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

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

***Abstract*-This paper describes a new *on-demand* wakeup 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% with only 0.08% performance overhead on average. The branch prediction based approach requires an extra pipeline stage for wakeup, which adds to branch misprediction penalty. Fortunately, this cost is mitigated because the extra wakeup stage is overlapped with misprediction recovery. This paper assumes the super-drowsy 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, Wakeup Policy**

I. INTRODUCTION

Power dissipation has emerged as a major concern both for high-end processors and 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 total energy. The 2001 International Technology Roadmap for Semiconductor [29] 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 [15]. 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 [19], DRG (Data-Retention Gated-ground) [1],

[1] *Sung Woo Chung is with the Division of Computer and Communication Engineering, Korea University, Seoul 136-713, KOREA. (telephone: 82-2-3290-3194, e-mail: swchung@korea.ac.kr). Corresponding Author*

[2] *Kevin Skadron is with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904-4740. (telephone: 434-982-2042, e-mail: skadron@cs.virginia.edu).*

Digital Object Indentifier 10.1109/TC.2007.70770

Gated-Vdd [20], and DVS for Vdd (which is also called drowsy cache) [4]. In the ABB MTCMOS technique, threshold voltage is dynamically changed but the wakeup penalty between active mode and leakage saving mode is long, making it difficult for use in L1 caches [5]. DRG retains the data while reducing leakage by gating ground and using remaining leakage to operate the cells in subthreshold mode and retain cell contents. It is promising for less timing-critical caches such as L2 [12], but again, the wakeup penalty is long, making it difficult for 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. 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 [5]. If the line is put to sleep prematurely and is later needed, 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 re-fetch 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 “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. 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 [4].

Among the above four techniques, the drowsy technique appears most suitable for L1 instruction caches, since it retains data and has short wakeup penalty. In order to alleviate 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 [4]. 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 and negligible performance degradation. In this paper, we propose and evaluate a new *on-demand* wakeup prediction policy [3] for an instruction cache. By on-demand, we mean that *only the cache lines currently in use 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. Good wakeup 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 wakeup prediction, we can allow a new pipeline stage between branch prediction and instruction cache fetch. On most branch mispredictions, the extra wakeup 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 III.

This work focuses on use of drowsy mode (actually super-drowsy mode [10], explained in

Section II) as 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. 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 II explains the concept of the drowsy/super-drowsy cache, reliability of the drowsy instruction cache, and previously proposed prediction policies. Section III proposes a new on-demand wakeup prediction policy by using branch prediction information. Section IV presents the analytical model for evaluation and simulation environments. Section V evaluates energy/performance for the proposed policy. Finally Section VI concludes the paper.

II. BACKGROUND WORK

A. Drowsy/Super-Drowsy Cache Circuit Technique

The drowsy cache technique [4] has received a great deal of attention, because it retains data while providing a short wakeup 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 active mode with nominal voltage, the cache line operates the same as in a conventional cache. In drowsy mode, however, the cache line cannot be accessed even though data is retained. After being woken up, 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 gated-Vdd, but data retention implies the drowsy cache does not need to refetch instructions and allows a more aggressive sleep policy. Moreover,

wakeup penalty is short: one cycle is expected in 70 nm technology [4].

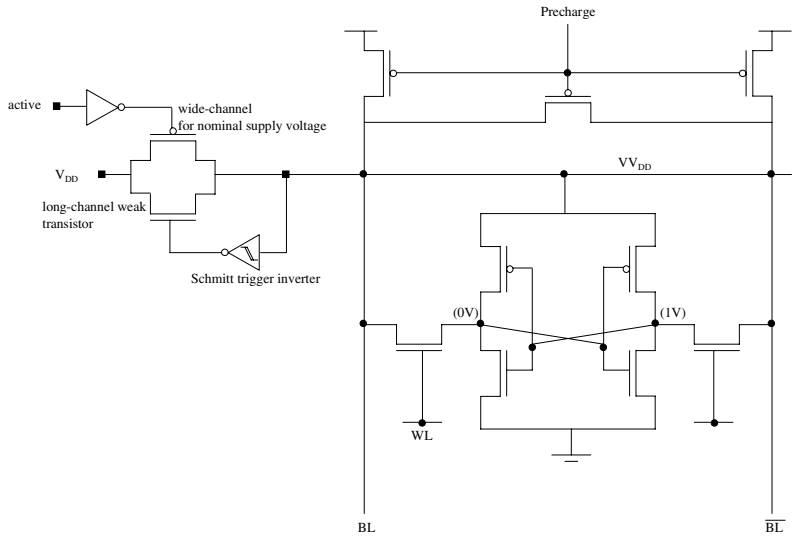

Fig. 1. Implementation of the super-drowsy cache line

Kim et.al proposed a refinement of this technique, called *super-drowsy cache* [10]. As shown in Figure 1 [10], 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 [25] 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.

B. Reliability of the Drowsy Instruction Cache

There has been concern that drowsy cache is more susceptible to soft errors than conventional caches [11]. Some researchers are reluctant to adopt drowsy (or super-drowsy) cache circuit technique, since the error rate is exponentially dependent on the decrease of the supply voltage. Without special error detection/correction technique, the reliability of the drowsy cache is lower than that of conventional cache. However, this is true only in 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 partiy and faulty lines can be re-fetched [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.

C. Previous Wakeup Prediction Policies

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* [4]: 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 [9] – *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 [9] due to large sub-bank turn-on energy. Zhang et.al. proposed the *Loop* policy [27] 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 [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-PA* (DHS-Per Access) 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 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 wakeup. As shown in [6], although the DHS-Bank-PA reduced leakage energy significantly, average performance degradation is as much as 2.3%, which incurs extra dynamic and leakage power.

The conventional super-drowsy cache deploys the noaccess-JITA policy, and in order to achieve high accuracy [10], up to a 32K-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 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 whose bitlines to be precharged in advance, since the on-demand gated precharge incurs extra penalty to enable an inactive sub-bank, and this can result in significant run-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 noaccess-JITA/NTSBP is poor in some applications. By slightly modifying noaccess-JITA/BTSBP, wakeup prediction can be adopted for more accuracy. When the way predictor can have 2-read ports in order to predict the next cache line that will be woken up as well, the prediction accuracy is higher and the NTSBP is unnecessary (In this paper, we call this policy as *Noaccess-JITA utilizing w.p. (Way Predictor)*).

In this paper, we compare the proposed on-demand policy to noaccess-JITA (utilizing w.p.) 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 the policies. All these policies are explained in Table I to prevent possible confusion.

TABLE I
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

III. NOVEL WAKEUP PREDICTION POLICY: UTILIZING BRANCH PREDICTION INFORMATION

In conventional drowsy (including super-drowsy) instruction caches, branch predictors are only used for conventional branch direction/target prediction. In previous wakeup 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. Accuracy of the previous policies is therefore highly dependent on the locality and the portion of active leakage is still large. As shown in Figure 2(a), the 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 wakeup penalty. The accuracy of these predictors has generally been problematic.

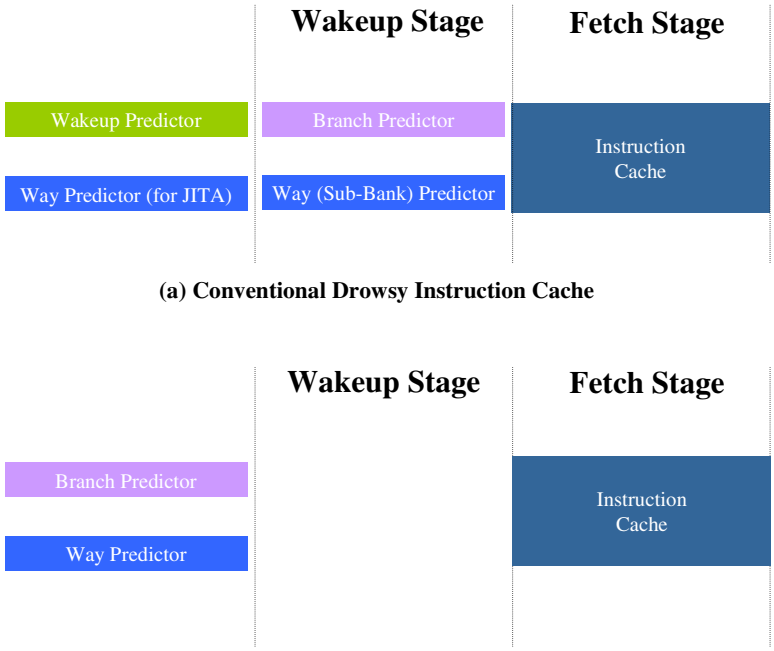

Fig. 2. Pipeline stage comparison

Instead, for near-optimal leakage energy reduction and performance, we propose a new wakeup prediction policy which enables on-demand wakeup. The key is that the branch predictor already identifies which line needs to be woken up. No additional wakeup-prediction structure is needed. In the proposed policy, as shown in Figure 2(b), the instruction cache is therefore accessed one or more cycles 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 wakeup 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 wakeup prediction policies (Figure 3 (b) and Figure 4 (b)), depending on 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 fetch address (Figure 3 (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, e.g. Figure 4 (a). This makes the impact of the wakeup penalty essentially negligible. As we 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 Figure 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) wakeup 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 wakeup cycle is overlapped with the other misprediction-recovery tasks, the on-demand scheme does not increase the misprediction penalty.

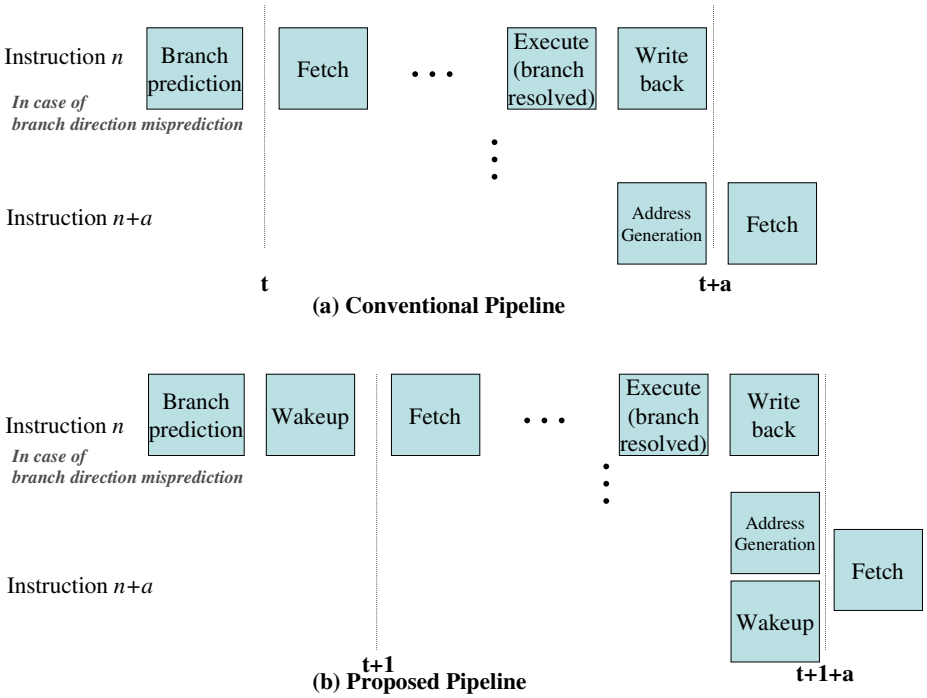

Fig. 3. Pipeline structure (when there is one-cycle penalty for effective address calculation). "a" represents the branch resolution latency.

- 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 Figure 4, this can be achieved if branches are resolved sufficiently early to allow wakeup 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 small portion of total instructions and turn-on energy is negligible, 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 (Figure 3 (a)) appears to be common [28]. For this reason, we selected the on-demand policy in Figure 3 for our evaluations in the remainder of this paper.

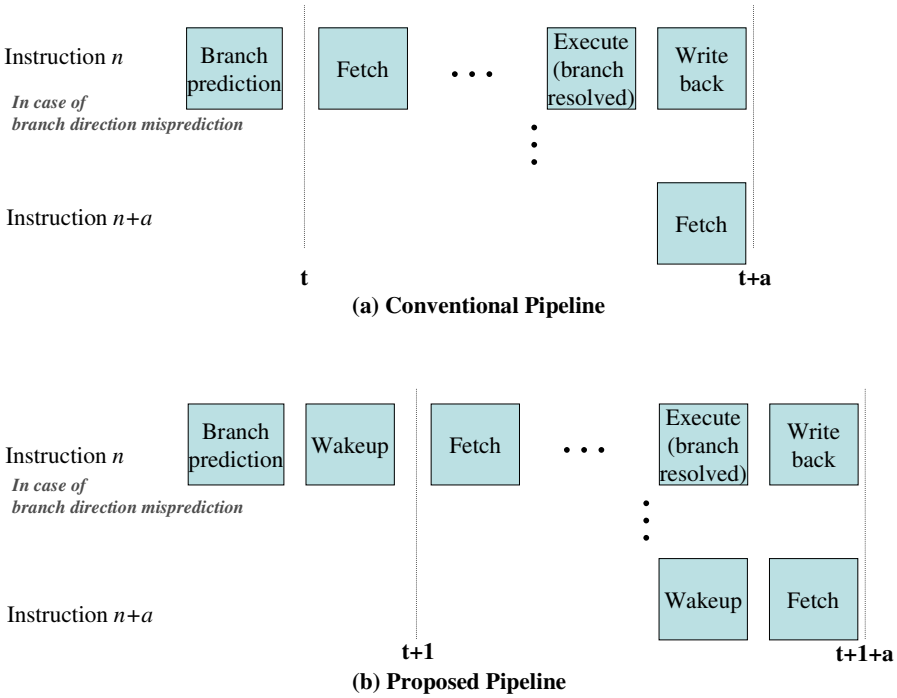

Fig. 4. Pipeline structure (when there is no penalty for effective address calculation)

We have now established that branch direction mispredictions generally need not incur 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 branch direction misprediction* (As previously explained, we can not hide the penalty *in case of branch target 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 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 (ex. In case of a 32B cache line and a 32 bit instruction, the probability is 12.5%).

In the on-demand policy, only the one cache line (or at most two cache lines in Figure 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 wakeup, the currently awake line is put to sleep. For a set-associative cache, the entire set could be woken up, but to save energy, only one way should be woken up. We adopt a way predictor [21] that employs an MRU (Most Recently Used) 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 woken up. 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 woken up and remain active. In the noaccess-JITA (utilizing w.p.), the way predictor can have 2-read ports for wakeup prediction as well as precharging and fetching. 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 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 woken-up cache line can be read in the proposed on-demand policy. Note also that, in this paper, we apply the on-demand wakeup 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 wakeup stage, leading to tag matching in the wakeup stage. Although this provides 100% accurate way prediction without any way predictor, we take the more pessimistic approach and assume a way predictor is needed for set-associative caches.

Figure 5 shows one example of the proposed policy for the Figure 3 option. After a misprediction by incorrect target address, the recovered target address (0x00182f10) is woken up

in cycle n. At the same time, the branch predictor is looked up to predict next fetch block. In the conventional pipeline, the first branch predictor access is done in cycle n+1. There is no predicted taken branch in the fetch block (0x00182f10), leading to waking up the next 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 wakeup address. Accordingly, the block (0x001820a0) is woken up in cycle n+2 and fetched in cycle n+3. Please note that the proposed policy does not affect the branch prediction accuracy.

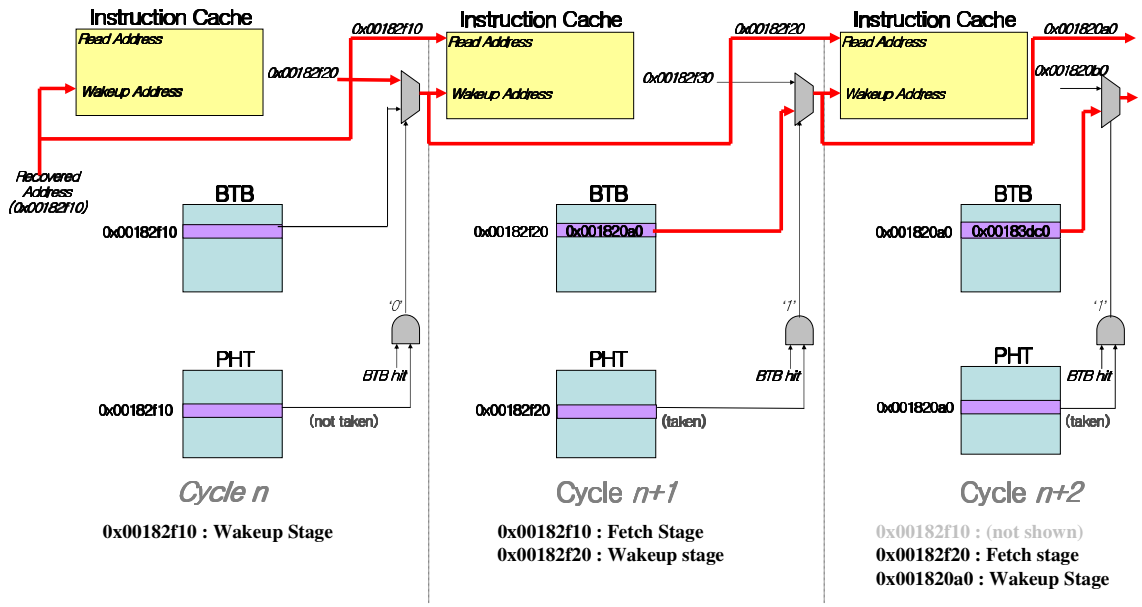

Fig. 5. Example operation of the on-demand wakeup prediction policy, in which fetch resumes at address 0x0018sf10 after a misprediction.

IV. EXPERIMENTAL METHODOLOGY

A. Analytical Models

Leakage-related energy includes leakage energy in the active mode, leakage energy in the drowsy mode (including super-drowsy mode), turn-on energy for prediction, turn-on energy for

correction of wakeup 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 II summarizes the notation used there.

TABLE II
NOTATION DESCRIPTION

Notation	Description
P_i	Leakage power consumed for a cache line in the *i* mode
$P_{turn\text{-}on}$	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
$TN_{prediction}$	Total Number of cache lines turned on by wakeup prediction
$TN_{correction}$	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

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

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

In the base model, all 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 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 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 the following formula (2), extended from [17], the 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.

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

B. 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 III. Table III also gives the technology and power/energy parameters used in this paper for modeling the I-cache. The power/energy parameters are based on the 70nm/1.0V technology [10]. We only specifically model power in the cache and omit a detailed cycle-by-cycle whole processor power model with SimpleScalar, 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 I-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 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 more accurate branch predictor were selected for evaluation, the proposed policy would perform even better, while it would not help wakeup 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 w.p.), 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 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 (when the access count exceeds over 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.

TABLE III
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)

V. SIMULATION RESULTS

This section presents simulation results comparing the proposed on-demand policy to other policies in terms of energy reduction and run-time 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.

A. Drowsy Fraction and Gated Bitline Precharging Fraction

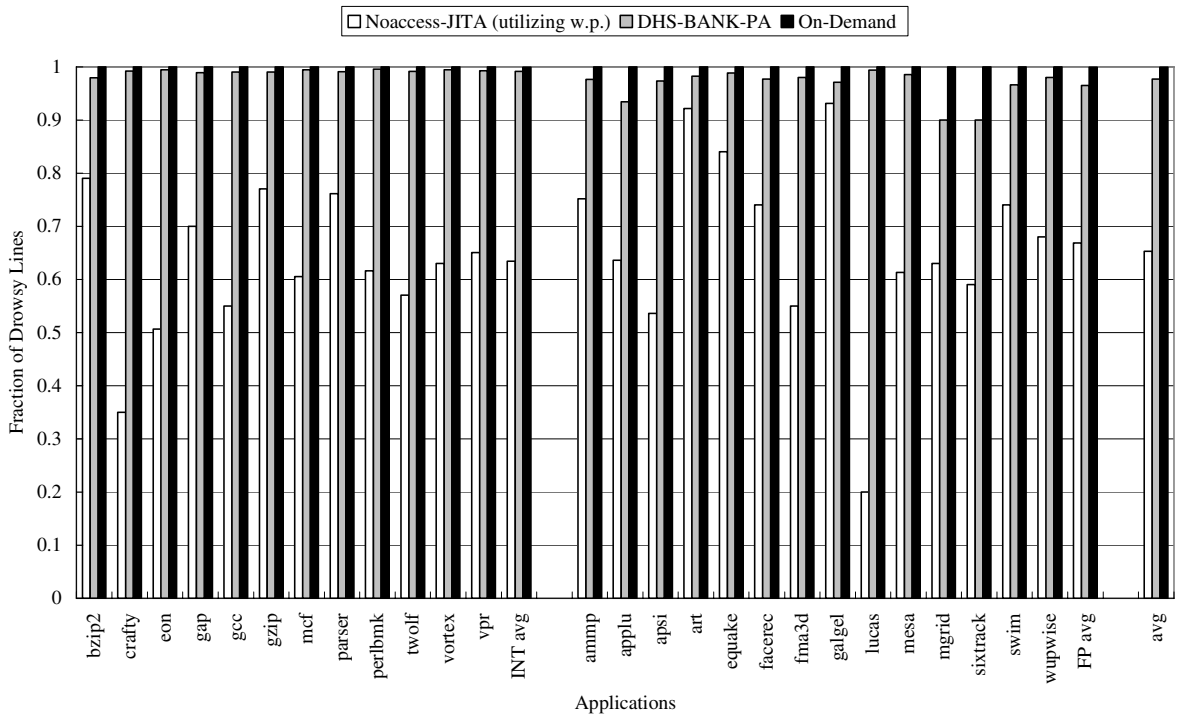

Fig. 6. Average drowsy fraction in instruction cache (direct-mapped)

Figure 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 99.9% drowsy fraction, on average. Since the update window size of the noaccess-JITA (utilizing w.p.) is as large as 32K, the drowsy fraction is relatively small (average 65.3%). In the DHS-Bank-PA, the average drowsy fraction is

97.7%. 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 case of the 4-way set-associative instruction cache, the drowsy fraction of the on-demand policy is 99.9% but the drowsy fraction of the noaccess-JITA (utilizing w.p.) is 69.6%, which is again less than 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 Figure 6.

In case of bitline precharging prediction, there is no energy penalty but there is one cycle timing penalty when mispredicted. In the 4-way set-associative caches, the bitline precharging prediction is same as way prediction for noaccess-JITA (utilizing w.p.) and on-demand policy (or next sub-bank prediction for DHS-Bank-PA). Consequently, the fraction of isolated bitlines in the 4-way set-associative is always 87.5% (1 sub-bank/8-sub-banks). Since a way predictor is used for sub-bank prediction for the direct-mapped cache, the fraction of gated bitline precharging is always 87.5 % (1 sub-bank/8 sub-banks) even in the direct-mapped cache.

B. Total Leakage-Related Energy

Figure 7 and Figure 8 show normalized leakage-related energy to the base model in the direct-mapped cache (As explained in Section IV. A, the base model is a conventional cache that does not perform leakage control), reflecting the increased run-time. The noaccess-JITA (utilizing w.p.) shows inconsistent and relatively small reductions depending on applications, whereas the other policies show very consistent reductions. Average leakage-related energy reduction is 70.0%, 90.6%, 92.66%, and 92.71% in the noaccess-JITA (utilizing w.p.), DHS-Bank-PA, on-demand, and optimal policies, respectively.

In the on-demand 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, such as the noaccess-JITA (utilizing w.p.) and the DHS-Bank-PA. However, turn-on energy in the on-demand policy still accounts for a small portion of 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%. In contrast, the average leakage-related energy difference between the JITA (utilizing w.p.) and the optimal policy is 22.7%, and the difference between DHS-Bank-PA and the optimal policy is 2.2% (2.2% seems acceptable but we should consider the extra run-time shown in Section V. D).

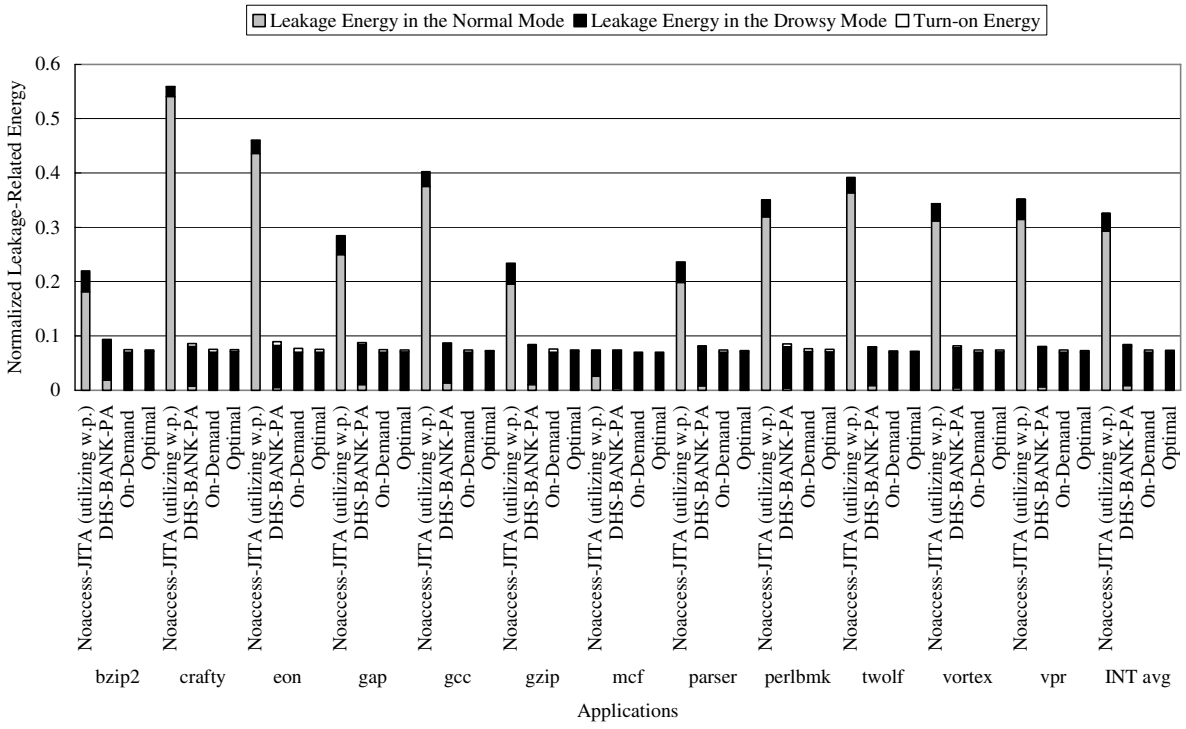

Fig. 7. Normalized leakage-related energy for SPEC2000 INT applications (direct-mapped)

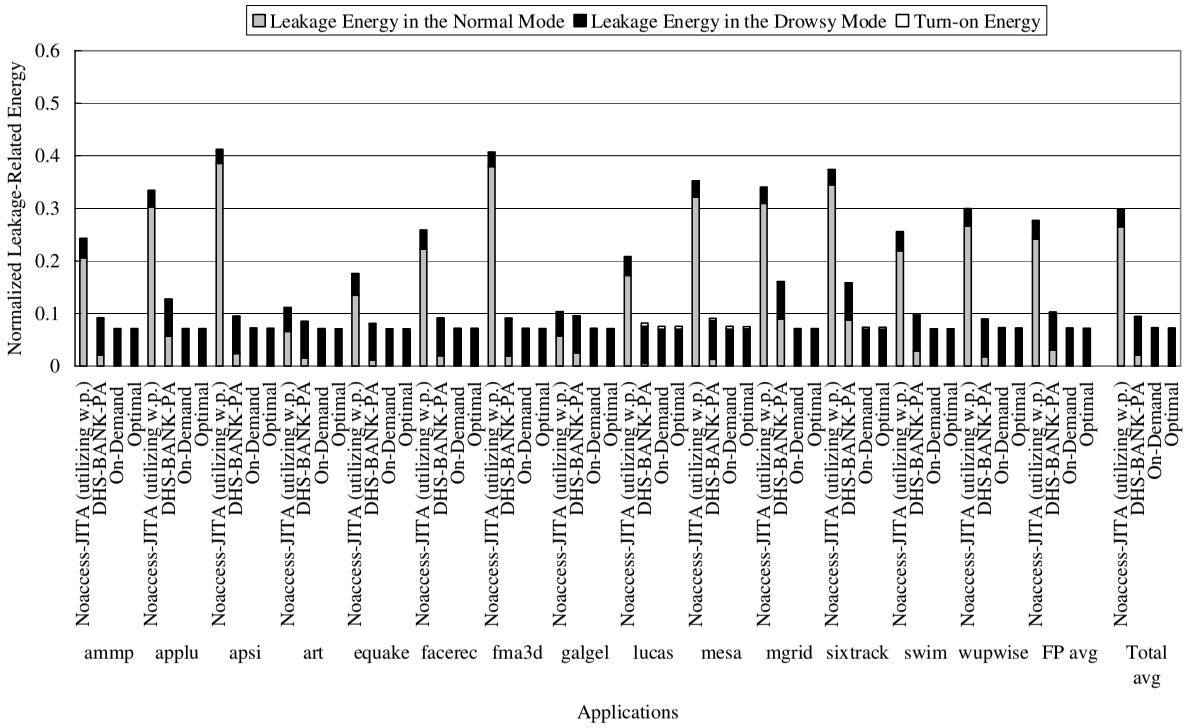

Fig. 8. Normalized leakage-related energy for SPEC2000 FP applications and total average (direct-mapped)

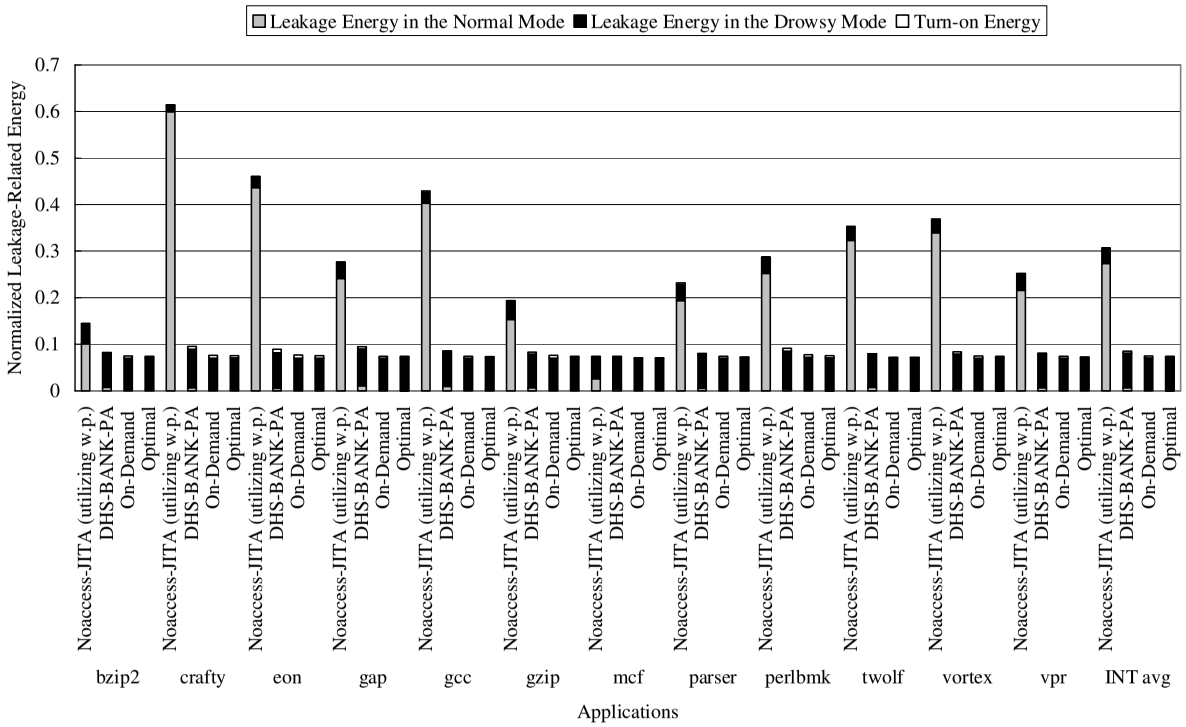

Fig. 9. Normalized leakage-related energy for SPEC2000 INT applications
(4-way set-associative)

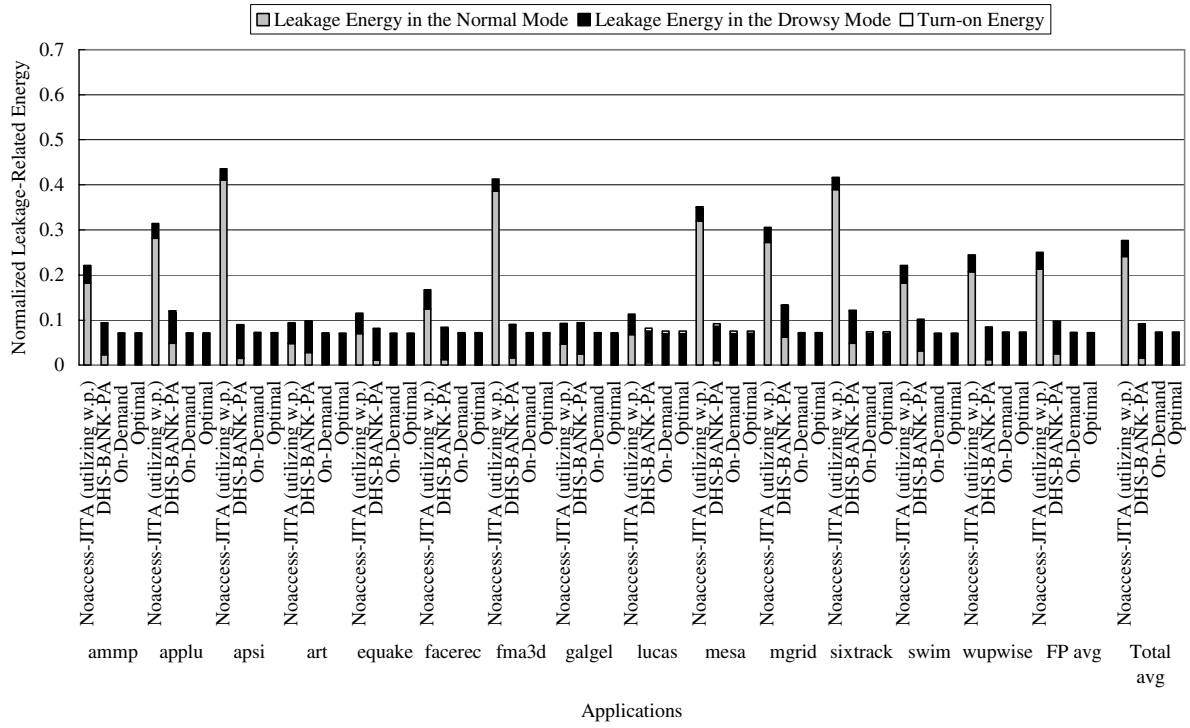

Fig. 10. Normalized leakage-related energy for SPEC2000 FP applications and total average (4-way set-associative)

Figure 9 and Figure 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 the way predictor. Average leakage-related energy reduction is 72.3%, 90.8%, 92.65%, and 92.70% in the noaccess-JITA (utilizing w.p.), DHS-Bank-PA, on-demand, and optimal policies, respectively. These results are similar to the results in the direct-mapped cache.

C. Wakeup Prediction Accuracy

Table IV shows the branch prediction accuracy and the branch instruction ratio (#branch instructions/#total instructions) for SPEC2000 applications. On average, the branch prediction accuracy is 94.7% and the branch instruction ratio is 8.9%. Recall that wakeup misprediction in the on-demand policy 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, vortex and applu show similar branch prediction accuracy

but the branch instruction ratio of applu is much less than that of vortex, resulting in higher wakeup prediction accuracy of applu in Figure 11 and Figure 12.

TABLE IV
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)**

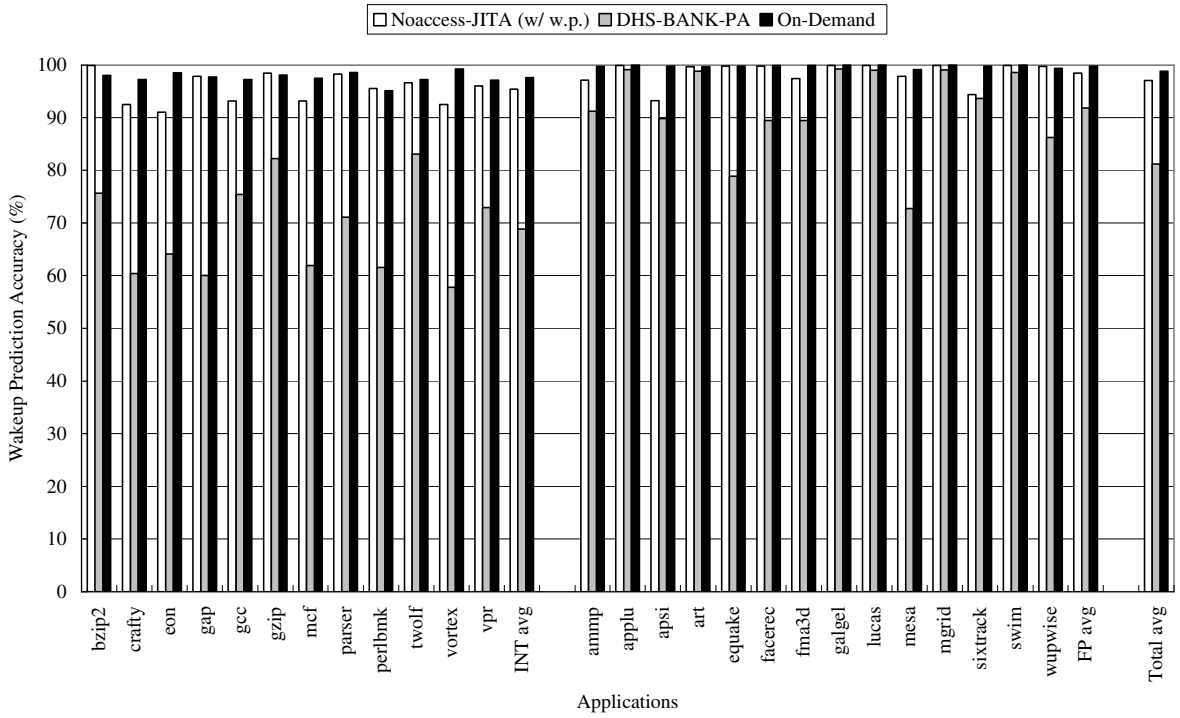

Fig. 11. Wakeup prediction accuracy per fetch, including bitline precharging accuracy (direct-mapped)

Figure 11 shows the wakeup prediction accuracy (#fetches with a target-mispredicted branch/# fetches) in the direct-mapped cache. Average wakeup prediction accuracy of noaccess-JITA (utilizing w.p.) is as high as 97.1%, although for eon, the prediction accuracy is only 91.0%. On the other hand, the average wakeup prediction accuracy of the DHS-Bank-PA is 81.2%. In the proposed on-demand policy, average wakeup prediction accuracy is as high as 98.8%. Even in the worst case, the wakeup prediction accuracy is no worse than 95%.

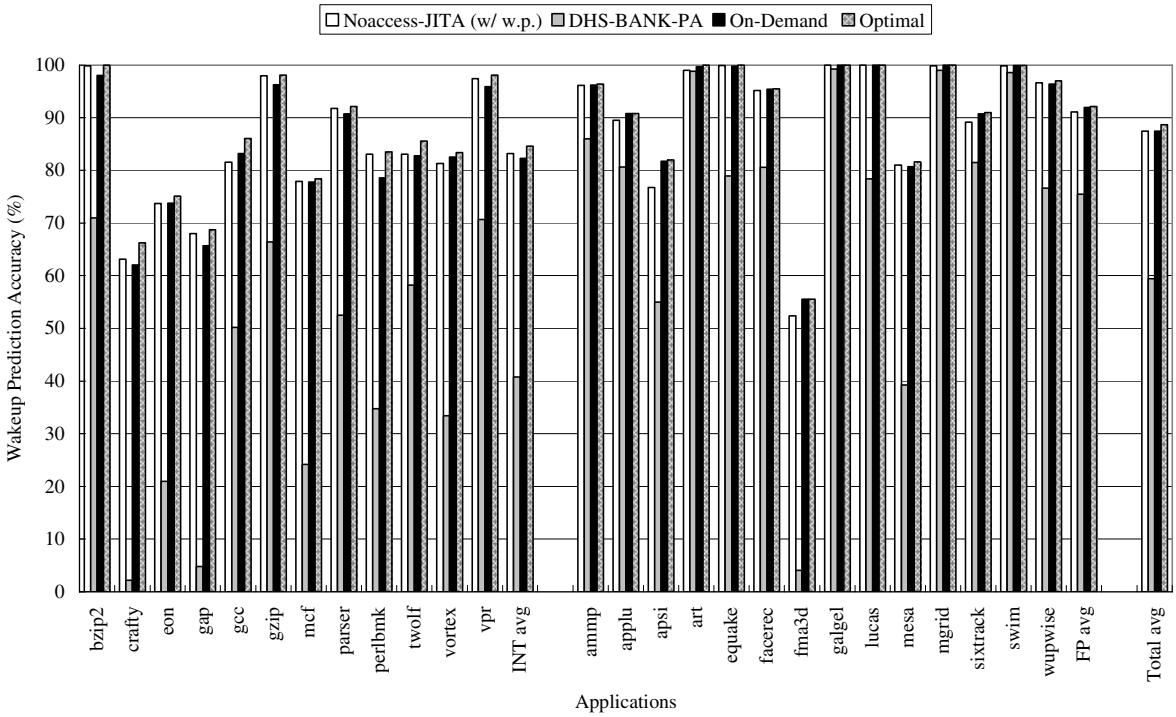

Fig. 12. Wakeup prediction accuracy per fetch, Including bitline precharging and way prediction accuracy (4-way set-associative)

Figure 12 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 (utilizing w.p.) is 97.5%, since a set-associative cache make it more difficult to predict sub-bank precharging. The on-demand and the optimal policy show 87.5% and 88.7% accuracy, respectively, where there is little difference. The

accuracy of DHS-Bank-PA is as low as 59.5%, on average, which might result in severe performance degradation. This is caused by flushing the previous sub-bank when execution jumps from one sub-bank to another, since the sub-bank hoppings are much more frequent in a set-associative cache.

D. Run-time Increase

Even a small increase in run-time leads to substantial increase in total processor energy consumption. This might outweigh the reduced L1 instruction cache leakage, because increased runtime extends the power dissipation of the entire processor (Note that, 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. This means that the wakeup prediction accuracy is not quite proportional to the run-time.

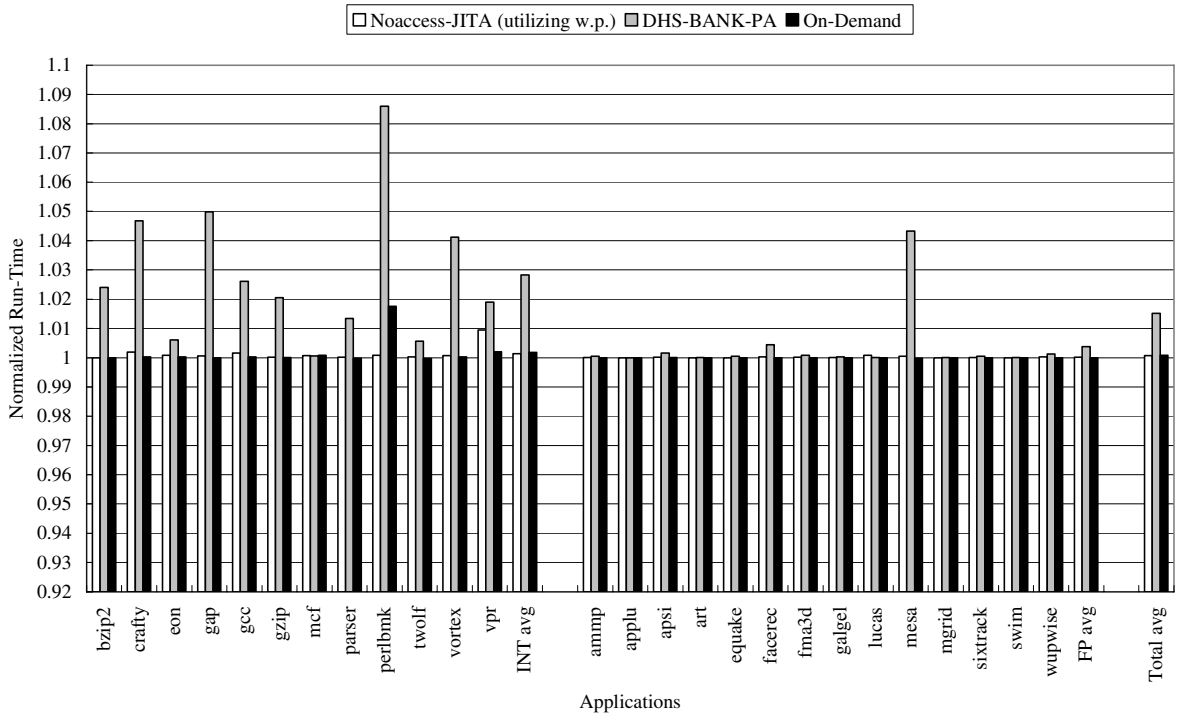

Fig. 13. Normalized run-time (direct-mapped)

Figure 13 shows the run-time normalized to the base model in the direct-mapped cache. The

noaccess-JITA (utilizing w.p.) increases run-time by 0.08%, on average, but recall that leakage reduction is only 70.0%–much smaller than the other policies. The DHS-Bank-PA shows 1.5% run-time increase, on average. Even worse, in some applications, run-time is increased by more than 8%. The proposed on-demand policy increases the run-time by only 0.08% on average. In perlbmk, the run-time of the on-demand policy is increased by 1.7 %. This means that wakeup mispredictions are not overlapped with cache misses in the on-demand policy.

Figure 14 shows the run-time normalized to the base model in the 4-way set-associative cache. The increases of average run-time are 0.11%, 3.33%, and 0.07% for noaccess-JITA (utilizing w.p.), DHS-Bank-PA, and the proposed on-demand policy, respectively. In crafty, the DHS-Bank-PA degrades the performance as much as 17.3%, which is especially severe.

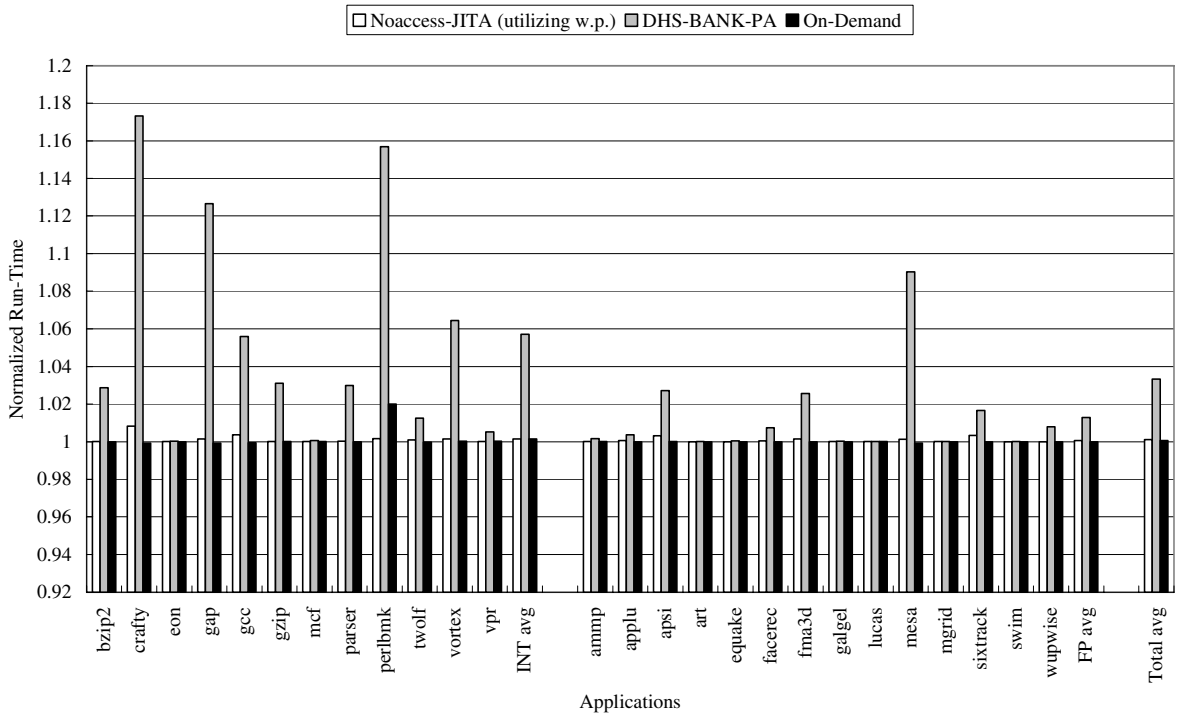

Fig. 14. Normalized run-time (4-way set-associative)

E. Total processor energy and ED²

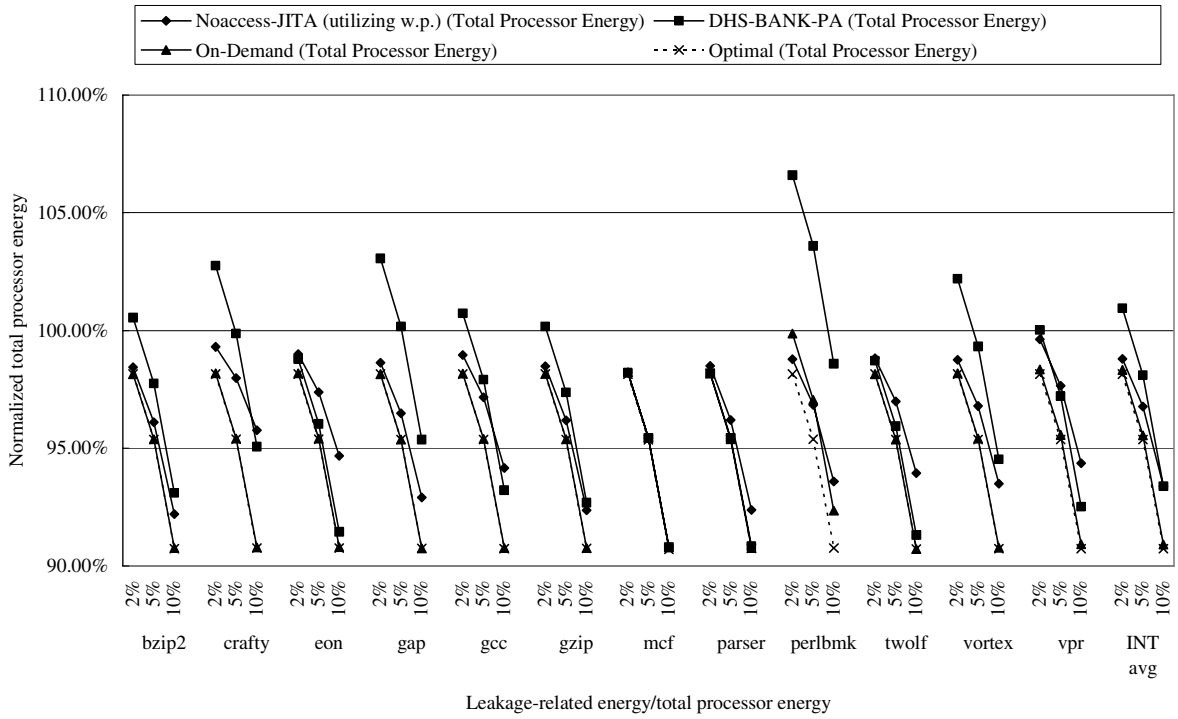

Fig. 15. Total processor energy for SPEC INT applications (direct-mapped)
(Note that the "on-demand" graphs are almost overlapped with the "optimal" graph)

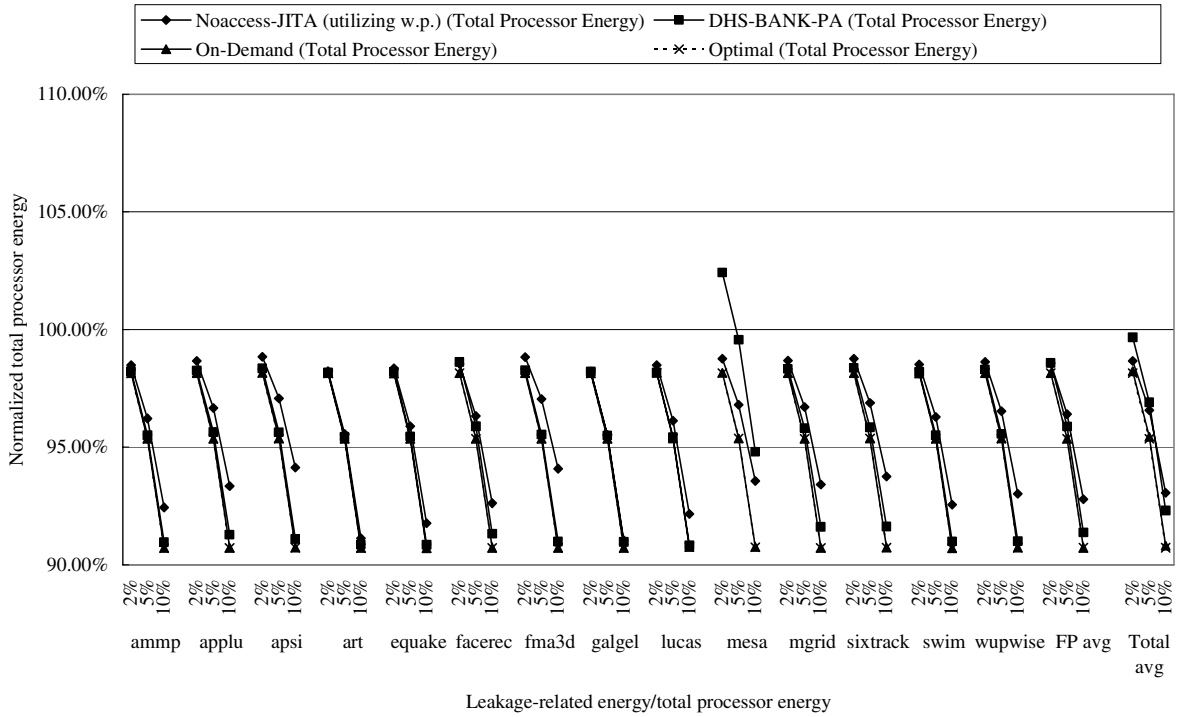

Fig. 16. Total processor energy for SPEC FP applications (direct-mapped)
(Note that the "on-demand" graphs are almost overlapped with the "optimal" graph)

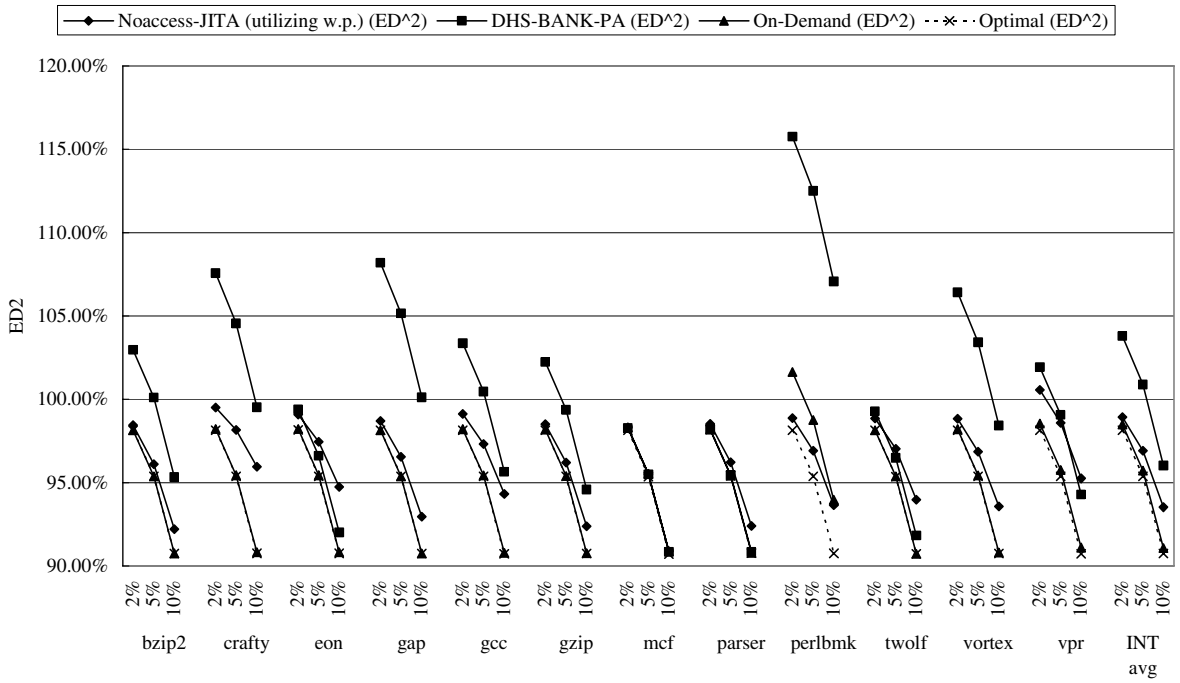

Fig. 17. ED^2 for SPEC INT applications (direct-mapped)
(Note that the "on-demand" graphs are almost overlapped with the "optimal" graph)

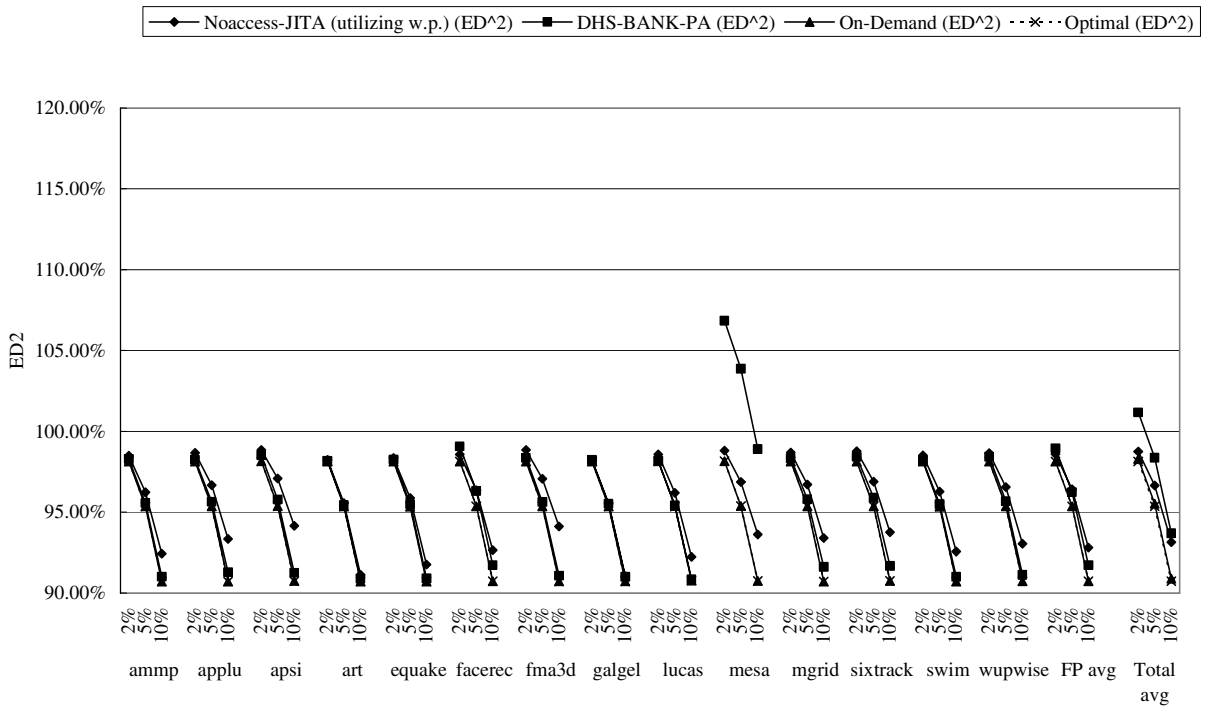

Fig. 18. ED^2 for SPEC FP applications (direct-mapped)

(Note that the "on-demand" graphs are almost overlapped with the "optimal" graph)

Figures 15-18 show average total processor energy and average ED^2 (Energy*$Delay^2$) for all SPEC2000 applications in the direct-mapped cache, including the cost of increased run-time 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 formula (1) of Section IV. A. The total processor energy includes the dynamic leakage-energy of the instruction cache as well as the dynamic and leakage energy consumed in other processor components except the instruction cache. For example, suppose that the instruction cache accounts for 10% of the total processor energy and leakage comprises 50% of the instruction cache energy, the ratio (leakage-related energy)/(total processor energy) is 5%.

Since the on-demand policy and the optimal policy show similar performance and leakage-related energy reduction as shown in the previous sub-sections, the total energy and ED^2 are also similar, regardless of 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 run–time (1.7%) 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 w.p.) is only 0.5%, when (leakage-related energy)/(total processor energy) is 2%. 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 L2 cache, the proposed on-demand policy is much more energy-efficient.

Although Section V. B indicated that DHS-Bank-PA reduces the leakage-related energy about

as well as on-demand, the total processor energy of the DHS-Bank-PA is largest among the three policies due to its severely increased run-time. Even worse, for a (leakage-related energy)/(total processor energy) ratio of 2%, the ED^2 of the DHS-Bank-PA is worse than the base model with no leakage-saving mode.

In the 4-way set-associative cache, the DHS-Bank-PA has larger total processor energy and ED^2 due to the more increased run-time compared to the direct-mapped cache. The other graphs are almost identical to the graphs in 15, Figure 16, Figure 17, and Figure 18 (We do not show the total processor energy and ED^2 for the 4-way set-associative cache).

The on-demand policy does incur the extra energy of an extra pipeline register due to the additional wakeup 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 (approx. *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 0.1W at 1.0V and 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 bit width and number of words/entries, and observed that writes are about 20% 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 w.p.) where at least additional 1K bits are required for wakeup prediction (Details are explained in Table V). The DHS-Bank-PA already shows noticeable differences in terms of the total processor energy and the ED^2, compared to the

on-demand policy. After considering additional 11K bits for wakeup prediction (Details are explained in Table V) that is much more than the extra hardware (400 bits) in the on-demand policy, the difference becomes much larger. The on-demand policy becomes therefore much more energy-efficient, after reflecting the extra energy overhead of the additional hardware required.

F. Combining with Other Leakage Saving Circuit Techniques

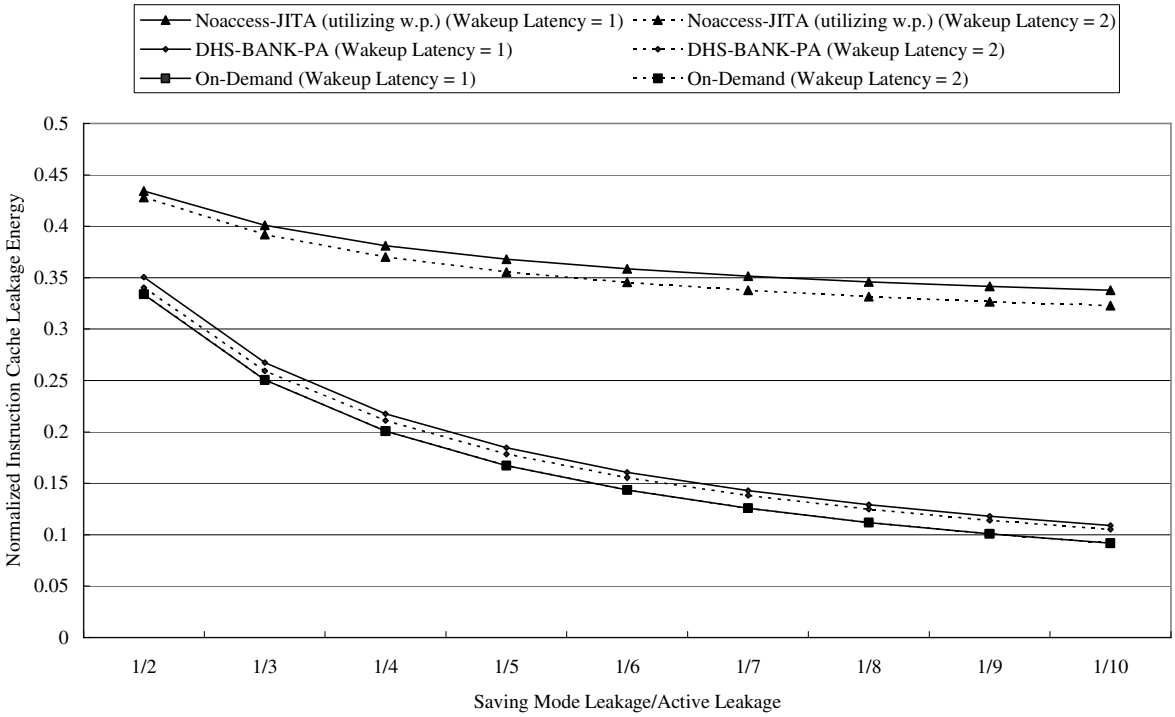

Fig. 19. Instruction cache leakage reduction combined with other leakage saving circuit techniques
(Note that the "on-demand" graphs are almost overlapped with the "optimal" graph)

The proposed on-demand policy can be adopted for other leakage saving circuit techniques that might have longer wakeup penalty or more leakage saving. Even for other techniques, such as ABB MTCMOS [19] and DRG [1] that have longer wakeup penalty, the proposed on-demand policy can be used to reduce leakage energy. Figure 19 makes it possible to estimate how much leakage energy is reduced, when other leakage saving circuit techniques are applied. We

normalized average instruction cache leakage energy to the base model for SPEC2000 applications, where the leakage energy due to extra run-time is captured but the energy consumption from the other processor components except the instruction cache and the dynamic energy consumption in the instruction cache is not considered (Please note that "on-demand (1 wakeup latency)" graphs are almost overlapped with the "on-demand (2 wakeup latency)" graph).

Regardless of the wakeup 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 wakeup latency is changed into two, the on-demand policy still reduces more leakage energy than the other policies.

G. Comparison of Hardware Overhead

For a wakeup prediction policy, hardware overhead is inevitable in additional to the DVS control circuitry. Noaccess-JITA (utilizing w.p.) 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 2-read port way predictor (instead of the baseline 1-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 [6]. Since the BTB has 1024 entries, the total storage overhead is 10K. For the proposed on-demand policy, a small register (ex. 10 bit for our 1024-entry cache) that records the most recently accessed cache line, and another pipeline register (400 bits) are needed. If the alternate path is woken up speculatively, as in Figure 4, a 2-read port way predictor is required. Otherwise, for the Figure 3 option that is more common [28], the baseline 1-read port way predictor is enough. Table V 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.

TABLE V
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)

VI. CONCLUSIONS AND FUTURE WORK

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. 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 run-time severely. In some cases, it increases the run-time by more than 15%. The proposed policy degrades the performance by only 0.06~0.08%, on average, and 2.0 % in the worst case. At the same time, active leakage energy is almost eliminated since only one cache line is active while all other lines are in the drowsy mode. The leakage-related energy reduction by the proposed policy is on average 92.7%, 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 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.

In this paper, we apply the on-demand wakeup 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 wakeup stage, leading to tag matching in the wakeup stage. This results in 100% 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 multi-cycle wakeup 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 can not hide the penalty due to its low wakeup prediction accuracy.

ACKNOWLEDGEMENT

This work was funded by US National Science Foundation under grant no. CCF-0429765, Army Research Office under grant no. W911NF-04-1-0288, a grant from Intel MRL, an IT National Scholarship Program from IITA & MIC, Korea, and a Korea Research Foundation Grant of the Korean Government (KRF-2006-D00452). We would like to thank Karthik Sankaranarayanan for his help in using SimPoint and in estimating the extra energy consumption of an additional pipeline stage. We would also like to thank Nam Sung Kim for his helpful comments on validating our simulation model. Finally, we would like to thank the anonymous reviewers for their helpful feedback.

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] S.W. Chung and Kevin Skadron. Using Branch Prediction Information for Near-Optimal I-Cache Leakage. Asia-Pacific Systems Architecture Conference, 2006, pp. 24-37.
[4] 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.
[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 Transaction on VLSI Systems, vol. 10, April 2002, pp. 91-95.
[6] 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.
[7] 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.
[8] R. Kessler. The Alpha 21264 Microprocessor. IEEE Micro Magazine. 1999, pp.24-36.
[9] 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.

[10] 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.
[11] 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.
[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. of Int. Conf. on Parallel Architectures and Compilation Techniques, 2002, pp.131-140.
[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. of the Design Automation and Test in Europe Conference. 2004, pp. 22-27.
[14] Y. Li, M. Hempstead, P. Mauro, D. Brooks, Z. Hu, and K. Skadron. Power and Thermal Effects of SRAM vs. LatchMux Design. Proc. of the ACM/IEEE 2005 International Symposium on Low-Power Electronics Design (ISLPED), 2005, pp. 173-178.
[15] 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.
[16] S. McFaring. Combining Branch Predictors. Technical Note TN-36. DEC June 1993.
[17] Y. Meng, T. Sherwood and R. Kastner. On the Limits of Leakage Power Reduction in Caches. Proc. of Int. Symp. on High-Performance Computer Architecture. 2005.
[18] M. Milenkovic, A. Milenkovic, and J. Kulick. Demystifying Intel Branch Predictor. Proc. of Workshop on Duplicating, Deconstucting and Debunking (in conjunction with ISCA-29). 2002.
[19] 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.
[20] 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.
[21] 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.
[22] E. Rotenberg, S. Nennett, and J. E. Smith. A Trace Cache Microarchitecture and Evaluation. IEEE Transaction on Computers. vol.48, no. 2, Feb. 1999, pp.111-120.
[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. Proc. of ASPLOS-X, 1997.
[24] S. H. Shin, S. W. Chung, and C. S. Jhon. On the Reliability of Drowsy Instruction Caches. Proc. of Asia-Pacific Computer Systems Conference, 2006, pp. 445-451.
[25] S. Yang and B. Falsafi. Near-Optimal Precharging in High-Performance Nanoscale CMOS Caches. Proc. of Int. Symp. on 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. of Int. Symp. on High-Performance Computer Architecture, 2001, pp.147-157.
[27] 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.
[28] ARM. ARM 1136 Technical Reference Manual. Available in http://www.arm.com
[29] ITRS (International Technology Roadmap for Semiconductor). 2001, Available in http://public.itrs.net.
[30] Standard Performance Evaluation Corp.. Available in http://www.specbench.org.
[31] VAR Business, Intel Clears up Post-Tejas Confusion, Available in http://www.varbusiness.com/sections /news/breakingnews.jhtml?articleId=18842588

Sung Woo Chung received the BS degree in Computer Engineering, the PhD degree in Electrical and Computer Engineering from Seoul National University, Seoul, Korea in 1996 and 2003, respectively. He worked as an academic visitor for IBM T.J. Watson Research Center, Yorktown Heights, NY, in 2002. From 2003 to 2005, He worked for Samsung Electronics as a senior engineer. In 2005, He worked as a research scientist in University of Virginia, Charlottesville. He joined the Division of Computer and Communication Engineering, Korea University, Seoul, Korea, 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.

Kevin Skadron received the BSEE 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, University of Virginia, Charlottesville, in 1999 and is now an associate professor. Skadron's research interests focus on the implications of technology trends and physical constraints (e.g. power, temperature, reliability) for future, highly multicore architectures. He is a co-founder and Associate Editor- In-Chief of IEEE COMPUTER ARCHITECTURE LETTERS. He recently served as program or general co-chair of PACT 2006, MICRO 2004, and PACT 2002. Dr. Skadron was coauthor of Best Student Paper Awards at ISCA 2003, RTSS 2003, and SHAMAN 2002.