Implementing Branch Predictor Decay using Quasi-Static Memory Cells

Philo Juang[†], Kevin Skadron[‡], Margaret Martonosi[†],
Zhigang Hu[+], Douglas W. Clark[†], Philip W. Diodato[*], Stefanos Kaxiras[*]

[†]Departments of Electrical Engineering and Computer Science, Princeton University
[‡]Department of Computer Science, University of Virginia
[*]Agere Systems, [+]IBM T.J. Watson Research Center

Abstract

With semiconductor technology advancing toward deep submicron, leakage energy is of increasing concern, especially for large on-chip array structures such as caches and branch predictors. Recent work has suggested that even larger branch predictors can and should be used in order to improve microprocessor performance. A further consideration is that more aggressive branch predictors, especially multiported predictors for multiple branch prediction, may be thermal hot spots, thus further increasing its leakage. Moreover, as the branch predictor holds state that is transient and predictive, elements can be discarded without adverse effect. For these reasons, it is natural to consider applying *decay* techniques—already shown to reduce leakage energy for caches—to branch-prediction structures.

Due to the structural difference between caches and branch predictors, applying decay techniques to branch predictors is not straightforward. This paper explores the strategies for exploiting spatial and temporal locality to make decay effective for bimodal, gshare, and hybrid predictors, as well as the branch target buffer. Furthermore, the predictive behavior of branch predictors steers them towards decay based not on state-preserving, static storage cells, but rather quasi-static, dynamic storage cells. This paper will examine the results of implementing decaying branch predictor structures with dynamic–appropriately, *decaying*–cells rather than the standard static SRAM cell.

Overall, this paper demonstrates that decay techniques can apply to more than just caches, with the branch predictor and BTB as an example. We show decay can either be implemented at the architectural level, or with a wholesale replacement of static storage cells with quasi-static storage cells which naturally implement decay. More importantly, decay techniques can be applied and should be applied to other such transient and/or predictive structures.

I. Introduction

As microprocessors have matured, power and energy concerns, once only considered in low-power embedded and mobile devices, have become a major concern in general-purpose and high-performance processors. *Dynamic* power dissipation due to switching activity has received a great deal of attention as circuit designers and architects have developed ways to reduce switching activity and capacitive loads. In ideal devices, dynamic power is the only source of dissipation, as current only flows during switching. Yet real devices also dissipate *static* or *leakage* power due to currents that flow regardless of activity, especially *subthreshold* leakage currents [25] that flow even when the transistor is nominally off.

For questions or comments, please e-mail pjuang@princeton.edu

As progress in fabrication processes has led to steady reductions in supply voltages, threshold voltages are being lowered to the point where leakage has become an important and growing fraction of total power dissipation in high-performance CMOS CPUs. If it is not addressed through fabrication or circuit-level changes, some forecasts predict as much as a five-fold increase in leakage energy per technology generation [1]. At such rates, the current leakage component, roughly 5% of total chip power now, would balloon to 50% or more in just a few generations.

In response to this trend, researchers have proposed circuit- and architecture-level mechanisms for managing leakage energy [20], [24], [42]. In particular, prior work by Kaxiras *et al.* on cache decay techniques [24] showed that turning cache lines off if they have not been used in a long time can be effective in reducing leakage energy with little performance impact.

After caches, branch predictors are among the largest and most power-consuming array structures in current CPUs. Current predictors are 4–8 Kbytes in size, already the size of a small cache. They dissipate about 10% of the processor's total dynamic power dissipation [33]. Cycle-time, power-dissipation, and thermal concerns tend to keep predictors from growing larger.

However, Jiménez *et al.* [21] pointed out that two-level predictors can avoid cycle-time constraints and that large second-level predictors can give substantial increases in prediction accuracy, resulting in predictors that could be as large and have the same substantial leakage as first-level caches. Futhermore, Skadron *et al.* [36] found that a very aggressive branch predictors may be thermal hot spot, as it is typically accessed every cycle. This indicates that branch predictors may expend much more leakage energy than their sizes would suggest, because leakage power increases exponentially with temperature. Furthermore, as the ratio of static to dynamic power continues to increase, regulating leakage becomes more important as a strategy to regulate operating temperatures.

Applying decay techniques to caches has proven effective, so applying decay techniques to branch predictors is an obvious next step. Unfortunately, several factors complicate this task, for it is much less obvious when a branch predictor entry may be considered "dead" and can therefore be turned off with little performance impact. First, many branches may map to the same predictor entry. Since this sharing is sometimes beneficial, notions of cache conflicts and eviction do not translate directly into the branch prediction world. Second, a branch predictor entry is not simply valid or invalid, as in a cache. A branch predictor entry may have reached the "strongly not taken" state due to the effects of several different branches and may be useful to the next branch that accesses it, even if this branch has never been executed before. Third, branch predictor entries are too small to deactivate individually, so one must consider some larger collection, such as a row of predictor entries in the square array in which the predictor is likely implemented. The challenge here is that unlike the grouping of data into a cache line, the grouping of branch predictor entries in a row is not something for which application programmers and systems builders have a sense of spatial locality. This paper evaluates design options related to these questions.

Further interesting questions arise when moving from simple *bimodal* branch predictors [37], which keep one two-bit counter per predictor entry, to multi-table predictors like *hybrid* predictors [31], which operate several prediction structures in parallel. For example, hybrid predictors may encounter instances when one of the predictor components has been deactivated but the other has not. The chooser might be designed to pick the still active component in such situations. For other branches, the chooser may exhibit a strong bias for one predictor component over the other. In this case, predictor entries that are not being selected might be deactivated.

Traditionally, state preserving structures have been designed with 6T SRAM cells; unlike DRAM cells, SRAM cells

retain their value while charged. By doing so, caches need not have built in circuitry to refresh the cells as seen in main memory. With this in mind, decay and similar techniques focus on somehow shutting down these cells; one such technique involves gating either the source or drain voltages to a signal that is enabled whenever the cell needs to be turned off. By attaching decay counters to each cell, one can detect if the cell has gone long ”enough” without activity. If this is the case, the cell is assumed to be unused and can be deactivated.

Traditional techniques of deactivating cells involved attaching a gate transistor to the supply voltage V_{dd}; when the cell is to be deactivated, the gate transistor disconnects the supply to the rest of the cell and thus the cell is effectively turned off [42]. Gated-V_{dd} based cache decay techniques have a few problems, however. First is the area requirement. Attaching a gate transistor to each cache line might have an adverse impact on the pitch of the cache line, increasing the overall size of the cache. Furthermore, the counters themselves consume dynamic and static power, which must be taken into account as well. Moreover, the realities involving gating the source or drain voltage have been previously implied but not fully understood. One cannot assume that powering up or powering down a transistor comes for free; instead, there is some dynamic cost to the process roughly of the same order as a cache line access.

An alternative to traditional decay is to use a quasi-static, 4-transistor memory cell referred to as the 4T cell. 4T cells are as fast as 6T SRAM cells, but do not have connections to the source voltage (V_{dd}). Rather, the 4T is charged upon each access, whether read or write, and slowly leaks the charge over time. When the charge in the cell has been depleted, the value stored is lost. Thus, they may be referred to as quasi-static; like conventional DRAM, they leak their charge, but are not subject to a full read-write-refresh cycle.

4T cells originally have been built into embedded processors, where low-power consumption is the most important aspect. They have been largely overlooked in general-purpose processors because, as mentioned above, they are not truly static. Rather, they require periodic refresh. However, in decay techniques, where the goal is to exploit the transient nature of short term storage by turning them on and off depending on usage, one can actually take advantage of 4T cells’ dynamic nature. Decaying a 6T cell seems somewhat like squashing its greatest advantage – that it retains its value for however long it is powered up. Implementing decay with 4T cells is therefore obvious; as it decays naturally over time if not accessed, this distinctive behavior fits perfectly with the transient behavior of elements in a cache or branch predictor.

Decay with quasi-static cells is advantageous to conventional decay in several respects. First, by virtue of two fewer transistors and by eliminating the need to run power rails lengthwise throughout the array, a 4T based cache is smaller than a 6T cache. Furthermore, as 4T cells decay and revive naturally through the read-write process, some of the costs behind activating decayed cells are eliminated.

Contributions

Because branch predictors have behavior more nuanced than caches, cache decay methods cannot be directly applied. This paper shows that effective decay techniques can nevertheless be devised for branch predictors. The paper then goes on to explore the interaction of decay policies with some of the wealth of branch predictor design parameters. We show that:

- Decay can reduce net leakage energy in the conditional branch predictor by 40–60% and the branch target buffer (BTB) by 90%.
- In hybrid predictors, decay policies can achieve 50% higher reductions in leakage energy if the decay policy takes advantage of the hybrid predictor organization to boost decay opportunities.

- Decay is most effective for intervals of 64K cycles or larger. If decay is applied too aggressively, extra mispredictions result with significant costs in both performance and dynamic power.

This paper first investigates the use of decay in several types of branch predictors. We present the results of implementing decay in simple branch predictors, and we suggest and evaluate different policies in complex branch predictors. We then introduce the 4T cell and examine its usage in simple caches and branch predictors.

II. Related Work

Much research has recently been done in the field of static leakage, both at the architectural level and at the device level. At the architectural level, prior work focused on improving the static leakage performance by shutting off (decaying) unused portions of large array-like structures, for example, in caches. Yang *et al.* [42] described an instruction cache organization that disables ways of a set-associative cache to match the capacity currently needed by the executing program.

Kaxiras *et al.* [24], Hu *et al.* [18], and Zhou *et al.* [44] describe techniques for disabling individual cache lines by inferring that lines which have not been used in a long time have *decayed* : they probably contain data that is not likely to be used again before replacement. Building on the cache decay work, Hu *et al.* [15] extended decay techniques to the branch predictor. Most of the above techniques assume technologies such as gated-V_{dd} to control whether or not a portion of the structure is enabled or disabled. More recently, Hanson *et al.* presented a comparison of leakage-control techniques [12], and concluded that gated-V_{dd} may not be the best approach for controlling static power in large data arrays. Instead, they find that dual-V_t techniques [34] save more overall energy for structures that should have fast lookup times (like first-level caches).

Taking a different approach, Juang *et al.* [22] looked at eliminating V_{dd} altogether in replacing the standard SRAM cell with a dynamic, quasi-static four transistor cell. Using the quasi-static cell allows for an easily built, naturally decaying cache. In addition to caches, Hu *et al.* [16] described ways of saving leakage in branch predictors by implementing them with quasi-static cells. Furthermore, using adaptive cache decay as an example, Velusamy *et al.* [40] applied formal feedback control mechanisms to improve leakage savings by implementing an adaptive controller which varied decay intervals dynamically. Finally, Hu *et al.* [19] included cache decay as an example of improving power dissipation and performance by looking at more than just simple time-independent behavior, such as event ordering, and exploiting the generational behavior of structures such as the cache and prefetch unit.

Alternately, Flautner *et al.* [9] presented a technique in which to improve leakage savings not by fully decaying cache lines, but rather to put the in a low-power ”drowsy” state, which maintains the cache state rather than discarding it as in decay techniques. Another leakage control technique saving the cache state was described by Heo *et al.* [13]. Combining state-preserving and state-destroying techniques, Li *et al.* [27] demonstrated a technique in which leakage energy was further reduced by exploiting data duplication across the cache hierarchy. Lastly, focusing instead on the compiler, Zhang *et al.* [43] presented techniques using the compiler to improve instruction cache leakage.

III. Experimental Setup

This section describes our simulation technique and benchmarks, and our method for evaluating energy savings.

A. Simulation Setup

Simulations in this paper are based on the SimpleScalar 3.0 toolkit [3]. Our model processor has microarchitectural parameters that resemble in most respects the Intel PIII processor [5]. The main processor and memory hierarchy parameters are shown in Table I. For performance estimates and behavioral statistics, we use SimpleScalar's *sim-outorder* simulator. For energy estimates, we use the Wattch simulator [2]. Wattch uses SimpleScalar's *sim-outorder* cycle-accurate model and adds cycle-by-cycle tracking of power dissipation by estimating unit capacitances and activity factors. Because most processors today have pipelines longer than 5 stages, both our architectural and power models extend the sim-outorder pipeline by adding three additional stages between decode and issue.

Processor Core	
Instruction Window	40-RUU, 16-LSQ
Issue width	4 instructions per cycle
Functional Units	4 IntALU,1 IntMult/Div, 4 FPALU,1 FPMult/Div, 2 MemPorts
Memory Hierarchy	
L1 D-cache Size	32KB, 1-way, 32B blocks, 3-cycle latency
L1 I-cache Size	16KB, 4-way, 32B blocks, 3-cycle latency
L2	Unified, 256KB, 8-way LRU, 32B blocks,8-cycle latency, WB
Memory	100 cycles
TLB Size	128-entry, 30-cycle miss penalty

TABLE I
CONFIGURATION OF SIMULATED PROCESSOR.

B. Technology Assumptions and Circuit Simulations

The results of this research, while broadly applicable, are evaluated based on particular technology assumptions. We chose a feature size of 0.16μ, a V_{dd} of 1.5V, and a clock speed of 1 GHz. We use 6T and 4T cells from cell libraries; no custom designs are assumed. 4T cells exist in these libraries because of their possible use as DRAM cells embedded onto a primarily-logic chip.

Process technology primarily determines leakage currents. With each successive generation, leakage increases substantially. In particular, Figure 1 illustrates this progression by illustrating leakage currents (in nA) for four industry process technologies currently in use at Agere Systems. At 2.5V and 0.25um, COM1 is largely out of date; we do not consider it further. COM2 is a reasonably current CMOS process, with a 1.5V supply voltage and 0.16um feature size. COM3 and COM4 are slightly more forward-looking CMOS processes. COM3 is a 1.0V, 0.12um process, and COM4 is a 1.0V, 0.1um process.

The leakage currents in this figure are shown for room temperature (25°C). Although this understates the magnitude of the leakage seen in operating chips where the temperature is likely to be much higher, our data agrees with other predictions [1] that leakage is growing exponentially with successive technology generations.

Our studies are based on the currently-in-production COM2 process. We have accurate transistor models for this process, and some of our results are obtained via detailed circuit simulations. We use Celerity tools for very detailed circuit simulations with a 25 pico-second resolution. Although leakage savings with COM2 are small, COM2 is a process for which we can gather verifiable simulation results. As such, it is a useful vehicle for detailed circuit studies

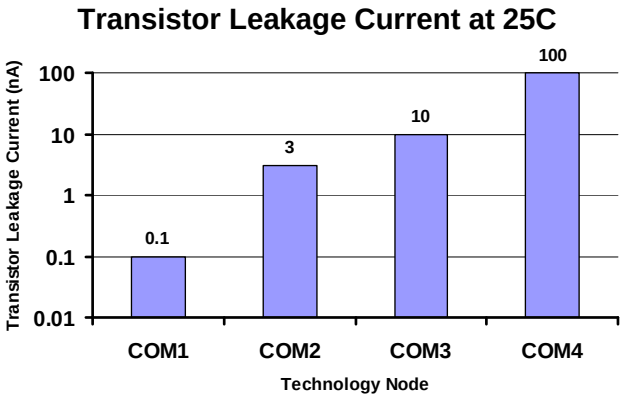

Fig. 1. Leakage current per transistor for a range of technologies in use at Agere.

of hold times and temperature effects. Leakage current will be a serious problem in subsequent generations, and this paper shows substantial savings for the COM3 and COM4 processes.

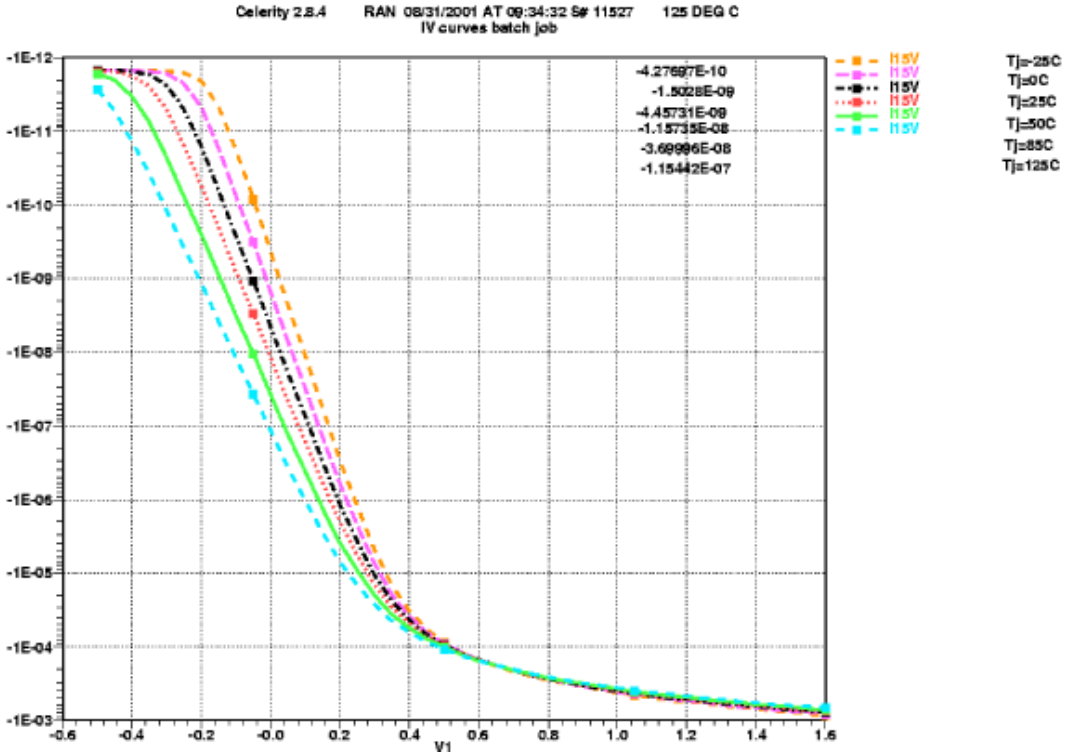

Fig. 2. Current-Voltage curves for COM2 transistors at varying temperatures.

An important factor affecting leakage is temperature. Figure 3 shows transistor leakage currents as temperature is varied for 1.5V 0.16u COM2 transistors. The further exponential relation of leakage to temperature is evident in these figure. Variations in temperature result in large variations in leakage, and these will impact design tradeoffs. Our designs target an operational temperature of 85 C (appropriate for example for high-performance or mobile processors) but we also simulated very high temperature (125 C) operation (appropriate in hard to control environments like automobiles).

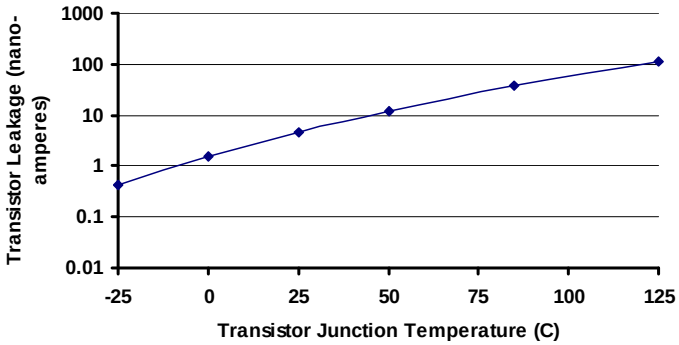

Fig. 3. Transistor leakage current (nA) for varying temperatures for the COM2 process.

C. Benchmarks

We evaluate our results using benchmarks from the SPEC CPU2000 suite [39]. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings and include all linked libraries. For each program, we skip the first 1 billion instructions to avoid unrepresentative behavior at the beginning of the program's execution. We then simulate 200M (committed) instructions using the reference input set. Simulation is conducted using SimpleScalar's EIO traces to ensure reproducible results for each benchmark across multiple simulations. Table II summarizes the benchmarks used and their prediction accuracies with several predictors.

	Dynamic Conditional Branch Frequency	Prediction Rate w/ Bimod 4K	Prediction Rate w/ Gshare 16K	Prediction Rate w/ Hybrid 21264-style
gzip	9.40%	87.58%	90.67%	92.40%
vpr	11.08%	90.00%	97.68%	97.03%
gcc	2.56%	99.61%	99.74%	99.75%
mcf	19.40%	98.42%	99.27%	98.94%
crafty	11.13%	91.76%	93.33%	94.38%
parser	15.60%	91.25%	94.31%	94.89%
eon	11.08%	81.03%	89.71%	91.76%
perlbmk	12.43%	95.15%	97.29%	97.60%
gap	6.62%	90.10%	96.50%	96.96%
vortex	16.00%	97.85%	97.61%	97.87%
bzip2	12.13%	94.06%	94.05%	94.12%
twolf	12.24%	86.44%	88.53%	88.48%
wupwise	10.50%	92.05%	97.54%	96.66%
swim	1.35%	99.36%	99.54%	99.55%
mgrid	0.33%	92.57%	97.77%	97.95%
applu	0.28%	93.07%	98.70%	98.21%
mesa	8.73%	94.09%	96.98%	96.31%
galgel	6.09%	99.13%	99.27%	99.29%
art	11.29%	92.99%	99.02%	96.40%
equake	17.13%	98.04%	99.39%	99.28%
facerec	3.49%	97.92%	99.04%	99.21%
ammp	21.67%	98.77%	99.27%	99.23%
lucas	8.67%%	90.84%	98.70%	98.84%
fma3d	18.09%	95.93%	98.39%	97.68%
sixtrack	8.08%	89.58%	99.72%	99.78%
apsi	3.51%	86.22%	96.52%	97.12%

TABLE II
BENCHMARK SUMMARY.

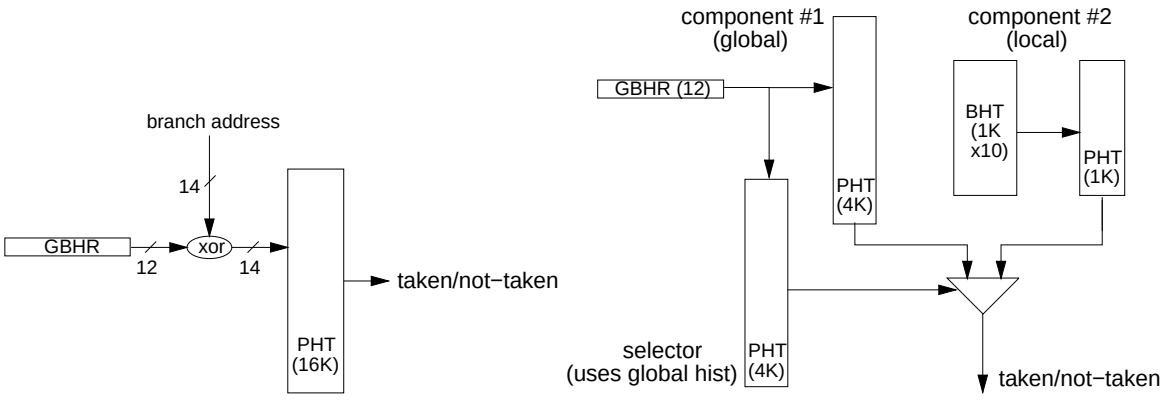

Fig. 4. Gshare predictor in the Sun UltraSPARC-III (Left) and 21264-style hybrid predictor (Right).

D. Energy Evaluation

An evaluation of the net energy savings from branch predictor decay must account for two opposite effects resulting from decay. On the one hand, turning off V_{dd} in idle counters or rows can prevent them from leaking; on the other hand, decaying the counters causes them to lose history information stored with them, possibly degrading the performance of the branch predictor and hence possibly resulting in more energy spent on mis-speculation and longer run time.

We use Wattch [2] to measure the total processor dynamic energy with and without applying decay, and subtract them to obtain the dynamic energy overhead. speed of 1 GHz. For leakage power, Yang *et al.* [42] estimate that with a 110°C operating temperature, a 1GHz operating frequency, a supply voltage of 1.0V and a threshold voltage of 0.2V, a SRAM cell consumes about $1740 * 10^{-9}$ nJ leakage energy per cycle. This is roughly in line with industry data that Agere has found with their COM3 and COM4 process generations [8].

Based on the above data, we estimate that the 4K-entry bimodal predictor, the 16K-entry gshare predictor, and the 21264 style hybrid predictor consume about 0.014nJ, 0.056nJ, and 0.030nJ leakage energy each cycle, respectively. In general, the overall leakage energy for the branch predictor can be calculated as *leakage energy per bit per cycle * #bits * #cycles*. The leakage energy of the extra status bits can be calculated similarly. Since these status bits only switch infrequently (at most once every decay interval, *e.g.* every 64K cycles), we ignore their dynamic energy overhead in our calculation.

E. Branch Predictors Studied

Although a wealth of dynamic branch predictors have been proposed, we focus on the effects of decay for the gshare and hybrid predictors, because they present the most interesting tradeoffs.

The gshare predictor, shown in the left-hand portion of Figure 4, tries to detect and predict sequences of correlated branches by tracking a global history (the global branch history register or GBHR) of the outcomes of the N most recent branches. In gshare, the global branch history and the branch address are XOR'd to reduce aliasing. This paper models a 16 K-entry gshare predictor in which 12 bits of history are XOR'd with 14 bits of branch address. This is the configuration that appears in the Sun UltraSPARC-III [38].

Instead of using global history, a two-level predictor can track local branch history on a per-branch basis. Local history is effective at exposing patterns in the behavior of individual branches. Because most programs have some

branches that perform better with global history and others that perform better with local history, a hybrid predictor [4], [31] combines the two. It operates two independent branch predictor components in parallel and uses a third predictor—the *selector* or *chooser*—to learn for each branch which of the components is more accurate and chooses its prediction. This paper models a hybrid predictor with a 4K-entry selector that only uses 12 bits of global history to index its PHT; a global-history component predictor of the same configuration; and a local history predictor with a 1 K-entry, 10-bit wide BHT and a 1 K-entry PHT. This configuration appears in the Alpha 21264 [10] and is depicted in the right-hand portion of Figure 4.

Logically, branch predictors are arrays of counters that are typically just two bits wide. Physically, however, branch predictors are, like caches, implemented as square or nearly-square array structures. This helps to minimize the complexity of the row and column decoders and balance wordline and bitline length and delay. The predictor array is thus similar to a cache array, except that it needs no tags. For example, the 16K-entry gshare predictor discussed above can be laid out as a 128×128 array of 2-bit counters. Alternatively, it can be divided into 4 banks, each a 64×64 counter array. We refer to these two organizations as "unbanked" and "banked" respectively.

IV. Decay with Basic Branch Predictors

A. Overview of Proposed Implementations

Since branch counters are only 2 bits in size, a cost-effective choice for turning off these counters is at the granularity of rows in the array structure rather than individual entries. This requires only two bits of state per row (the reference bit and the active bit) and hence a total overhead of $2 * \sqrt{entries}$ bits. Our techniques have the following general structure. At regular intervals, all rows of predictor entries not been used during the interval are assumed to have *decayed* and are therefore *deactivated*. The interval, called the *decay interval*, is measured in processor cycles and is a critical parameter for these schemes. The shorter the interval, the more opportunities for rows to be deactivated but the more likely it is to deactivate rows prematurely and induce extra mispredictions. Intervals long enough to minimize extra mispredictions, on the other hand, result in the deactivation of fewer entries.

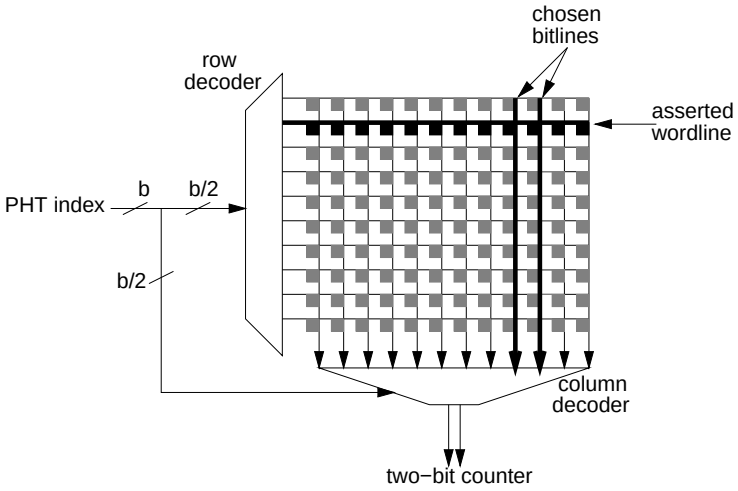

Fig. 5. A schematic of a squarified branch predictor table of two-bit counters (the PHT).

Figure 5 shows a picture of a typical branch predictor arranged as a square memory array. A group of entries, then, is a *row* of the square memory array. One bit per row, the *reference bit*, indicates whether any predictor entry in that

row has been accessed within the last decay interval. All reference bits are cleared at the end of each interval. A second bit for each row, the *active bit*, indicates whether that row is currently active (*i.e.*, not decayed). If a predictor lookup tries to access a decayed row, the predictor signals that a prediction cannot be made; the row is re-activated and possibly initialized to some desired starting state; in the meantime, a default prediction is made. Upon activation, our experiments use a default of not taken and initialize all the counters to 01. Thus, subsequent branches using the re-activated line start in the weakly-not-taken state.

The *active ratio* in a particular experiment is the average percentage of predictor rows found to be active (not decayed); it is a proxy for the actual leakage energy consumed by the predictor. Of course shorter decay intervals yield smaller active ratios (and larger leakage energy savings), but performance may suffer, since useful predictor entries are sometimes deactivated. Exploring this power-performance tradeoff is a key objective of this paper.

Logically, branch predictor structures have many entries but each entry is typically just two bits wide. When first considering applying decay techniques to branch predictors, it might be tempting to consider deactivating individual predictor entries. This is untenable, however, because our methods require two bits of state per independently-activated unit; such overhead would be excessive if applied to individual two-bit predictor entries.

To evaluate the net effectiveness of decay for reducing leakage energy, we combine the reduced value of leakage energy that was observed with decay, and the overhead energy associated with the decay technique. We then compare this to the original value for leakage energy. For each of the predictor types we study, we present plots of *normalized leakage energy* for different decay intervals, where the basis for normalization is the original value for leakage energy. This approach for measuring the net reduction in leakage energy is similar to the techniques used by Kaxiras *et al.* in their work on cache decay [24].

B. Spatial/Temporal Locality in Branch Predictors

The first question in exploring decay for branch predictors is to determine how often an entire row of branch predictor entries is likely to lie idle long enough for decay techniques to be effective. In today's machines, branch predictor rows typically include 32-256 counter entries. Fortunately, program branches are clustered rather than random, and across all the predictor organizations we examine, our experiments consistently show that some rows have heavy activity while others are idle and can be deactivated.

Clearly, programs exhibit spatial locality in the instruction cache. Over a short period of time, only one or a few small contiguous regions of the program are likely to be active, so branch instructions are likely to be close in terms of their PC. This also translates into spatial locality in branch-predictor accesses. For branch predictors, spatial locality means that at any point in the program, active rows are likely to have many counters active and idle rows are likely to be entirely idle. This is most true for the bimodal predictor, which is indexed only by PC. Indeed, the probability that two successive conditional branches fall into the same row in a 4 K-entry bimodal predictor is greater than 40% for all our benchmarks, and greater than 50% for all but five. If the branches were uniformly distributed, we would expect rates close to 1/*rows* (about 24 K-entry predictor array). For gshare or hybrid predictors, spatial locality is not as pronounced as for bimodal, since these other predictors are indexed by the branch history. Nevertheless, in half of the benchmarks the probability of hitting the same row as the previous branch is above 10%. This is much higher than a random distribution (about 1% for the large, 16 K-entry gshare), so these predictors also exhibit some spatial locality.

	1K cycles	10Kc	100Kc	1Mc	Overall
gzip	24	32	45	103	281
vpr	31	45	58	65	742
gcc	2	9	79	193	512
mcf	65	83	92	116	565
crafty	104	305	592	855	1701
parser	53	90	157	294	2265
eon	81	289	357	415	652
perlbmk	90	453	631	1112	1541
gap	62	281	325	576	745
vortex	124	502	1227	1642	1996
bzip2	22	33	45	56	460
twolf	48	210	300	334	351
wupwise	42	52	53	55	193
swim	3	6	11	15	687
mgrid	3	6	9	25	500
applu	1	2	4	7	579
mesa	83	114	139	267	697
galgel	2	6	8	10	508
art	2	2	5	18	109
equake	167	192	193	202	226
facerec	7	24	25	39	144
ammp	11	26	105	230	794
lucas	3	3	3	4	242
fma3d	80	450	452	465	499
sixtrack	39	49	55	99	734
apsi	14	85	117	125	342
geomean	20	46	70	109	529
max	167 (equake)	502 (vortex)	1227 (vortex)	1642 (vortex)	2265 (parser)
min	1 (applu)	2 (applu)	3 (lucas)	4 (lucas)	109 (art)

TABLE III

AVERAGE NUMBER OF STATIC BRANCHES TOUCHED EVERY SAMPLE INTERVAL FOR SPEC2000. THE RIGHTMOST COLUMN LABELED 'OVERALL' GIVES THE STATIC BRANCH FOOTPRINT FOR THE WHOLE SIMULATION PERIOD.

Yet this is not useful if the active rows change rapidly, so temporal locality is also necessary. One immediate factor that creates temporal locality is the fact that many benchmarks have small static branch footprints (the number of unique branch instruction sites that are executed), as seen in Table III. Decay will therefore clearly help bimodal predictors, because each static branch touches only one predictor entry and we know from the data in Table III that they are clustered.

Other predictor structures, however, may not do as well. With gshare, the branch address is XOR'd with the global branch history, so that one branch can touch many PHT entries. We evaluate decay for gshare predictors in section IV-D. Hybrid predictors use global- and local-history predictors as components, which brings more design choices. We explore the design space for hybrid predictors in section IV-E.

C. Decay with Bimodal Predictors

Figure 6 shows the active ratio and direction prediction accuracy for 4K-entry bimodal branch predictors with different decay intervals. We see from the active ratio graph that decay is very effective at shutting off idle counters in bimodal predictors. The geometric mean active ratio, which is the percentage of predictor entries powered on (so smaller values are better), is 37%, 28%, 22% and 18% for decay intervals of 4096K, 512K, 64K and 8K cycles respectively. This means that decay techniques have the potential to reduce branch predictor leakage by 2X or more.

Since bimodal predictors are indexed by PC, benchmarks with large static branch footprints tend to have higher active ratios, as shown in *crafty, perlbmk, gap and vortex*. Other benchmarks typically leave more than half of their counters deactivated due to their small footprints.

Furthermore, the prediction rate graph shows that shutting off these idle counters comes with minimal loss in performance except for the apparently too-aggressive decay interval of 8K cycles. With a 64K cycle decay interval, the overall loss in prediction accuracy is about 0.14%, with only one benchmark (*mgrid*) over 1%.

Interestingly, in some benchmarks (*wupwise* and *facerec*) we actually observed tiny improvements in prediction accuracy. We attribute this to the effect of removing some destructive interference. Interference occurs when two different branches map to the same counter. The interference is destructive when the branches are biased in opposite directions, for example, when one of the conflicting branches is strongly taken and the other is strongly not taken. Deactivation resets the counter to the neutral, weakly not taken value, which effectively isolates the two conflicting branches.

Figure 7 compares the leakage energy before and after applying decay. When decay is enabled, we add to the leakage energy the extra leakage energy from the decay status bits as well as any dynamic-energy overhead due to extra mispredictions or longer run time. We show the geometric mean reduction in leakage energy for SPEC2000 benchmarks. The figure demonstrates the trade-off between savings in leakage energy and the overhead incurred. With a small decay interval such as 8K cycles or less, branch prediction rate degrades enough so that the dynamic energy overhead dominates the savings in leakage energy. When the decay interval is longer, few mispredictions are induced, the extra dynamic energy incurred becomes minimal, and over 60% of the original leakage energy can be saved.

D. Decay With the Gshare Predictor

To understand whether decay is effective for a gshare predictor, it is helpful to measure, for various interval lengths, how many rows stay active for the duration of the interval. This is exactly measured by the active ratio. Smaller active ratios are better for decay. Figure 8 shows the geometric mean of the active ratios across the benchmarks for both banked and unbanked 16K-entry gshare predictors and, as a reference, the 4K-entry bimodal predictor. As expected, the active ratios are quite small (*i.e.*, good from a decay point of view) for the bimodal predictor. For gshare predictors, the active ratios are larger. Yet significant numbers of rows remain untouched. This indicates that even for predictor structures designed to smear branch addresses over many entries, decay-based techniques still show significant promise for addressing leakage concerns.

We include data in Figure 8 for a banked version of gshare. Breaking the predictor into banks makes the active ratio smaller (better for decay) by reducing the granularity over which activity is measured. Indeed, the active ratio for gshare is 15–35% smaller if it is broken into four banks of 4K entries each. Overall, these active ratios yield leakage energy savings of 40% for unbanked gshare and about 50% for banked gshare. Even greater savings can be achieved for structures directly indexed by PC: about 65% for bimodal predictors and 90% for the BTB. For more details, refer to [17].

Figure 9 further breaks down the data on a per-benchmark basis. It shows active ratio and branch misprediction rate for an unbanked gshare predictor. The geometric mean active ratio across the 26 benchmarks is 46% for a 64 K-cycle decay interval, and the average drop in predictor accuracy is negligible. Ten benchmarks stand out as having larger

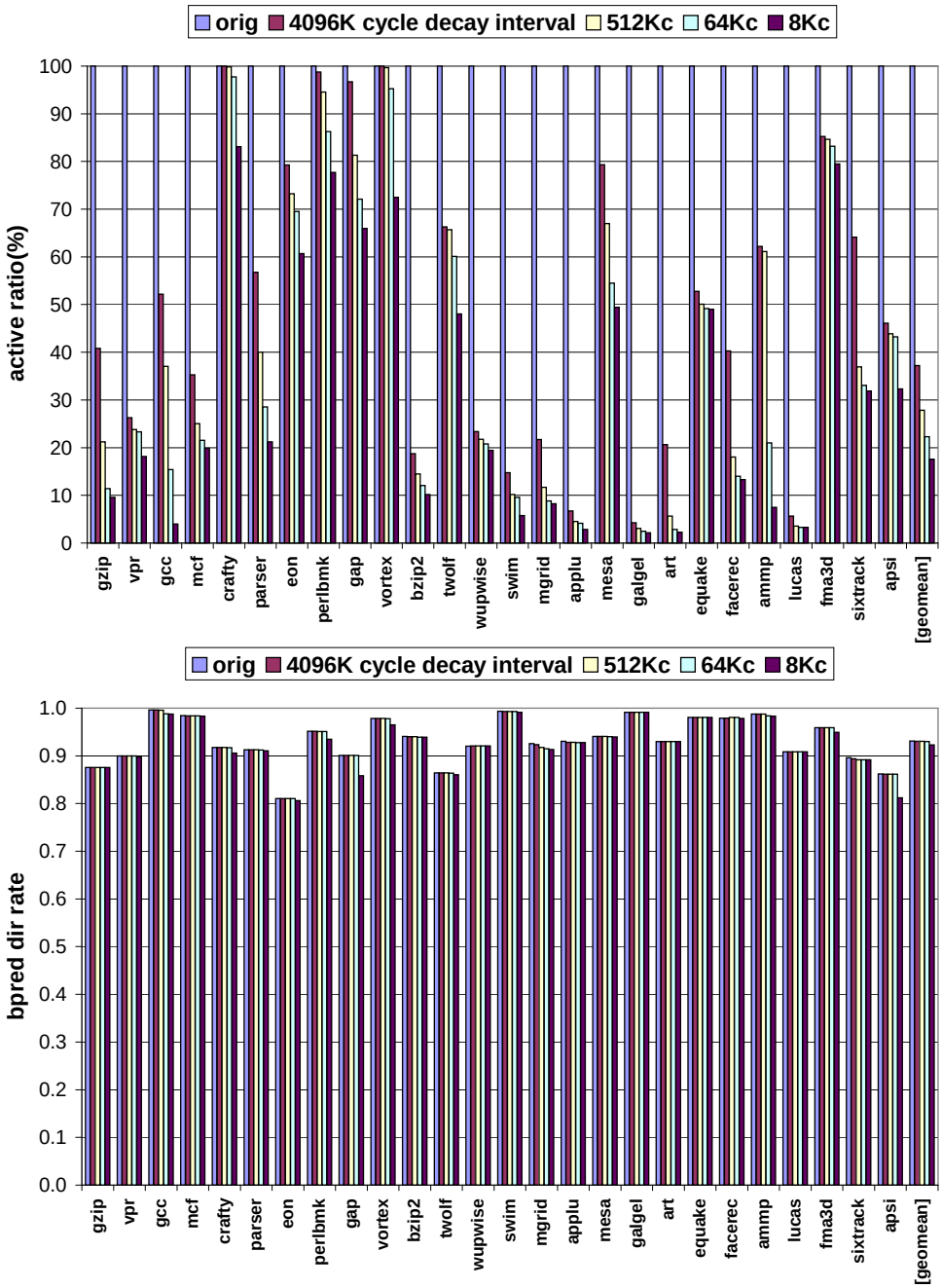

Fig. 6. Active ratio (Top) and prediction success rate (Bottom) for a 4 K-entry bimodal predictor

active ratios with decay, but even these benchmarks suffer only negligible loss in prediction accuracy, so decay does not harm their performance. Because gshare is designed to spread branch state across the predictor to minimize aliasing, its decay benefits are not as pronounced as for a bimodal predictor. Nevertheless, decay still produces substantial reductions in leakage power with minimal performance impact. Figure 10 shows that the reduction in leakage energy is 41% for a 64 K-cycle decay interval.

Further energy savings can be realized using a banked organization, which gives a reduction in leakage energy of 51%. Note that, with the banked predictor, the reduction in leakage energy will be somewhat less than the reduction in active ratio for two reasons. First, because the banked organization has smaller rows and hence the overhead of more

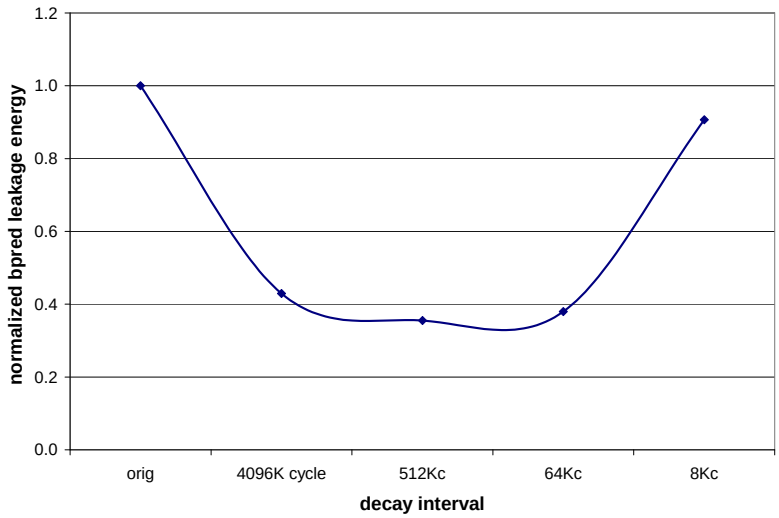

Fig. 7. Normalized leakage energy for the 4K bimodal predictor

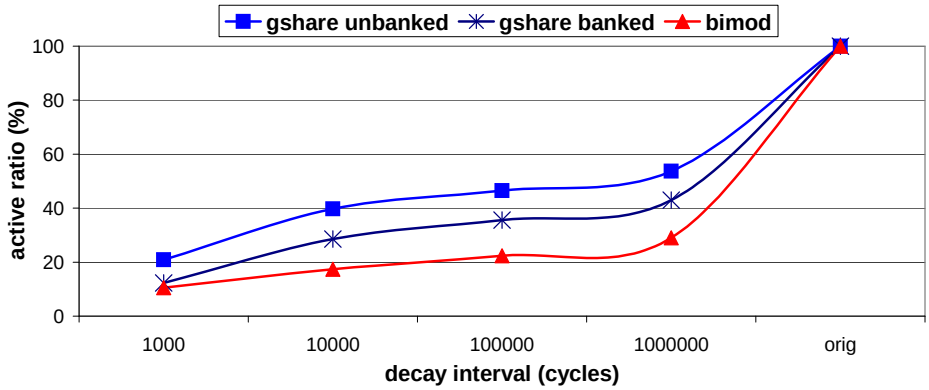

Fig. 8. Mean active ratio for unbanked and banked gshare predictors and a bimodal predictor with different decay intervals. The rightmost label "orig" corresponds to non-decaying predictors.

decay status bits. Second, because the banked organization is more aggressive than unbanked gshare in deactivating rows for the same decay interval. This makes it more likely that the banked organization will deactivate a row that actually harms prediction accuracy. This extra dynamic-power overhead, however, decreases with increasing decay interval because for very long intervals, even a banked row that is idle is extremely likely to have genuinely decayed. This is why the difference in energy savings is larger for 512K cycles and 4096K cycles than for 64K cycles.

E. Decay with Hybrid Predictors

With two competing components (the global component and the local component), hybrid predictors exhibit many interesting design choices when implementing decay. In this section we will explore these design choices as well as their effect on decay in an Alpha 21264-style hybrid predictor.

a) Selection Policy: The selection policy refers to the policy for choosing a prediction from one of the two component predictors. In a non-decaying 21264-style hybrid predictor, the chooser makes this decision using the global history; see Figure 4. However, when decay is enabled, it may happen that only one of the two components is active while the other is decayed. In this case, since the decayed component has lost its information, it is intuitively appealing to use the prediction from the active component, no matter what the chooser suggests. This policy is called

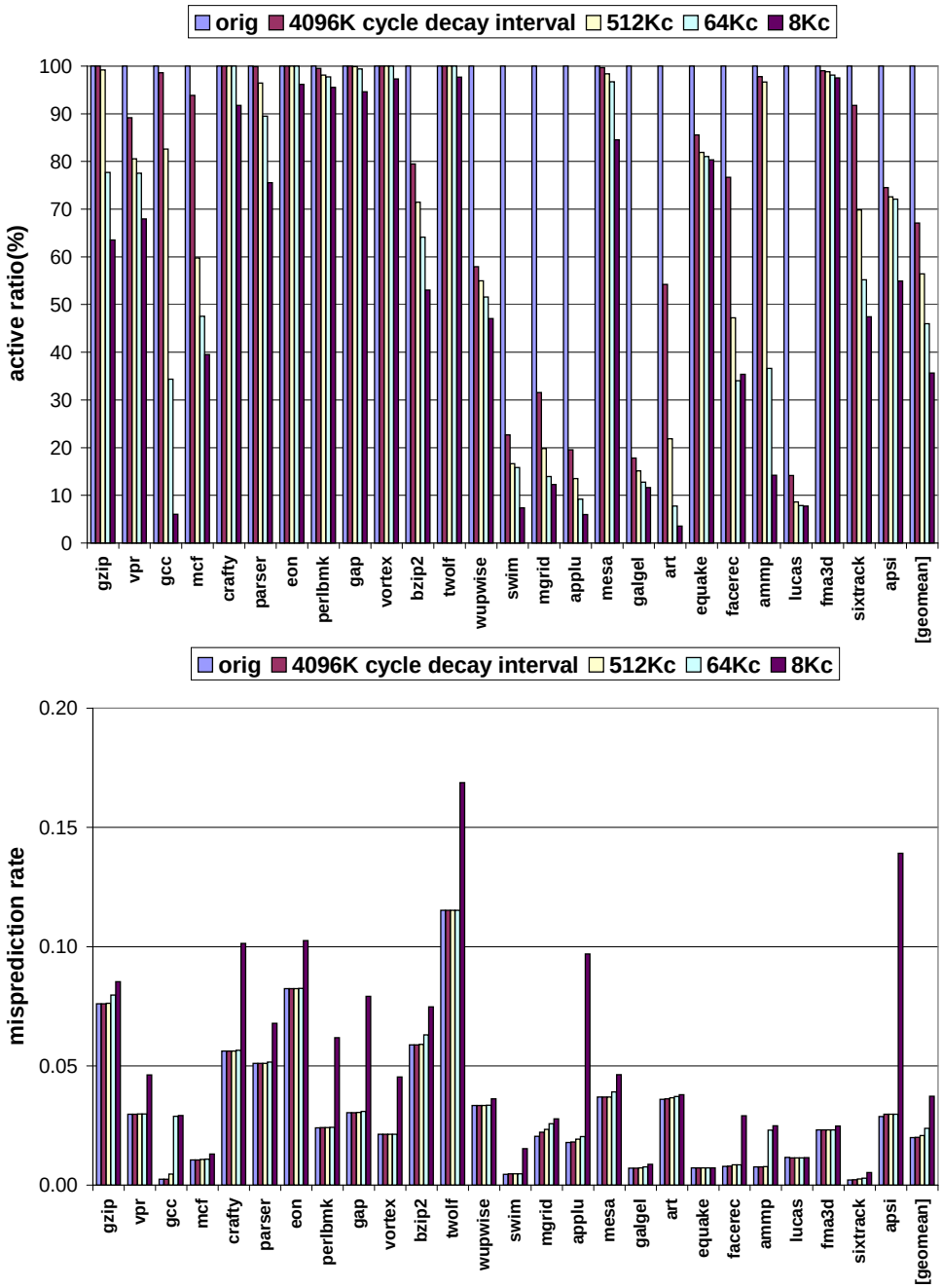

Fig. 9. Active ratio (Top) and misprediction rate (Bottom) for unbanked gshare predictor

"believe the active component", and is implemented in all our experiments. It may also happen that both the two components are decayed, in which case all components are reactivated and the branch is predicted as "weakly not taken", as in bimodal and gshare predictors.

b) Wakeup Policy: The wakeup policy refers to the decision of whether to reactivate a decayed row when it is accessed by a branch instruction. A naive policy would always wakeup any rows that are accessed. In a hybrid predictor, a more elegant policy is possible: the decayed component will be reactivated only if the chooser wants to select it. We refer to this policy as "believe the chooser".

In the situation when the accessed row in the chooser is decayed, we know that the global component is also

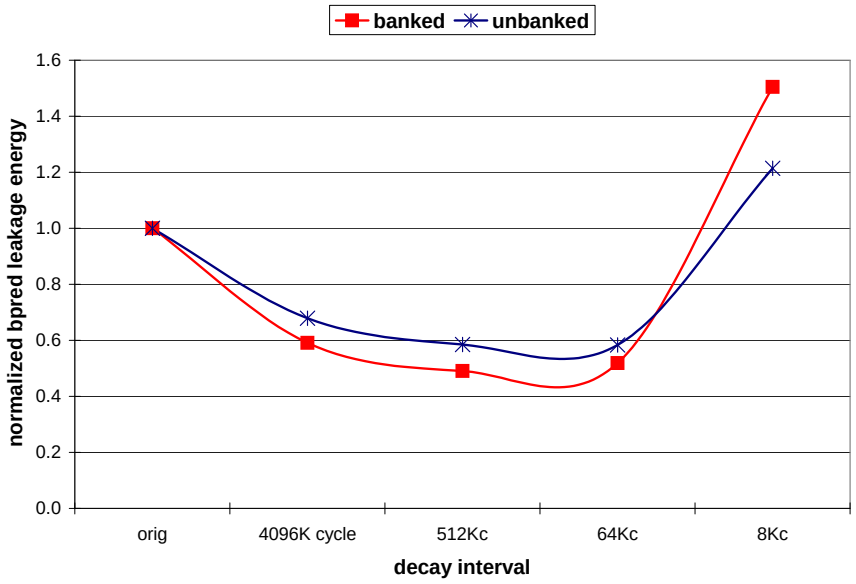

Fig. 10. Normalized leakage energy for gshare branch predictor for both unbanked and banked predictors.

decayed in the 21264-style predictor. This is because the chooser and global predictor are indexed, accessed and thus decayed in exactly the same way; see Figure 4. In this case, the chooser has no useful information. If the local component is active, then we leave the chooser and the global component inactive and return the prediction from the local component. Otherwise (when the local component is also inactive), we reactivate all components and return a prediction of "weakly not taken".

c) Results: Figure 11 details the active ratio and branch misprediction rate for naive decay, which always wakeup any rows that are accessed, with a 21264-style hybrid predictor. We see that even though the active ratios are higher than for bimodal or gshare predictors, decay has a negligible impact on the misprediction rate for intervals of 64K cycles or larger. Note that in order to compute active ratio sensibly on a multi-table structure, we compute it over all prediction and chooser bits in the structure. Overall, as Figure 12 shows, naive decay realizes strong reductions in energy savings—40% for a 64 K-cycle interval.

We can obtain even better energy savings by taking advantage of the "believe the chooser" wakeup policy. As shown in Figure 12, this more sophisticated policy leads to leakage power reductions about 50% better than for the naive policy.

F. Decay with the Branch Target Buffer

In addition to the structure for predicting the direction of conditional branches, a branch target buffer (BTB) [14], [29] is commonly used to store the targets of taken branches. The BTB is typically organized like a cache, either direct-mapped or set associative, but tagged in order to identify hits and misses. Since it is solely indexed by PC, it has similar locality characteristics as instruction caches and bimodal predictors. However, the granularity in the BTB is single branch instructions, instead of instruction blocks as in instruction caches. This allows even finer control for decay. Decay is therefore especially effective for BTBs.

We evaluated a 2048-entry, 4-way associative BTB, which appears in the Intel PIII processor [5]. Except for *vortex*, *perlbmk* and *crafty*, most benchmarks have a very low active ratio, below 20% or even below 10%. This is no surprise,

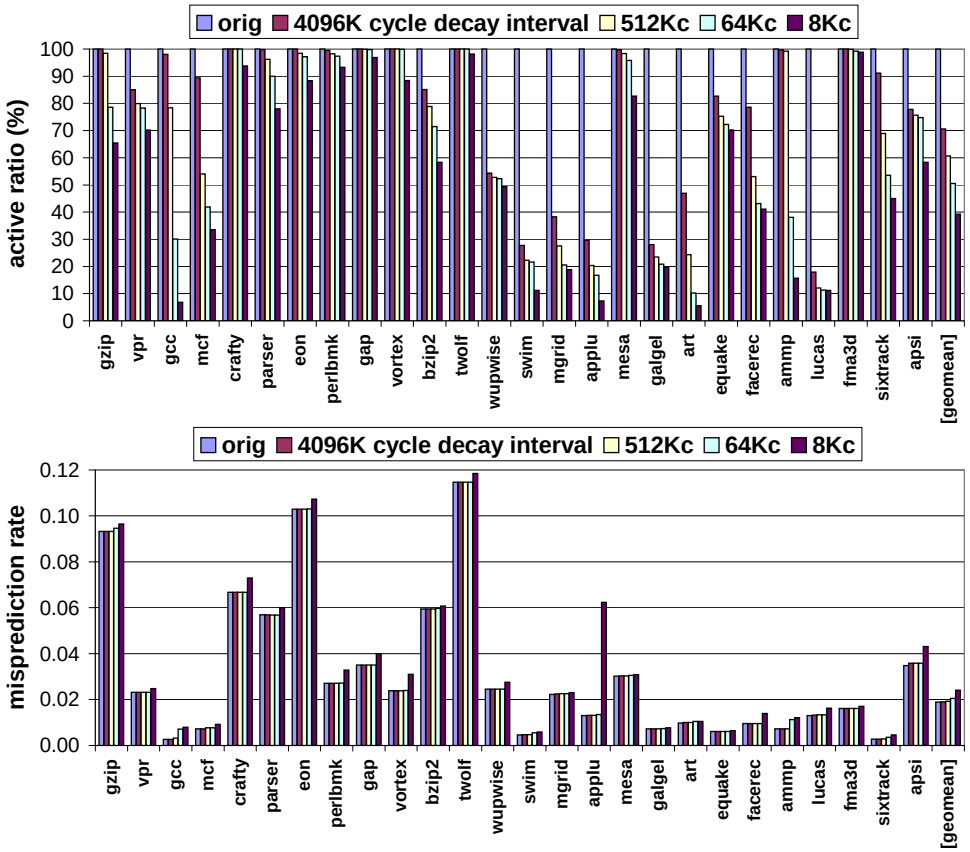

Fig. 11. Active ratio (Top) and misprediction rate (Bottom) for 21264's hybrid predictor

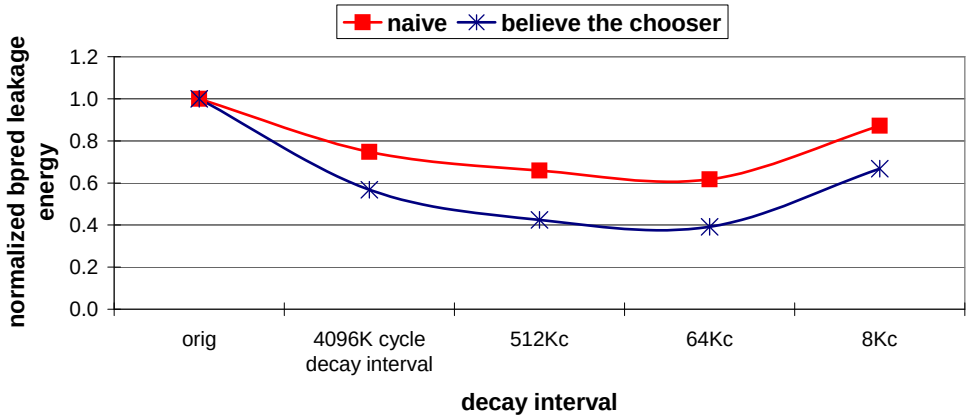

Fig. 12. Normalized leakage energy of 21264-style Hybrid predictor with naive and "believe the chooser" wakeup policies

considering that the static footprints of most benchmarks in Table III are very small compared to our BTB capacity. With decay, these static footprints are automatically tracked and all other idle slots are turned off to save leakage energy. On the other hand, BTB hit rates observe only negligible degradation until decay interval drops to 8 Kcycles or less. Overall, with decay intervals of 64 Kcycles or more, about 90% of the BTB leakage energy can be saved.

G. Basic Branch Predictor Decay : Summary

In this section, we have explored the application of decay techniques to reduce leakage energy in branch predictor structures. The particular policies that we have developed (*e.g.*, "believe the chooser") apply to all non-state-preserving

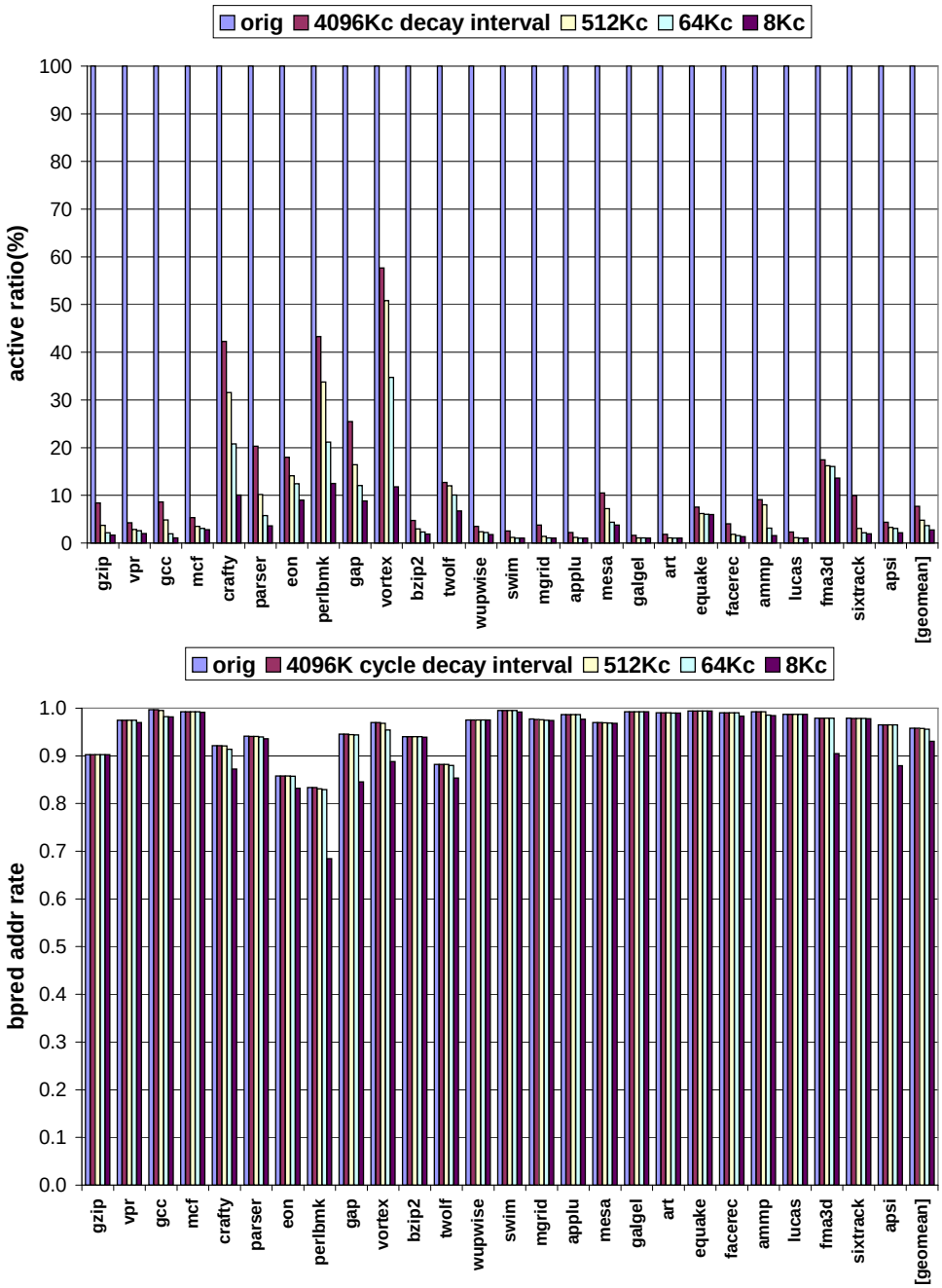

Fig. 13. Active ratio (Top) and accuracy (Bottom) for the BTB with different decay intervals.

leakage-control mechanisms. The key contributions are more general, giving useful guidance to any work on leakage control. In particular, our locality analysis shows that all predictor organizations we examined have significant numbers of rows that are inactive for long periods of time. This makes leakage control in the branch predictor and BTB profitable, regardless of mechanism. Additionally, prior work by Jiménez *et al.* [21], Parikh *et al.* [33], and Skadron *et al.* [36] suggests that branch predictors should be *larger* for both performance and power-saving considerations, but that localized-heating concerns may be an obstacle. This makes leakage control in the branch predictor and BTB particularly important.

Over all the configurations we explored, the best reductions in leakage power were achieved with the bimodal

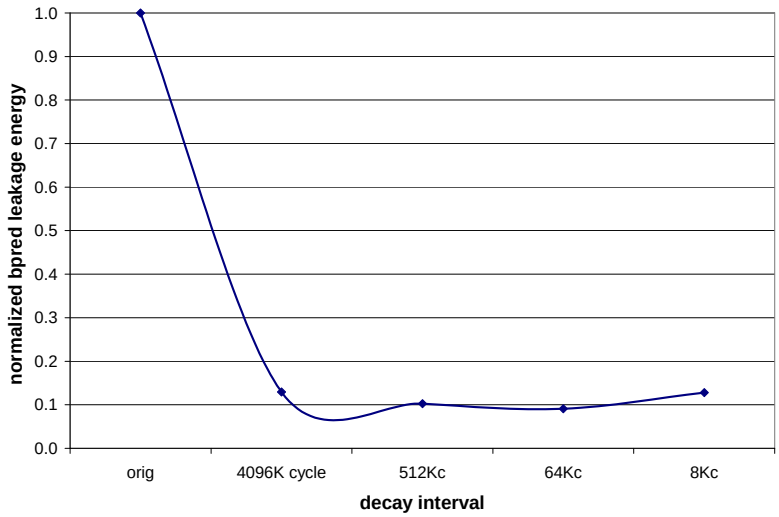

Fig. 14. Normalized leakage energy for the BTB with different decay intervals

predictor, but the power savings achieved with the "believe the chooser" policy for hybrid prediction were nearly as good. Since hybrid prediction has superior prediction accuracy and hence performance and overall energy, hybrid prediction remains the best choice from both a performance and energy standpoint [33].

Some of our interesting results were specifically due to the fact that there are two independent component predictors in a hybrid predictor. In the 21264's hybrid predictor, for example, the chooser and global component were organized in the same way. This let us deduce useful facts about one based on state in the other. More generally, the influence of indexing on decay may suggest that the choice of predictor index is worth revisiting (yet again). In the past, indexing functions have been chosen to minimize aliasing, usually by spreading the state from branches in the same working set as widely as possible. In contrast, decay can be more effective by clustering similar states into rows, with the goal of making as many counters in a row as possible idle or active at any point in time. This observation therefore sets up a tradeoff between reducing aliasing and increasing decay opportunities. Although beyond the scope of this paper, seeking index functions that can balance the two factors is an interesting area for future work. It may be possible to develop tractable index functions that cluster similar state into rows with minimal increase in aliasing. In addition, large predictors n particular may be able to accommodate index functions that boost row clustering.

As for the actual decay intervals, this paper applies only fixed decay intervals. Kaxiras *et al*. [24] show that for caches, even better decay rates can be achieved with decay intervals that adapt to changing program behavior. While studying adaptive branch predictor decay was beyond the scope of this paper, it remains worth investigating since our data showed that some benchmarks suffered no performance loss even with a short, 8 K-cycle decay interval (and would get even more benefit from decay), while others suffer severely with such a short interval.

A further consideration is that banked and multi-table organizations provide substantial benefits in reduced *dynamic* energy, by reducing word- and bit-line lengths. Furthermore, when decay is applied to multi-table predictors, some tables can be left inactive, as in the "believe the chooser" policy. Another example arises in local-history prediction, where timing issues may require caching the predicted direction for each BHT entry in the BHT. As with "believe the chooser", decay might permit the PHT update and lookup in such a local-history organization to be omitted if that PHT entry is inactive and the cached direction was correct. Such a policy might be called "believe the BHT". Not only do these "believe" policies reduce leakage energy, they avoid the dynamic power associated with accessing that table.

Our work here did not model these extra savings, but doing so would only make decay more valuable.

Another issue that warrants study further is interference in branch predictors. In some experiments we observed mild but interesting improvements in prediction rate with decay. This shows that decay (by setting the two-bit counters to a weak state) may have the effect of reducing destructive interference, something we plan to quantify in our future work. Lastly, since many other prediction structures such as value predictors [28] and prefetch address predictors [26] are organized similarly to branch predictors, an interesting future effort will be to apply strategies found in this paper to these structures.

V. Decay With Four Transistor RAM Cells : Overview

Thus far we have explored the use of traditional SRAM cells and their use in decay. Traditional decay techniques involve manipulating the power supply to create the illusion of dynamic behavior on top of a cell originally designed and selected for its ability to hold its value indefinitely. Viewed in this light, it seems natural to use quasi-static 4T cells to implement decaying branch predictors.

A. The Quasi-Static 4T Cell

Basic four-transistor (4T) DRAM cells are well established and described in introductory VLSI textbooks and various articles [30], [32], [41]. 4T cells are similar to ordinary 6T cells but lack two transistors connected to V_{dd} that replenish the charge that is lost via leakage (Figure 15). Using exactly the same transistors as an optimized 6T design the 4T cell requires approximately 2/3 of the area. Under a 0.18μ process, an ordinary 6T cell occupies an area of 5.51μm^2, whereas a 4T cell would occupy an area of 3.8μm^2, i.e., 69% of the area [6].

Performance-wise the 4T cell is just as fast as a 6T cell; while our data demonstrates a slight speed disadvantage, the difference is so small that coupled with the smaller amount of parasitic interconnect, the difference essentially disappears. More importantly, 4T DRAM cells naturally decay over time (without the need to switch them off); once they lose their charge they leak very little since there is no connection to V_{dd}. However, some secondary leakage via the access transistors still remains due to bit-line precharging which we do take into account in our transistor-level simulations.

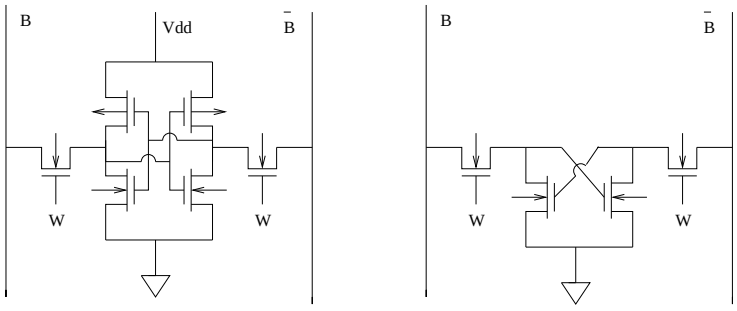

Fig. 15. Circuit diagrams of the 6T SRAM cell (left) and the 4T quasi-static RAM cell (right).

In addition, 4T cells are automatically refreshed from the precharged bit lines whenever they are accessed. When a 4T cell is accessed, its internal high node is restored to high potential, refreshing the logical value stored in it; there is no need for a read-write cycle as in 1T DRAM. As the cell decays and leaks charge, the voltage difference of its

internal nodes drops to the point where the sense amps cannot distinguish its logical value. Conservatively, this occurs when the node voltage differential drops below a threshold of the order of 100 mV (for 1.5V designs). Below this threshold we have a decayed state, where reading a 4T DRAM cell may produce a random value —not necessarily a zero. Over a long time the cell reaches a steady state where both the high node and the low node of the cell "float" at about 30mV (for 1.5V designs).

4T cells possess two characteristics fitting for decay: they are refreshed upon access and decay over time if not accessed. In the rest of this section we discuss extensively the 4T decay design, including decay or data retention times, dynamic energy and other considerations.

B. Hold Times In 4T Cells

The critical parameter for a 4T design is *retention time*. When implemented in 4T cells, decay techniques have the cell's retention time as their natural decay interval. We define retention time to be the time from the last access to the time when the internal differential voltage of the cell drops below the detection threshold. Hold time depends on the leakage currents present in the 4T cell which in turn depend on process technology variations and temperature.

To study retention times for the 4T cache decay we chose Agere's COM2 0.16u CMOS process for which we have accurate transistor models. We also use COM2 for this study because it is the most modern of the four COM processes that is available in-house, in real silicon, to validate our models against real measurements. Although this particular technology does not suffer excessively from leakage, our analysis scales to future generations. Subsequent sections will discuss how to manipulate retention times to match application needs, so although our initial retention time numbers are for COM2, we feel they are accessible in COM3 and COM4 as well.

Variations in the process technology significantly affect leakage currents and therefore retention times. In our studies we use three reference points in the process technology: (i) Nominal transistors (NOM) represent the expected behavior and constitute the bulk of the production; (ii) Worst Case Fast (WCF) transistors are transistors that are six standard deviations (sigma) ($>> 99.99\%$) from the process mean in terms of speed. They are very fast but very leaky transistors; (iii) Worst Case Slow (WCS) transistors are the opposite 6-sigma point in the process distribution, where transistors are quite slow but leak far less.

The WCF and WCS are extreme cases that by definition rarely appear in production; they are essentially theoretical constructs used as bounds. Modest variations in process technology around NOM do occur, but usually among different production lots; it is rare to see significant variation on the same wafer and even more so on the same die. In general, fast chips are likely to leak more and thus have shorter retention times. However, fast chips are also clocked at higher speeds which means that even their short retention times correspond to many cycles. It is the cycle count that matters, not real time, since decay is an architectural phenomenon characterized by cycle counts and largely independent of cycle duration.

As mentioned previously, variations in temperature also result in large variations in retention times. Our designs target an operational temperature of 85 C (appropriate for example for mobile processors) but we also discuss mechanisms to adjust in very high temperature (125 C).

Finally, selecting 3.3V I/O transistors —readily available in COM2 technology — to replace the 1.5V transistors while maintaining 1.5V signaling in the 4T cells significantly extends retention times at the expense of increased cell area. Even in this case the 4T cell, area is still less than that of the 6T cell. Building 4T cells out of 3.3V transistors

while operating them at 1.5V is feasible because of the nature of the 4T cell which acts as a placeholder for charge. The same cannot be done for an active 6T circuit (two cross-coupled inverters) which requires its transistors to be fully biased to work correctly.

	1.5V			3.3V		
	RETENTION TIME			RETENTION TIME		
	25° C	85° C	125° C	25° C	85° C	125° C
NOM	18,000	1700	560	1,040,000	57,200	9400

TABLE IV

HOLD TIMES IN NANOSECONDS FOR 1.5V AND 3.3V VERSIONS OF 4T CELLS AT DIFFERENT OPERATING TEMPERATURES. FOR A 1GHZ (1NS CYCLE TIME) PROCESSOR, ONE CAN ALSO CONSIDER THESE RETENTION TIMES AS CYCLE COUNTS.

Based on these assumptions, we determined retention times for our technology through detailed transistor-level simulations. We simulated an access to a cell, followed by a long period in which the cell was left unread. During this time, leakage causes the cell's internal nodes to lose charge. We defined the hold time to be the duration between an access and the point at which the differential voltages of the 4T cells internal nodes lapsed to a value less than 100mV. We used 100mV as our criteria for the minimum voltage we would expect the sense amps to distinguish the logical state. If the 4T cell is read after it has decayed beyond this point, it still operates safely and does not raise metastability or other concerns; it just does not have the data it had before. Reading a decayed cell produce an electrically stable, though logically random, value. It is essential therefore to know well in advance when data have decayed in a 4T cache. This necessitates decay counters not to turn the branch predictor row on and off as in previous work but to just signal the decay of a branch predictor row. Table IV gives the cell retention times in nanoseconds for the COM2 technology. These retention times are for a 4T cell using the same highly optimized 6T cell transistors.

C. Sense Amplifier Considerations

One implication of the 4T cell is that it becomes progressively more expensive to read as it decays. As the cell decays the sense amp must work harder to detect and amplify the differential signal on the bit lines. Besides the sense amp energy there is also energy expended to refresh the cell as we read it. We have studied both of these effects for all the types of 4T cells (with 1.5V and 3.3V devices) and for various stages of decay. Table V gives the read energy for the 4T cell at various internal voltages.

Internal	4T		6T
Internal	Read Energy (fJ)		Read Energy (fJ)
voltage	1.5V	3.3V	1.5V
1.5 V	na	149	156
1.0 V	166	151	na
0.7 V	171	153	na
0.5 V	175	155	na
0.1 V	219	199	na

TABLE V

READ ENERGIES IN FJ FOR NOMINAL 1.5V AND 3.3V 4T CELLS AT 85° C FOR DIFFERENT STATES OF DECAY (*i.e.*, DIFFERENT INTERNAL VOLTAGES). VALUES INCLUDE ENERGY DISSIPATED IN THE SENSE AMPLIFIER. THE 3.3V 4T CELLS ARE OPERATED AT 1.5V. AN INTERNAL VOLTAGE OF 1.5V IS NOT AVAILABLE IN THE 1.5V 4T BECAUSE THE 4T CELL INTERNAL NODE VOLTAGES ARE CONSTRAINED BY THE CUT-OFF REGION OF THE MOSFETS USED AS ACCESS TRANSISTORS.

D. Locality Considerations

Granularity is also relevant in the 4T design but here it stems from the way 4T cells are refreshed. Branch predictors are typically laid out as a square, with each row having multiple neighboring predictors. In a predictor as nearly square as possible ('squarified', as in Figure 5), reading a row refreshes all the cells in a row because the wordline is asserted. (Segmented wordlines would allow more selective refresh but these designs are outside the scope of this paper.)

Retention time selection and locality granularity go together because large row granularities make the apparent rate of refresh much higher. Cells that would have decayed if left alone get refreshed coincidentally by nearby active cells. Thus 4T cells with short retention times may not lose data as quickly if the row size is long enough.

In contrast, in a design with very fine row granularity, one would opt for 4T cells with very long retention times. Fine granularity leads to a very good decay ratio but the important cells must remain alive on their own (without the benefit of accidental refreshes) for considerable time.

E. Results for 4T-Based Branch Predictors

To illustrate the effects of the increase in the misprediction rate due to decay, we can look at a processor built with 4T structures. It is important to first realize that the two major branch predictor components – direction predictor and BTB – are built somewhat differently. Direction predictor counters, when squarified, tend to have much lower row granularity than BTB targets and thus exhibit different locality characteristics.

For this paper, we simulated under the worst case scenario, i.e. small static leakage energy in comparison to dynamic energy. This correlates closely with the COM2 process, and will show that in future technologies, our branch decay technique will only improve. Thus, we use slow-decaying 3.3V 4T cells in our design, both in the BTB and in the direction predictor. As for the overall configuration, we again use the same 16K-entry gshare configuration as in Table I. We again target an operational temperature of 85 C; this leaves us a decay interval of 57,200 cycles for the direction predictors and a decay interval of 57,200 cycles for the BTB. It is important to note that these values can be adjusted; for example, if 57,200 cycles is deemed too long, we can extend the retention times of 1.5V 4T cells to select a more aggressive decay interval.

Figure 16 (left) shows the execution time (in percentages) of various benchmarks using standard (non-decaying) branch predictors and 4T based branch predictors. From the graph, we see that execution time is virtually unchanged. In fact, a few benchmarks actually improve 2-3% due to the random effects of reading decayed values. On average, however, the effect, whether beneficial or detrimental, is negligible. Furthermore, prediction accuracy (figure 16, right) was also virtually unchanged. Over all the benchmarks, the overall prediction accuracy was down less than 0.5%. Figure 17 shows the active ratio of the direction counters under various SPEC benchmarks. On average, we see a 28.2% active ratio, which directly translates into over 70% savings on leakage power over a traditional, non-decaying predictor.

The savings is not without cost, however; additional read energy is required every time a nearly decayed or completely decayed counter is read. Under our current process, this penalty is an additional 27.5% of the read energy of a non-decayed counter.

Moreover, this penalty is applied also when nearly decayed counters on the same row are refreshed accidentally due to a nearby read; fully decayed counters on the same row, though, are left decayed and therefore not refreshed. We implement this by detecting the voltage differential on the bitlines; if this voltage swing is close to zero, we can tie

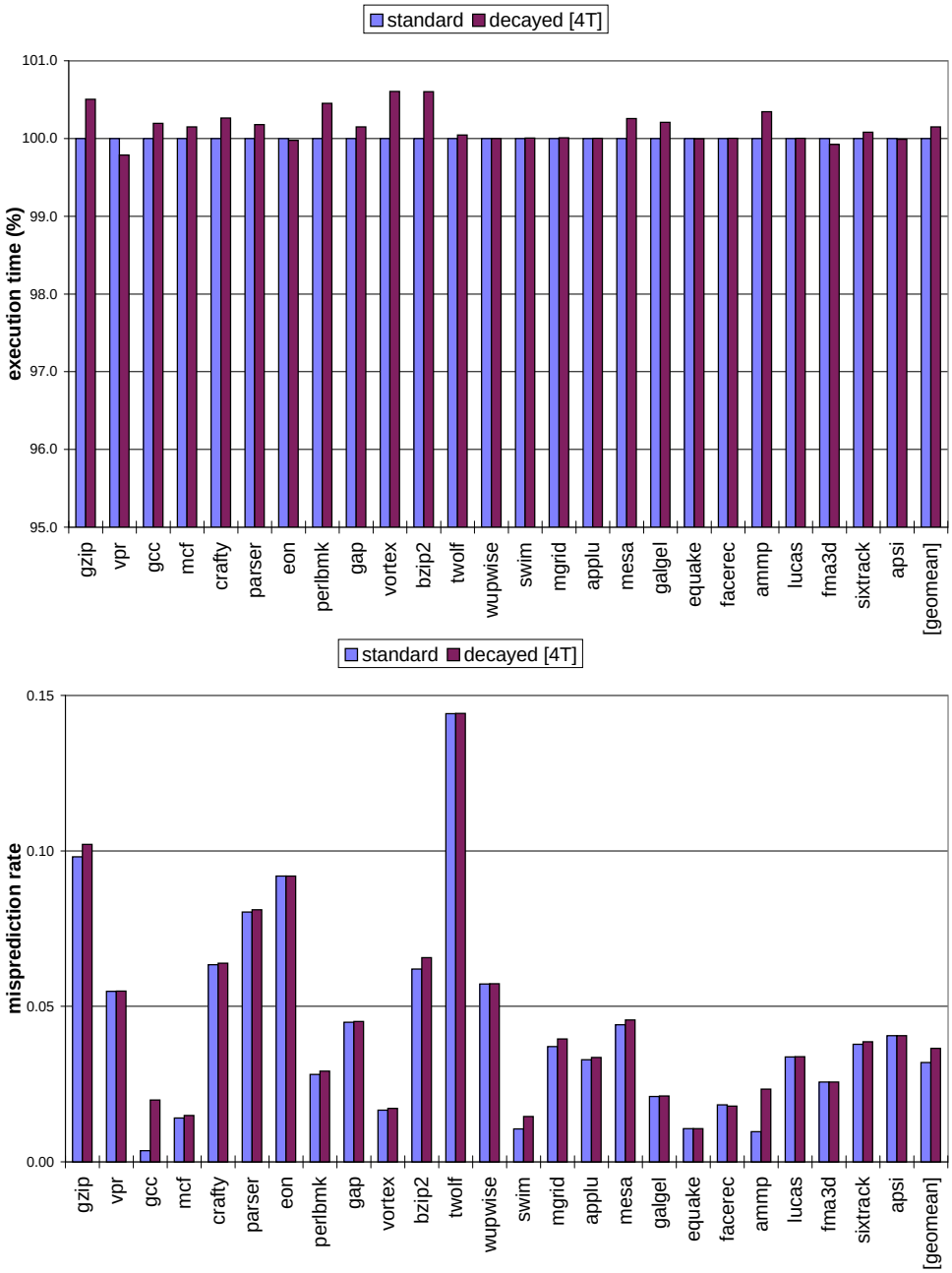

Fig. 16. Normalized execution time (Top) and misprediction rate (Bottom) of standard and 4T predictors. 4T predictors produce minimal performance losses.

the signal into the refresh lines and avoid refreshing an unused cell. The sum total is that we retain the effective long decay interval of low row granularity but achieve much higher active ratios.

Finally, the normal dynamic energy overhead of additional mispredictions must be included in our results. Figure 18 shows the 4T based branch predictor in comparison with the previous 6T designs. While the 4T decay interval is set at 57.2k, at equivalent processes, the 4T usually outperforms the 6T. For instance, a 6T based decaying predictor on a COM3 process would actually consume more energy than a standard, non-decaying predictor, whereas the 4T version of the same predictor, with a shorter decay interval, does better.

In addition, the data also shows that it is possible with 4T cells to use very aggressive decay intervals in the direction

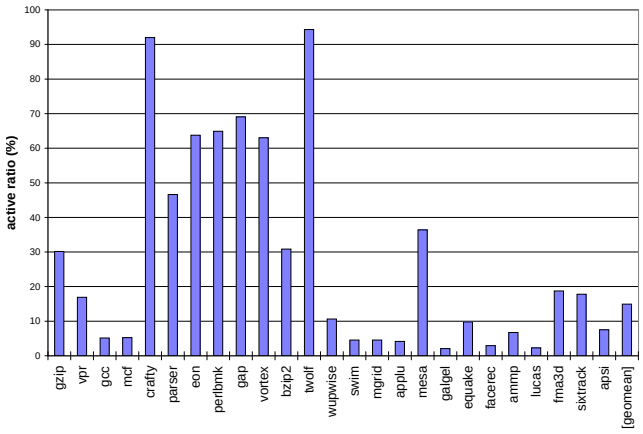

Fig. 17. Active ratio of a 4T based predictor.

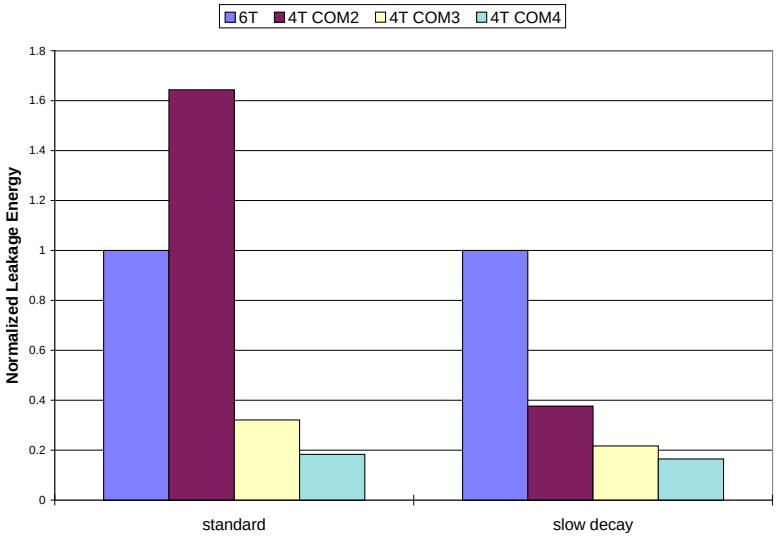

Fig. 18. Normalized leakage energy for the 4T branch predictor at 57.2k and 8k cycles.

counter tables. Under the COM4 process, we see leakage power savings even when the decay interval is 8000 cycles. As mentioned above, we are seeing the effects of the lower active ratios at the same row granularity that the 4T cells allow.

Examining the BTB decay reveals similar observations. Because each BTB target is much larger than the two-bit counter, we can afford to attach counters to each BTB target and thus achieve the optimum granularity; those counters that are used are refreshed, and those that decay are not accidentally refreshed. As a result, the active ratios of the 6T BTB are identical to the 4T BTB. Our savings in the BTB come mainly from lower leakage 4T cells and the elimination of the decay counters.

We see, then, that using quasi-static 4T cells allows us to build naturally decaying branch predictors with minimal impact on performance. Furthermore, predictors built using 4T cells offer additional benefits over those already shown with predictors configured with 6T cells.

F. 4T Decay : Discussion and Summary

This section expands on some key additional issues regarding decay based on 4T cells, and then summarizes our results.

1) Controlling Retention Times: The success of a 4T decay design depends on matching retention times to access (i.e., "refresh") intervals. Thus, the ability to control retention times could give us a new degree of freedom in designing 4T decay structures.

A way to affect retention times is to add devices such as resistors or capacitors to the basic 4T cell [7]. Such devices can be used to slowly replenish the lost charge. If the rate of replenishment is less than the leakage the cell will still decay albeit much more slowly, thus retention time can be extended significantly.

2) Metastability: Another key issue regarding 4T structures is the fact that metastability problems are a possibility when the cell's internal differential voltage is too small. To avoid metastability, it is tempting to use refresh, but this would obviate the savings of our approach. Instead, one can detect the small differential voltages and avoid relying on array data at these points. As an example of the latter, we propose that one could avoid metastability in a 4T branch predictor by adding a reference column whose sole purpose is to detect low differential voltage and to prevent the sense amp output from propagating further into the circuit. In this column, instead of a sense amplifier we have a voltage comparator. When the voltage difference is too small, the comparator output forces the reference cell to read as a logical zero (otherwise it reads as a logical one). The output of the comparator qualifies the result. A small differential is therefore prevented from inducing metastability in the subsequent circuit.

Other approaches are possible, such as the use of decay counters [24], but the one we have proposed—the use of a single comparator in a reference column—is appealing because it prevents metastability while requiring minimal extra area or power.

3) Reliability, Determinism, and Controlled Operation: Another key issue regarding 4T structures is the fact that in some cases it is necessary for 4T memory structures to appear static. As in any other DRAM, this is done by periodically refreshing the dynamic cells. Refresh can be accomplished in a way that is largely transparent to the normal operation of the memory array. Hanamura et al. [11] describe a refresh mechanism that periodically momentarily strobes the address lines after the bit lines have been precharged. Sense amps are idle at this point, resulting in low power consumption. The short strobe pulse allows charge to flow from the bit lines to the cells of the selected line and restore their contents. Ordinary reads and writes take precedence over refresh resulting in minimal performance impact.

Such refresh circuitry (whose area-cost is negligible) is likely to be included with a 4T data array. In normal operation the refresh circuit is inactive and the 4T structure operates in decay (low-leakage) mode. However, under special circumstances refresh provides indispensable functionality. These include : (i) chip testing, (ii) deterministic operation for real-time applications, (iii) possible operation beyond recommended temperature range. While the first two should be apparent, the third one arises because at high temperatures, leakage may be so high that branch predictors decay too quickly. In such situations, the system will want to re-enable refresh thus revert to non-decay mode.

4) Improvements in Process Technology: With the advent of new process technology, transistor leakage will increase at a tremendous rate, making 6T solutions somewhat less attractive due to power dissipation concerns. This same transistor leakage will of course lower the data retention of 4T cells, however since access times will also improve there would appear to be some kind of balance between the number of machine cycles that the 4T cell can hold its data and the number of machines cycles that the 4T cell is required to hold its data in cache applications. Therefore, 4T still wins.

As process technology advances:

- transistor leakage increases for all devices, especially for those types of transistors used in 6T cells, but not so rapidly for those longer channel transistors used in 4T cells
- data access time decreases in all cases
- data retention time decreases in 4T cells
- alpha particle immunity (and soft errors in general) will get worse for both types of cells.

The last point is a key point because 6T cells in this era have shown soft error problems owing to their larger source-drain area. 1T cells (previously notorious for their soft error sensitivity) are not getting as bad as once thought because their source-drain area is so small that the offending particle has a smaller target to hit it, and when it hits there is less of a charge imbalance because the cross-sectional area of the depletion region is so small. Once again the 4T cell seems to be a possible middle ground between 6T and 1T.

5) Decay based on Quasi-Static 4T Cells : Results Summary: This section shows that 4T RAM cells are as much as one-third smaller in area and they are comparable in terms of read energy when accessed frequently. Most importantly, they reduce leakage energy compared to 6T cells and are comparable in terms of program performance.

This paper's proposal to use 4T cells for predictor structures is driven by the following observations.

- Branch predictor entries exhibit locality. Data arrays are approximately square, and decay techniques are most easily applied to rows in these arrays. This helps 4T structures, because an access to anything along a row boosts the voltages of all cells in that row. Locality means that active code regions have their predictors refreshed while inactive regions decay. Once decayed, they leak very little, essentially capping the energy dissipated by idle entries.
- The retention time for 4T cells—the amount of time it takes for a 4T cell to leak enough charge that the value it stores is corrupted—is long enough that 4T cells' decaying behavior is ideally suited to exploiting decay in large structures : it is rare that a value leaks away before it is needed again.
- Retention times vary with design style, fabrication technology, and temperature. Nonetheless, we have discussed techniques that allow one to modulate the retention times sufficiently to guarantee good performance. We have also shown that there exist several techniques that will prevent small voltage differentials in decayed cells from inducing metastability.

These insights suggest that 4T cells are inherently useful for managing leakage in predictive structures. The fact that 4T cells decay naturally provides leakage-energy savings without either the overhead of gated-V_{dd} techniques or the overhead of maintaining decay counter bits. Finally, 4T designs are smaller, saving die area and possibly permitting faster access times for a given number of bits.

VI. Conclusions

In this paper, we examine implementations of decay-based leakage control using quasi-static memory cells. Cache decay, first proposed in [23], [24], aimed to turn off unused portions of caches with long idle times to reduce cache leakage energy.

We start by evaluating decay implementations based on coarse-grained counters appended to traditional 6T SRAM arrays. While previously considered for caches, this paper shows that such techniques can be effective for branch predictors as well. Inherent in this success is the observation that branch predictors exhibit row-based spatial locality that allows some of the rows to be deactivated due to long idle times, while other rows are heavily accessed.

Thus, we see that, much like caches, exploiting this spatial and temporal locality allows us to save significant amounts of leakage energy, especially in simpler branch predictors. In more complex branch predictors we show that an intelligent policies can save significant amounts of leakage energy over naive decay.

Prior implementations of decay were based on gated-V_{dd} or gated-V_{ss} [42], where power and ground are gated by a transistor. This implementation, however, not only leads to about 5% area overhead but also brings about complex design issues especially for turning on a decayed cache line.

A closer examination of gated-V_{dd} based decay reveals an intrinsic conflict between the standard 6-transistor cell design and gated-V_{dd}. Specifically, while gated-V_{dd} tries to shut off cache lines, the two load transistors in 6-transistor cells are merely wasting charge. To resolve this conflict, we proposed using 4-transistor quasi-static memory cells to implement cache decay. By removing the two transistors that connect the cell to V_{dd}, quasi-static cells naturally implements the two key functions for decay: their charges lose over time and are replenished during each cache access. Compared to gated-V_{dd} implementation, this method avoids the area overhead and the design issues related to the gate transistor.

Thus, we also examine the use of quasi-static 4T cells for implementing branch predictor decay. Such cells have been proposed to implement on-chip embedded DRAM in a fairly-traditional style with refresh circuitry. In our work, we examine using the natural decay of the 4T cells to implement decay for leakage-control in branch predictors. Because branch predictors are performance hints, not correctness-critical, lost entries do not cause incorrect execution. Moreover, we show that 4T cells can be built with sufficient natural retention times to implement useful decay-based predictors with negligible impact on prediction rate.

Using a combination of transistor-level simulation and instruction-level simulation we show that a 4-transistor decay implementation achieves significant savings in leakage power. While 4-transistor decay typically offers greater savings than 6-transistor solutions, there are many places were it may not be feasible to implement a 4T branch predictor. 6T decay is more flexible in the selection of retention time, as well as a finer granularity in selecting dynamic retention times. Furthermore, it may not always be possible, either through device concerns or layout concerns to wholesale replace a 6T solution with a 4T solution.

Thus, by presenting both traditional (6T) and quasi-static (4T) solutions, we show that decay can be easily applied either architecturally on top of a given design, or be an integral part of the design from the beginning. While branch predictor leakage is perhaps 10% of total CPU leakage, we are able to reduce it by a significant fraction, sometimes 90% or more. This reduces overall chip leakage by 5-7%. Furthermore we can reduce leakage with essentially no performance cost and in general, an area improvement. Most broadly, the paper prompts a rethinking of how transient data can best be exploited in designing power-efficient processors.

REFERENCES

[1] From website : The international technology roadmap for semiconductors. 2001. http://public.itrs.net/Files/2001ITRS/Home.htm.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architecture-Level Power Analysis and Optimizations. In *Proc. ISCA-27*, ISCA 2000.

[3] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[4] P.-Y. Chang, E. Hao, and Y. N. Patt. Alternative implementations of hybrid branch predictors. In *Proc. Micro-28*, pages 252–57, Dec. 1995.

[5] K. Diefendorff. Pentium III = Pentium II + SSE. *Microprocessor Report*, Mar. 8 1999.

[6] P. Diodato et al. Merged DRAM-LOGIC in the Year 2001. *Proc. of the IEEE International Workshop on Memory, Technology, Design, and Testing*, Aug. 1998.

[7] P. Diodato et al. Embedded DRAM: An Element and Circuit Evaluation. In *IEEE Custom Integrated Circuits Conference*, Jun 2001.

[8] P. W. Diodato. Personal communication, 2001.

[9] K. Flautner et al. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[10] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, pages 11–16, Oct. 28, 1996.

[11] S. Hanamura et al. A 256K CMOS SRAM with Internal Refresh. In *ISSCC*, 1987.

[12] H. Hanson et al. Static energy reduction techniques for microprocessor caches. In *Proceedings of the 2001 International Conference on Computer Design*, pages 276–83, Sept. 2001.

[13] S. Heo et al. Dynamic Fine-Grain Leakage Reduction using Leakage-Biased Bitlines. In *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[14] R. W. Holgate and R. N. Ibbett. An analysis of instruction fetching strategies in pipelined computers. *IEEE Transactions on Computers*, C-29(4):325–329, Apr. 1980.

[15] Z. Hu et al. Applying Decay Strategies to Branch Predictors for Leakage Energy Savings. In *Proceedings of the 2002International Conference on Computer Design*, Sep. 2002.

[16] Z. Hu et al. Managing Leakage for Transient Data : Decay and Quasi-Static 4T Memory Cells. In *Proceedings of the 2002 International Symposium on International Symposium on Low Power Electronics and Design*, Aug. 2002.

[17] Z. Hu, P. Juang, K. Skadron, D. Clark, and M. Martonosi. Applying decay strategies to branch predictors for leakage energy savings. Tech. Report CS-2001-24, Univ. of Virginia, 2001.

[18] Z. Hu, S. Kaxiras, and M. Martonosi. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. *ACM Transactions on Computer Systems*, May 2002.

[19] Z. Hu, S. Kaxiras, and M. Martonosi. Timekeeping techniques for predicting and optimizing memory behavior. In *The 2003 IEEE International Solid-State Circuits Conference*, Feb. 2003.

[20] J. A. Butts and G. Sohi. A Static Power Model for Architects. In *Proc. Micro-33*, Dec. 2000.

[21] D. A. Jiménez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proc. Micro-33*, pages 67–77, Dec. 2000.

[22] P. Juang et al. Implementing decay techniques using 4t quasi-static memory cells. *Computer Architecture Letters*, Sep. 2002.

[23] S. Kaxiras, Z. Hu, et al. Cache-Line Decay: A Mechanism to Reduce Cache Leakage Power. In *Workshop on Power-Aware Computer Systems (PACS, In conjunction with ASPLOS-IX*, 2000.

[24] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proc. ISCA-28*, July 2001.

[25] A. Kesharvarzi et al. Intrinsic iddq: Origins, reduction, and applications in deep sub-m low-power cmos ic's. In *Proceedings of the IEEE International Test Conference*, pages 146–155, 1997.

[26] A. Lai, C. Fide, and B. Falsafi. Dead-Block Prediction and Dead-Block Correlating Prefetchers. In *Proc. ISCA-28*, July 2001.

[27] L. Li et al. Leakage energy management in cache hiearchies. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2002.

[28] M. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *Proc. ASPLOS-VII*, pages 138–47, Oct. 1996.

[29] J. J. Losq. Generalized history table for branch prediction. *IBM Technical Disclosure Bulletin*, 25(1):99–101, June 1982.

[30] R. Lyons et al. CMOS Static Memory With a New Four-Transistor Memory Cell. *Proc. of the 1987 Stanford Conf. On Advanced Research in VLSI*, pages 111–132, 1987.

[31] S. McFarling. Combining branch predictors. Tech. Note TN-36, DEC WRL, June 1993.

[32] K. Noda et al. A 1.9 um2 loadless CMOS Four-Transistor SRAM Cell in a 0.18 um logic technology. *IEDM Tech. Dig.*, pages 847–850, 1998.

[33] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proc. HPCA-8*, pages 233–44, Feb. 2002.

[34] K. Roy. Leakage power reduction in low-voltage CMOS designs. In *Proceedings of the International Conference on Electronics, Circuits, and Systems*, pages 167–73, 1998.

[35] S. Schuster, L. Terman, and R. Franch. A 4-Device CMOS Static RAM Cell Using Sub-Threshold Conduction. In *Symposium on VLSI Technology, Systems, and Applications*, 1987.

[36] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proc. HPCA-8*, pages 17–28, Feb. 2002.

[37] J. E. Smith. A study of branch prediction strategies. In *Proc. ISCA-8*, pages 135–48, May 1981.

[38] P. Song. UltraSparc-3 aims at MP servers. *Microprocessor Report*, pages 29–34, Oct. 27 1997.

[39] The Standard Performance Evaluation Corporation. WWW Site. http://www.spec.org, Dec. 2000.

[40] S. Velusamy et al. Adaptive cache decay using formal feedback control. May 2002.

[41] W. Wolf. *Modern VLSI Design: Systems on Silicon.* Prentice Hall, 1998. Prentice-Hall.

[42] S.-H. Yang et al. An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches. In *Proc. HPCA-7*, 2001.

[43] W. Zhang et al. Compiler-directed instruction cache leakage optimization. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, Nov. 2002.

[44] H. Zhou, M. Toburen, E. Rotenberg, and T. Conte. Adaptive mode control: A static-power-efficient cache design. In *Proc. PACT '01*, Sept. 2001.