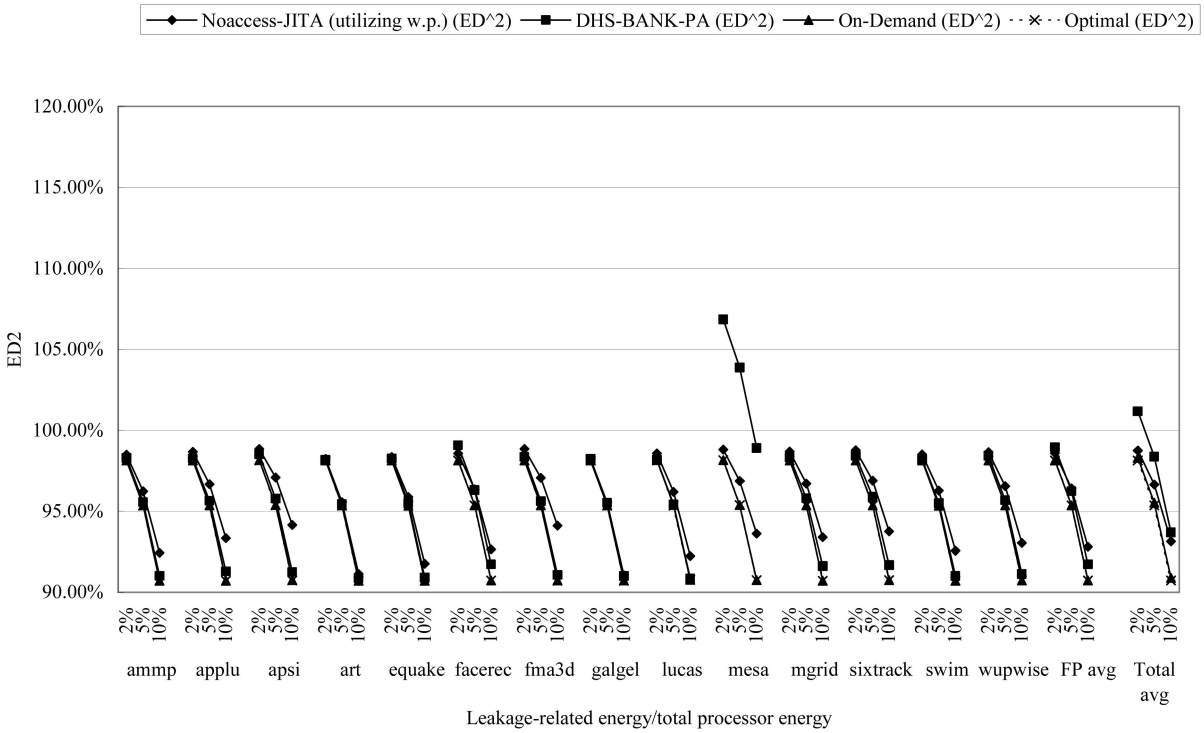

Fig. 18. ED^2 for SPEC FP applications (direct-mapped). Note that the "on-demand" graphs are almost overlapped with the "optimal" graph.

2.5 GHz. Even if we allow a substantial margin of error in this estimate, the cost of the pipeline register in microprocessors is negligible. This is based on the latch modeling data obtained by Li et al. using Nanosim for our work in [14], in which we verified that the power for a latch array scales almost linearly with the bit width and the number of words/entries and observed that writes are about 20 percent more expensive than reads (we assume one read + one write per cycle here).

Considering the extra energy overhead, the on-demand policy consumes much less overall processor energy than the noaccess-JITA (utilizing way predictor), where at least an additional 1,000 bits are required for wake-up prediction (details are explained in Table 5). The DHS-Bank-PA

TABLE 5
Hardware Overhead for the Policies

Noaccess-JITA (utilizing w.p.)	DHS-Bank-PA	On-Demand
1K bit + 2-read port way predictor (instead of 1-read port way predictor)	11K bit (=1024 + 1024*10)	10 bit + approx. 400 bit (=$\log_2$1024 + extra pipeline register)

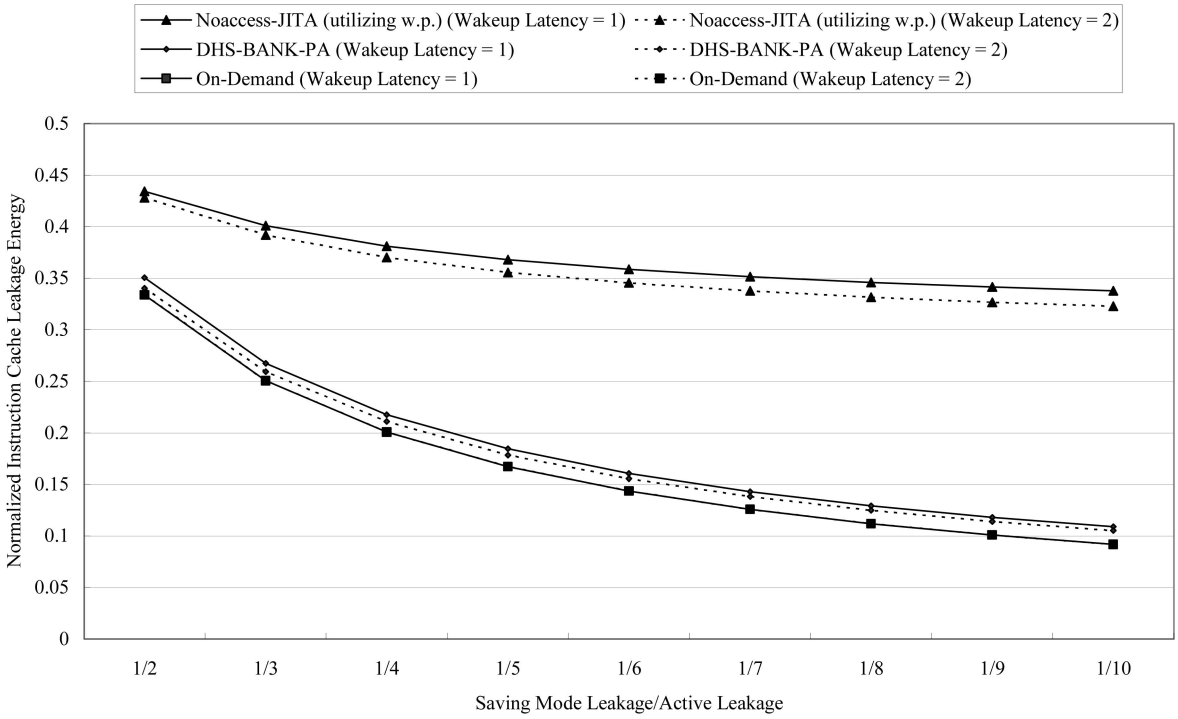

Fig. 19. Instruction cache leakage reduction combined with other leakage-saving circuit techniques. Note that the "on-demand" graphs are almost overlapped with the "optimal" graph.

already shows noticeable differences in terms of the total processor energy and the ED^2, compared to the on-demand policy. After considering an additional 11,000 bits for wake-up prediction (details are explained in Table 5) that is much more than the extra hardware (400 bits) in the on-demand policy, the difference becomes much larger. The on-demand policy therefore becomes much more energy efficient after reflecting the extra energy overhead of the additional hardware required.

5.6 Combining with Other Leakage-Saving Circuit Techniques

The proposed on-demand policy can be adopted for other leakage-saving circuit techniques that might have a longer wake-up penalty or more leakage saving. Even for other techniques such as ABB MTCMOS [19] and DRG [1] that have a longer wake-up penalty, the proposed on-demand policy can be used to reduce leakage energy. Fig. 19 makes it possible to estimate how much leakage energy is reduced when other leakage-saving circuit techniques are applied. We normalized the average instruction cache leakage energy to the base model for SPEC2000 applications, where the leakage energy due to extra runtime is captured, but the energy consumption from the other processor components, except for the instruction cache and the dynamic energy consumption in the instruction cache, are not considered (please note that "on-demand (one-wake-up-latency)" graphs are almost overlapped with the "on-demand (two-wake-up-latency)" graph).

Regardless of the wake-up latency, the on-demand policy consistently reduces the leakage energy most since its accuracy is high enough and its energy reduction is near optimal. When the wake-up latency is changed into two, the on-demand policy still reduces more leakage energy than the other policies.

5.7 Comparison of Hardware Overhead

For a wake-up prediction policy, hardware overhead is inevitable in addition to the DVS control circuitry. Noaccess-JITA (utilizing way predictor) requires one bit per cache line in order to detect whether the cache line is accessed or not in the fixed time period. In addition, for associative caches, it needs a two-read port way predictor (instead of the baseline one-port way predictor) for bitline precharging (subbank) prediction. In the DHS-Bank-PA, one bit per cache line is also required to store the access history. Additionally, 10 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 [6]. Since the BTB has 1,024 entries, the total storage overhead is 10,000. For

the proposed on-demand policy, a small register (for example, 10 bits for our 1,024-entry cache) that records the most recently accessed cache line and another pipeline register (400 bits) are needed. If the alternate path is awakened speculatively, as in Fig. 4, a two-read port way predictor is required. Otherwise, for the Fig. 3 option that is more common [28], the baseline one-read port way predictor is enough. Table 5 presents the total hardware overhead for each policy. The hardware overhead is crucial since it not only increases chip area but also incurs extra dynamic/leakage energy.

6 Conclusions and Future Work

In this paper, we propose an on-demand wake-up prediction policy using the branch prediction information. Our goal is not only less energy consumption but also consistent near-optimal performance. Noaccess-JITA (utilizing way predictor) shows competitive performance, but its energy consumption is more than three times that of the proposed policy on average. DHS-Bank-PA reduces energy close to the optimal policy, but it increases the runtime severely. In some cases, it increases the runtime by more than 15 percent. The proposed policy degrades the performance by only $0.06 \sim 0.08$ percent on average and 2.0 percent in the worst case. At the same time, active leakage energy is almost eliminated since only one cache line is active, whereas all other lines are in the drowsy mode. The leakage-related energy reduction by the proposed policy is, on average, 92.7 percent, almost identical to the reductions by the optimal policy. This is especially beneficial for controlling leakage in future instruction caches, which might be much larger. The total processor energy and the ED^2 of the proposed policy are also almost identical to those of the optimal policy. Therefore, we conclude that the proposed on-demand wake-up prediction policy is near optimal. We believe that there is no reason to try to reduce the remaining leakage by adopting non-state-preserving techniques at the risk of severe performance degradation.

In this paper, we apply the on-demand wake-up policy to the tag portion of the instruction cache. However, when the tag array is always in the active mode, the tag access can be moved to the wake-up stage, leading to tag matching in the wake-up stage. This results in a 100 percent accurate way prediction without any way predictor. The trade-off between the reduction of leakage energy in the tag part and perfect way prediction will be an interesting research topic. Another interesting topic is to combine the proposed on-demand policy with a trace cache [22]. Since the branch direction/target can be known at least several cycles earlier in the trace cache, the proposed on-demand policy combined with the trace cache can hide a multicycle wake-up penalty of other circuit techniques. Even with the trace cache, the noaccess-JITA (utilizing way predictor) still has large active leakage consumption and DHS-Bank-PA cannot hide the penalty due to its low wake-up prediction accuracy.

Acknowledgments

This work was funded by the US National Science Foundation under Grant CCF-0429765, the US Army Research Office under Grant W911NF-04-1-0288, a grant from Intel Microprocessor Research Laboratories (MRL), an IT National Scholarship Program from the Institute of Information and Technology Assessment (IITA) and Ministry of Information and Communication (MIC), Korea, and a Korea Research Foundation Grant of the Korean Government (KRF-2006-D00452). The authors would like to thank Karthik Sankaranarayanan for his help in using SimPoint and in estimating the extra energy consumption of an additional pipeline stage. They would also like to thank Nam Sung Kim for his helpful comments on validating their simulation model. Finally, they would like to thank the anonymous reviewers for their helpful feedback. S.W. Chung is the corresponding author for this paper.

References

[1] A. Agarwal, L. Hai, and K. Roy, "A Single-Vt Low-Leakage Gated-Ground Cache for Deep Submicron," *IEEE J. Solid-State Circuits,* vol. 38, pp. 319-328, Feb. 2003.

[2] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer,* vol. 35, pp. 59-67, 2002.

[3] S.W. Chung and K. Skadron, "Using Branch Prediction Information for Near-Optimal I-Cache Leakage," *Proc. 11th Asia-Pacific Computer Systems Architecture Conf.,* pp. 24-37, 2006.

[4] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," *Proc. 29th Int'l Symp. Computer Architecture,* pp. 148-157, 2002.

[5] 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 Trans. VLSI Systems,* vol. 10, pp. 91-95, Apr. 2002.

[6] J.S. Hu, A. Nadgir, N. Vijaykrishnan, M.J. Irwin, and M. Kandemir, "Exploiting Program Hotspots and Code Sequentiality for Instruction Caches Leakage Management," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 593-601, 2003.

[7] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," *Proc. 28th Int'l Symp. Computer Architecture,* pp. 240-251, 2001.

[8] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro,* pp. 24-36, 1999.

[9] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power," *IEEE Trans. VLSI Systems,* vol. 12, no. 2, pp. 167-184, Feb. 2004.

[10] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Single-VDD and Single-VT Super-Drowsy Techniques for Low-Leakage High-Performance Instruction Caches," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 54-57, 2004.

[11] L. Li, V. Degalahal, N. Vojaykrishnan, M. Kandemir, and M.J. Irwin, "Soft Error and Energy Consumption Interactions: A Data Cache Perspective," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 132-137, 2004.

[12] L. Li, I. Kadayif, Y.-F. Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and A. Sivasubramaniam, "Leakage Energy Management in Cache Hierarchies," *Proc. 11th Int'l Conf. Parallel Architectures and Compilation Techniques,* pp. 131-140, 2002.

[13] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, M. Stan, and K. Skadron, "State-Preserving vs. Non-State-Preserving Leakage Control in Caches," *Proc. Design, Automation and Test in Europe Conf. and Exhibition,* pp. 22-27, 2004.

[14] Y. Li, M. Hempstead, P. Mauro, D. Brooks, Z. Hu, and K. Skadron, "Power and Thermal Effects of SRAM vs. LatchMux Design," *Proc. ACM/IEEE Int'l Symp. Low-Power Electronics Design,* pp. 173-178, 2005.

[15] S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction," *Proc. 25th Int'l Symp. Computer Architecture,* pp. 132-141, 1998.

[16] S. McFaring, "Combining Branch Predictors," Technical Note TN-36, Digital Equipment Corp., June 1993.

[17] Y. Meng, T. Sherwood, and R. Kastner, "On the Limits of Leakage Power Reduction in Caches," *Proc. 11th Int'l Symp. High-Performance Computer Architecture,* 2005.

[18] M. Milenkovic, A. Milenkovic, and J. Kulick, "Demystifying Intel Branch Predictors," *Proc. Workshop Duplicating, Deconstructing and Debunking,* 2002.
[19] K. Nii et al., "A Low Power SRAM Using Auto-Backgate-Controlled MT-CMOS," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 293-298, 1998.
[20] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 90-95, 2000.
[21] G. Reinman and B. Calder, "Using a Serial Cache for Energy Efficient Instruction Fetching," *J. Systems Architecture,* vol. 50, no. 11, pp. 675-685, 2004.
[22] E. Rotenberg, S. Nennett, and J.E. Smith, "A Trace Cache Microarchitecture and Evaluation," *IEEE Trans. Computers,* vol. 48, no. 2, pp. 111-120, Feb. 1999.
[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 1997.
[24] S.H. Shin, S.W. Chung, and C.S. Jhon, "On the Reliability of Drowsy Instruction Caches," *Proc. 11th Asia-Pacific Computer Systems Architecture Conf.,* pp. 445-451, 2006.
[25] S. Yang and B. Falsafi, "Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* 2003.
[26] 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. Seventh Int'l Symp. High-Performance Computer Architecture,* pp. 147-157, 2001.
[27] W. Zhang, J. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "Compiler-Directed Instruction Cache Leakage Optimization," *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* pp. 208-218, 2002.
[28] ARM, *ARM 1136 Technical Reference Manual,* http://www.arm.com, 2007.
[29] *ITRS (Int'l Technology Roadmap for Semiconductor),* http://public.itrs.net, 2001.
[30] Standard Performance Evaluation Corp., http://www.specbench.org, 2007.
[31] VAR Business, *Intel Clears Up Post-Tejas Confusion,* http://www.varbusiness.com/sections/news/breakingnews.jhtml?articleId=18842588, 2007.

Sung Woo Chung received the BS degree in computer engineering and the PhD degree in electrical and computer engineering from Seoul National University, Korea, in 1996 and 2003, respectively. He worked as an academic visitor for the IBM T.J. Watson Research Center, Yorktown Heights, New York, in 2002. From 2003 to 2005, he worked for Samsung Electronics as a senior engineer. In 2005, he worked as a research scientist at the University of Virginia at Charlottesville. He joined the Division of Computer and Communication Engineering at Korea University, Seoul, as an assistant professor in 2006. His research interests include technology-aware design for microarchitecture and SoC (system on chip) and architectural supports for flash memories. He is a member of the IEEE Computer Society.

Kevin Skadron received the BSEE degree and the BS degree in economics from Rice University, Houston, Texas, and the MA and PhD degrees from Princeton University, Princeton, New Jersey. He joined the Department of Computer Science at the University of Virginia at Charlottesville in 1999 and is now an associate professor. His research interests focus on the implications of technology trends and physical constraints (for example, power, temperature, and reliability) for future highly multicore architectures. He is a cofounder and associate editor-in-chief of the *IEEE Computer Architecture Letters*. He recently served as program or general cochair of the PACT '06, MICRO '04, and PACT '02. He was a coauthor the articles that won Best Student Paper Awards at ISCA '03, RTSS '03, and SHAMAN '02. He is a senior member of the IEEE and a member of the IEEE Computer Society.