Implementing Branch-Predictor Decay Using Quasi-Static Memory Cells

PHILO JUANG
Princeton University
KEVIN SKADRON
University of Virginia
MARGARET MARTONOSI
Princeton University
ZHIGANG HU
IBM T.J. Watson Research Center
DOUGLAS W. CLARK
Princeton University
PHILIP W. DIODATO
Agere Systems
and
STEFANOS KAXIRAS
University of Patras

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 larger, aggressive 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 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 (BTB). 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

Authors' addresses: P. Juang, M. Martonosi, D. W. Clark, Departments of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ 08544; email: pjuang@princeton.edu; K. Skadron, Department of Computer Science, University of Virginia; Z. Hu, IBM T.J. Watson Research Center; P. W. Diodato, Agere Systems; S. Karxiras, University of Patras.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2004 ACM 1544-3566/04/0600-0180 $5.00

ACM Transactions on Architecture and Code Optimization, Vol. 1, No. 2, June 2004, Pages 180–219.

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.

Categories and Subject Descriptors: B.3.1 [**Semiconductor Memories**]:

General Terms: Design, Performance

Additional Key Words and Phrases: Energy aware computing

1. INTRODUCTION

Power and energy consumption are a major concern in general-purpose and high-performance processors. Power dissipation can be separated into two parts. *Dynamic* power dissipation is due to switching activity in the processors, and in ideal devices, is the only source of dissipation, as current only flows during switching. *Static* or *leakage* power is consumed regardless of activity, especially through *subthreshold* leakage currents [Kesharvarzi et al. 1997] that flow even when the transistor is nominally off.

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 fivefold increase in leakage energy per technology generation [Semiconductor Industry Association 2001]. 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 [Butts and Sohi 2000; Kaxiras et al. 2001; Yang et al. 2001; Flautner et al. 2002]. Using gated-V_{ss} techniques, Kaxiras et al. [2001] 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. Li et al. [2004] examined several circuit-level mechanisms for managing leakage control, showing that in cases such as fast on-chip caches, non-state-preserving techniques such as gated-V_{ss} are superior to drowsy caches. After caches, branch predictors are among the largest and most power-consuming array structures in current CPUs. Current predictors and branch target buffers (BTBs) are 4–8 Kbytes in size, already the size of a small cache. They dissipate 7–10% of the processor's total dynamic power dissipation [Parikh et al. 2002].

However, as pipelines grow deeper and the demand on the branch predictor increases, the branch predictor may increase in size faster than caches, raising their contribution to the chip's total energy budget. For example, in the proposed Alpha EV8 [Seznec et al. 2002], the branch predictor was 352 Kbits (44 Kbytes) in size, over 12 times that of the 21264. In addition, Jiménez et al. [2000] pointed out that hierarchical 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. The increased popularity of tournament predictors would rapidly increase the size of the branch predictor as well. As leakage-energy consumption increasingly becomes a concern with respect to dynamic energy consumption, how to conserve leakage energy in a large array structure such as the branch predictor becomes a significant and important problem. Therefore, energy consumption problem in the branch predictor will likely grow in importance in the future, especially with the increasing importance of leakage energy.

As decay techniques for caches have proven effective, branch predictors are a natural next step. However, there are several important distinctions. First, it is less obvious when a branch-predictor entry may be considered "dead" and can therefore be turned off with little performance impact. 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.

Last, branch predictors can be logically organized in a number of ways, from simple *bimodal* branch predictors [Smith 1981], which keep one two-bit counter per predictor entry, to multitable predictors-like *hybrid* predictors [McFarling 1993], 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 six transistor (6T) SRAM cells; unlike DRAM cells, SRAM cells retain their value while charged. Thus, caches need not have built in circuitry to refresh the cells as in main memory. With this in mind, decay and similar techniques focus on somehow shutting down these cells, for example either gating 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 involve attaching a gate transistor to the supply voltage V_{dd}; to deactivate a cell, the gate transistor disconnects the supply to the rest of the cell and thus the cell is effectively turned off [Yang et al. 2001]. Gated-V_{dd}-based cache decay techniques have a few problems, however. First, 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. In addition, the counters themselves consume dynamic and static power, which must be taken into account as well. 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, four-transistor (4T) memory cell. 4T cells are approximately as fast as 6T SRAM cells, but do not have connections to the supply voltage (V_{ss}). Rather, the 4T cell 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 do not require a full read–modify–write operation to accomplish a cell refresh.

4T cells typically appear in embedded processors, where low-power consumption is the most important aspect. 4T cells have the speed of SRAM but in terms of power behave more like DRAM. They have been largely overlooked in general-purpose processors because without periodic refresh they are not truly static. However, in decay techniques, where we exploit the transient nature of short-term storage by turning them on and off depending on usage, one can take advantage of 4T cells' dynamic nature. Decaying a 6T cell seems somewhat like squashing its greatest asset—that it retains its value for as long as it is powered up. Implementing decay with 4T cells is therefore convenient; 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 having two fewer transistors and by eliminating the need to run power rails lengthwise throughout the array, 4T-based structures are smaller than equivalent 6T structures. As 4T cells decay and revive naturally through the read–write process, some of the costs behind activating decayed cells are eliminated. For example, a write to a 4T cell automatically reactivates the cell, whereas reactivating a 6T cell is somewhat more complex, requiring extra hardware involved in gating the supply voltage.

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:

(1) Branch predictors, like caches, exhibit a significant amount of spatial and temporal locality, and thus decay techniques can be applied.

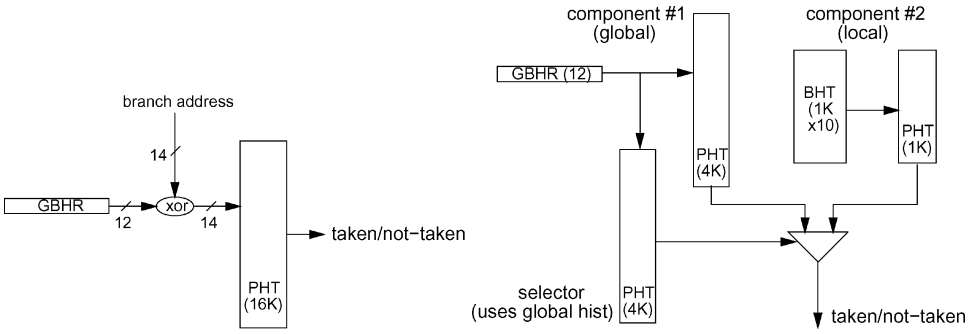

Fig. 1. Gshare predictor in the Sun UltraSPARC-III (Left) and 21264-style hybrid predictor (Right).

(2) Unlike caches, a decayed prediction need not impact performance as much as a decayed cache line. A decayed prediction, whether a static fallback or a logically random value, may still be a correct prediction.

(3) Decay can reduce net leakage energy in the conditional branch predictor by 40–60% and the BTB by 90%.

(4) 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.

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

(6) Quasi-static, 4T cells are a natural fit for decay, as they gradually leak charge over time . Moreover, they suffer from less leakage overall than 6T cells, making them an attractive choice for managing leakage energy.

The paper is organized as follows. First, we discuss the organization and layout of branch predictors, and establish that spatial and temporal locality exists. We then show results for basic decay in (6T) branch predictors. Next, we discuss the structure and behavior of the 4T quasi-static cell, followed by the results of the decay implementation. Last, we discuss related work and offer our conclusions.

2. DECAY WITH BASIC BRANCH PREDICTORS

2.1 Logical Structure of Branch Predictors

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 half of Figure 1, 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 16k-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 [Song 1997].

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 [Chang et al. 1995; McFarling 1993] 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 1k-entry local-history predictor with a 10-bit wide BHT and 1k-entry PHT. This configuration appears in the Alpha 21264 [Gwennap 1996] and is depicted on the right side of Figure 1.

2.2 Proposed Decay Implementation

Logically, branch predictors are arrays of counters that are typically just two bits wide. Like caches, however, branch predictors are physically implemented as square or nearly square array structures. This helps 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 two-bit counters. Alternatively, it can be divided into four banks, each a 64×64 counter array. We refer to these two organizations as "unbanked" and "banked" respectively.

When first considering decay techniques with 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. Thus, 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), or a total overhead of $2\sqrt{\text{entries}}$ bits.

Our techniques have the following general structure. At regular intervals, all rows of predictor entries not 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.

Figure 2 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

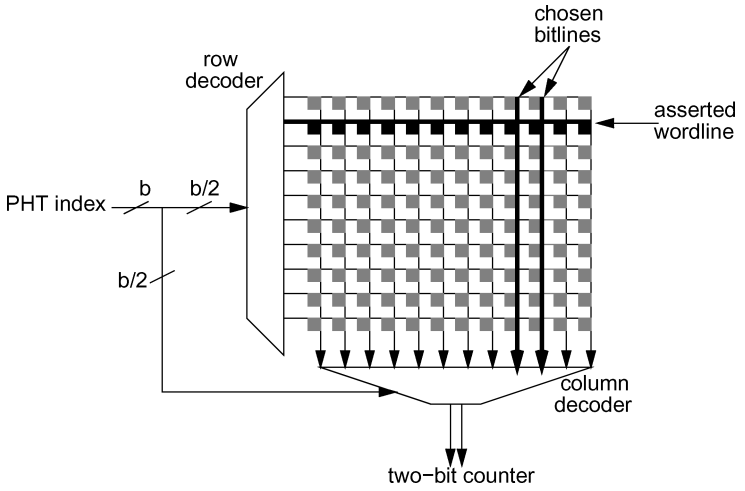

Fig. 2. A schematic of a squarified branch-predictor table of two-bit counters (the PHT).

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 reactivated 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 reactivated line start in the weakly-not-taken state.

The *active ratio* 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. 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.

To evaluate the net effectiveness, we combine the reduced value of leakage energy that was observed with decay and the overhead energy associated with the decay technique. For each of the predictor types we study, we then plot the *normalized leakage energy* for different decay intervals against 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 [Kaxiras et al. 2001].

2.3 Spatial and Temporal Locality in Branch Predictors

In order to apply decay to branch predictors, we must first establish the existence of spatial and temporal locality. 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.

In the instruction cache, programs clearly exhibit spatial locality. 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.

If branches were uniformly distributed, we would expect the probability of two successive conditional branches falling into the same row to be close to 1/rows, or about 2% for a 4k-entry predictor array. To establish spatial locality, we must show that this probability is much higher than this. Indeed, the probability that two successive conditional branches fall into the same row in a 4k-entry bimodal predictor is greater than 40% for all our benchmarks, and greater than 50% for all but five.

For gshare or hybrid predictors, spatial locality is not as pronounced as for bimodal, since these other predictors are indexed by a combination of the branch index and global history. Thus, depending on the global history, a single branch can index into many different PHT entries, across many different rows. 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, 16k-entry gshare), and thus even these more complex predictors also exhibit some spatial locality.

Yet this is not useful if the active rows change rapidly, so we must also establish temporal locality. 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 I. Decay will therefore clearly help bimodal predictors, as each static branch touches only one predictor entry. From Table I, we know that they are clustered.

3. EXPERIMENTAL SETUP

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

3.1 Simulation Setup

Simulations in this paper are based on the SimpleScalar 3.0 toolkit [Burger and Austin 1997]. Our model processor has microarchitectural parameters that resemble in most respects the Intel P-III processor [Diefendorff 1999]. The main processor and memory hierarchy parameters are shown in Table II. For performance estimates and behavioral statistics, we use SimpleScalar's *sim-outorder* simulator. For energy estimates, we use the Wattch simulator [Brooks et al. 2000]. 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 five stages, both our architectural and power models extend the sim-outorder pipeline by adding three additional stages between decode and issue.

Table I. Average Number of Static Branches Touched Every Decay Interval for SPEC2000

	1k cycles	10k cycles	100k cycles	1M cycles	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)

The rightmost column labeled 'overall' gives the static branch footprint for the whole simulation period.

Table II. Configuration of Simulated Processor

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	32 KB, 1-way, 32 B blocks, 3-cycle latency
L1 I-cache Size	16 KB, 4-way, 32 B blocks, 3-cycle latency
L2	Unified, 256 KB, 8-way LRU, 32 B blocks,8-cycle latency, WB
Memory	100 cycles
TLB Size	128-entry, 30-cycle miss penalty

We tried to match as closely in performance respects to a P-III class processor as possible, a four-issue width machine. As for the proposed branch predictors, we selected branch predictors found in other four-issue width machines, such as the hybrid found in the 21264 and the 16k gshare found in the UltraSparc-III.

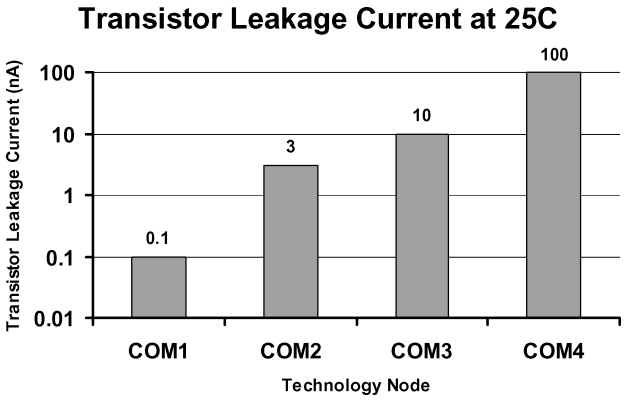

Fig. 3. Leakage current per transistor for a range of technologies in use at Agere.

3.2 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 0.16 μm, a V_{dd} of 1.5 V and a clock speed of 1 GHz. We use 6T and 4T cells from cell libraries; no custom designs are assumed. 4T cells are simple to design, do not require special process technology, and can be included in libraries with other SRAM memory cells.

As process technology primarily determines leakage currents, with each successive generation leakage increases substantially. In particular, Figure 3 illustrates this progression by illustrating leakage currents (in nA) for four industry process technologies from Agere Systems. At 2.5 V and 0.25 μm, COM1 is largely out of date; we do not consider it further. COM2 is a reasonably current CMOS process, with a 1.5 V supply voltage and 0.16 μm feature size. COM3 and COM4 are slightly more forward-looking CMOS processes. COM3 is a 1.0 V, 0.12 μm process, and COM4 is a 1.0 V, 0.1 μm process.

The leakage currents 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 [Semiconductor Industry Association 2001] that leakage is growing exponentially with successive technology generations (Figure 4).

Our transistor models are based on the currently-in-production COM2 process. We use Agere's SPICE-like simulators for very detailed circuit simulations with a 25 ps 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 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.

Figure 5 shows the exponential relationship of temperature to transistor leakage currents as temperature is varied for 1.5 V 0.16 μm COM2 transistors. 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 high-performance or mobile processors) but we also

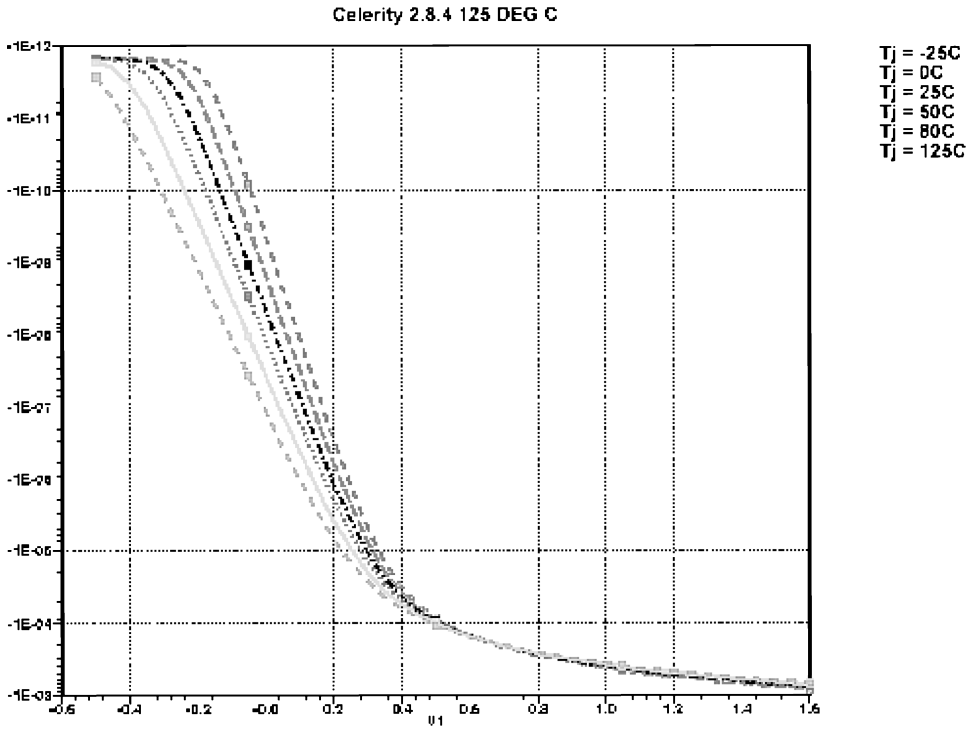

Fig. 4. Current–voltage curves for COM2 transistors at varying temperatures.

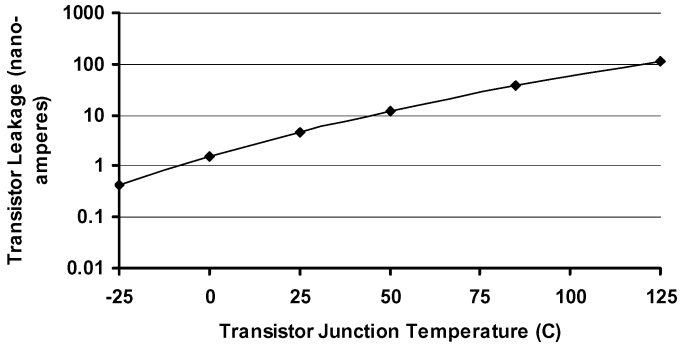

Fig. 5. Transistor leakage current (nA) for varying temperatures for the COM2 process.

simulated very high-temperature (125°C) operation, appropriate in hard to control environments like automobiles.

3.3 Benchmarks

We evaluate our results using benchmarks from the SPEC CPU2000 suite [The Standard Performance Evaluation Corporation 2000]. 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.

Table III. Benchmark Summary

	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%

For each program, we skip the first one billion instructions to avoid unrepresentative behavior at the beginning of the program's execution. We then simulate 500M (committed) instructions using the reference input set. Simulation is conducted using SimpleScalar's EIO traces, which include all external inputs and outputs into the program to ensure reproducible results for each benchmark across multiple simulations.

Table III summarizes the benchmarks used and their prediction accuracies with several predictors.

3.4 Energy Evaluation

The net energy savings from branch-predictor decay must account for two opposite effects resulting from decay. Turning off V_{dd} in idle counters or rows can prevent them from leaking; however, decaying the counters causes them to lose their stored history information stored, possibly degrading the performance of the branch predictor and resulting in more energy spent on misspeculation and a longer execution time.

We use Wattch [Brooks et al. 2000] to measure the total processor dynamic energy with and without applying decay, and subtract them to obtain the dynamic energy overhead. For leakage power, Yang et al. [2001] estimate that with a 110°C operating temperature, a 1 GHz operating frequency, a supply

voltage of 1.0 V, and a threshold voltage of 0.2 V, 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 [Diodato 2001].

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 0.014, 0.057, and 0.0517 nJ 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.

4. DECAY WITH 6T STATIC CELLS

To establish the effectiveness of basic branch-predictor decay, we first need to examine how decay interacts with basic branch predictor. Using conventional 6T cells and gated-V_{ss}/V_{dd} techniques, we simulated three basic branch predictors—a bimodal predictor, a gshare predictor, and a hybrid predictor.

4.1 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

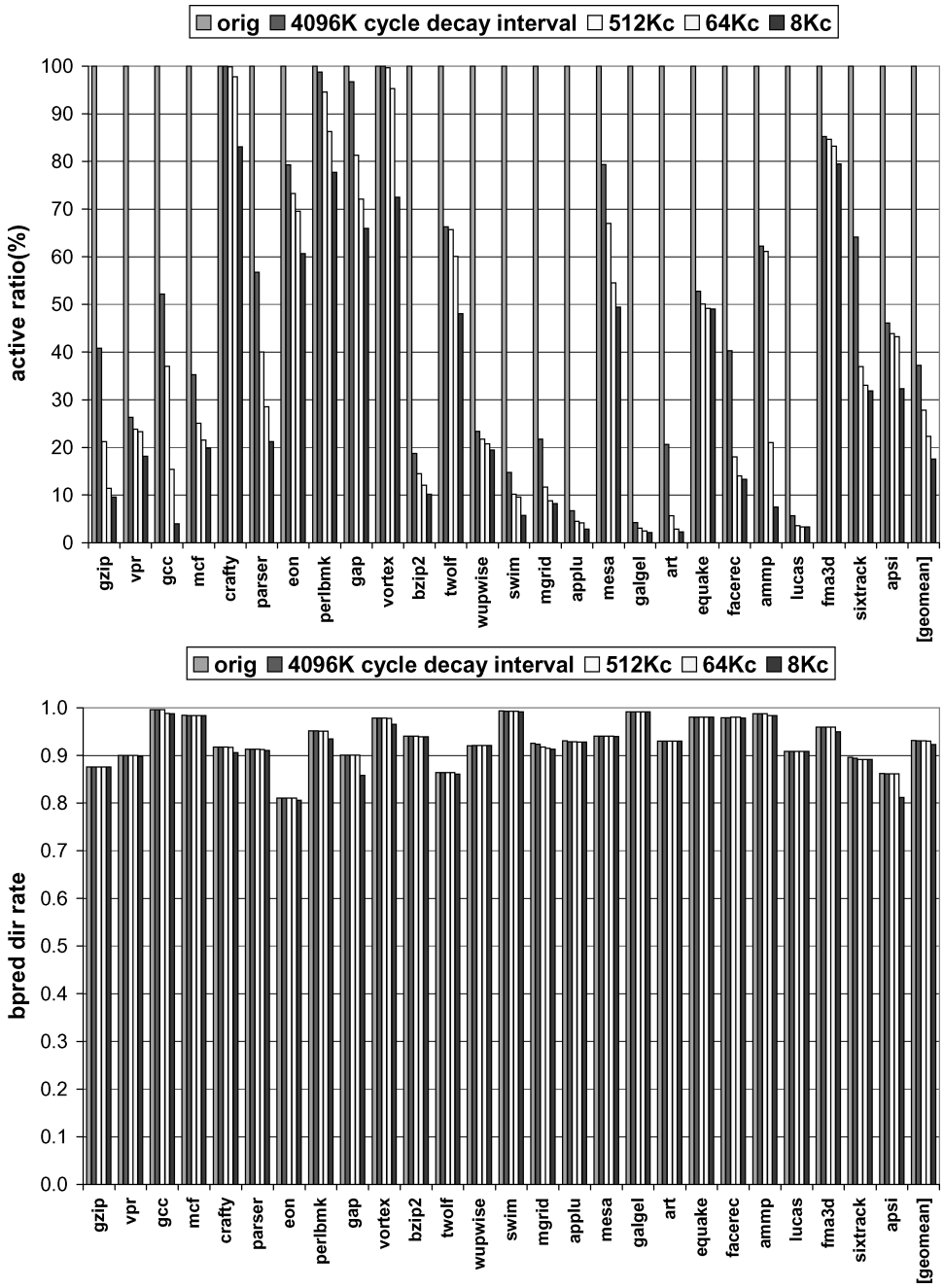

Fig. 6. Active ratio (Top) and prediction success rate (Bottom) for a 4k-entry bimodal predictor.

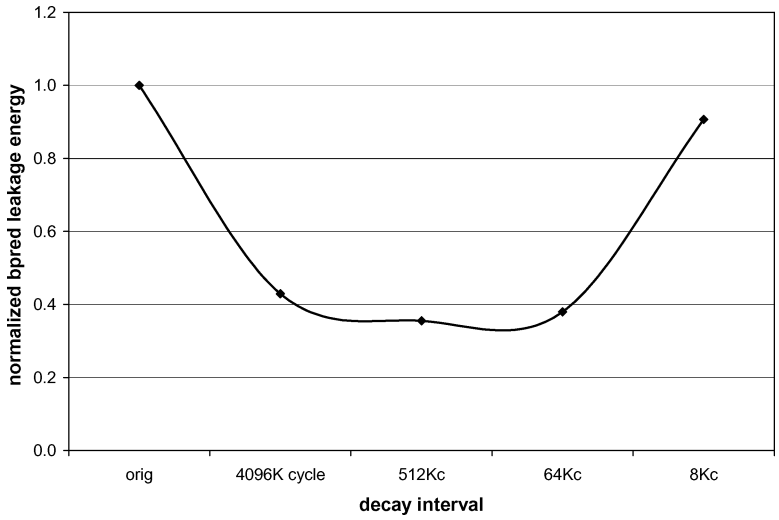

Fig. 7. Normalized leakage energy for the 4k bimodal predictor.

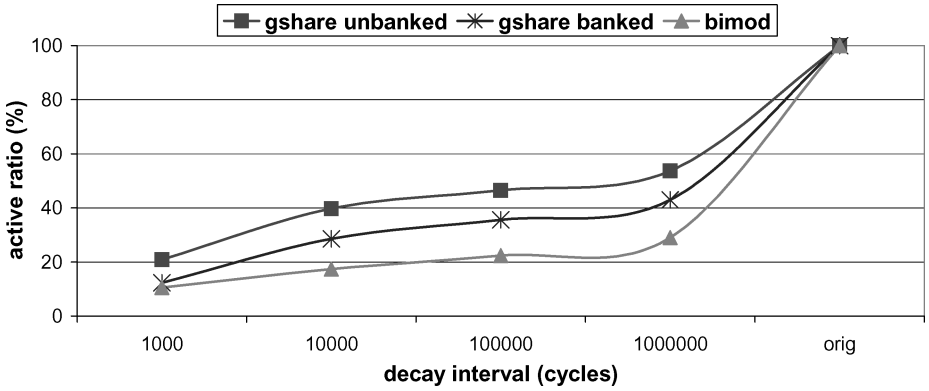

Fig. 8. Mean active ratio for unbanked and banked gshare predictors and a bimodal predictor with different decay intervals. The rightmost label "orig" corresponds to nondecaying predictors.

the decay status bits as well as any dynamic-energy overhead due to extra mispredictions or longer run time. As the decay interval lengthens, fewer mispredictions are induced, the extra dynamic energy incurred becomes minimal, and over 60% of the original leakage energy can be saved.

4.2 Gshare Predictors

To evaluate if decay will be effective for a gshare predictor, we must first look at the active ratio over various interval lengths. 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. Because gshare is designed to spread branch state across the predictor, the active ratio will be larger. Yet, significant numbers

of rows remain untouched. This indicates that decay-based techniques still show significant promise for addressing leakage concerns.

Figure 8 shows data for an unbanked as well as 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 Hu et al. [2001]. Note that energy savings are presented in terms of net leakage savings in the branch predictor, not the total processor, to separate net leakage-energy savings in the branch predictor from potential net leakage savings independent of the branch predictor.

Figure 9 shows the active ratio and branch misprediction rates for an unbanked gshare predictor on a per-benchmark basis. The geometric mean active ratio across the 26 benchmarks is 46% for a 64k-cycle decay interval, with a negligible drop in predictor accuracy. While 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 overall reduction in leakage energy is 41% for a 64k-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 decay status bits. Second, 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 deactivates a row that harms prediction accuracy. This extra dynamic-power overhead, however, decreases with longer 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.

4.3 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.

4.3.1 *Selection Policy.* The selection policy refers to the policy for choosing a prediction from one of the two component predictors. In a nondecaying 21264-style hybrid predictor, the chooser makes this decision using the global history (see Figure 1). 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.

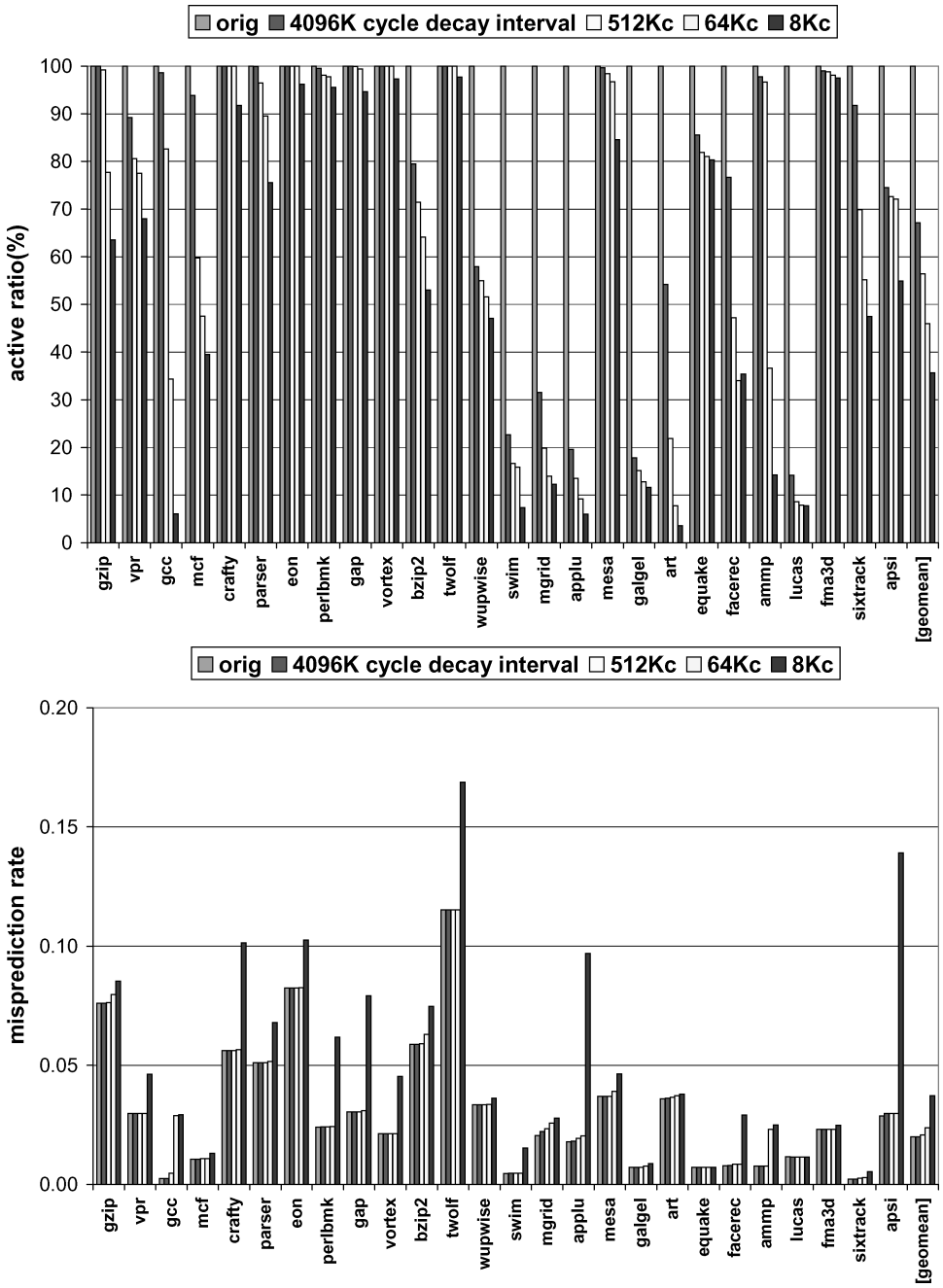

Fig. 9. Active ratio (Top) and misprediction rate (Bottom) for unbanked gshare predictor.

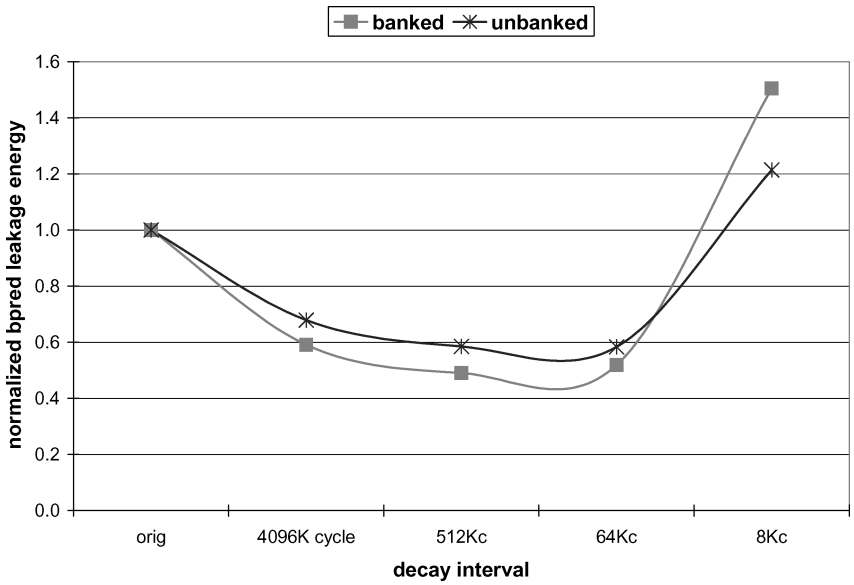

Fig. 10. Normalized leakage energy for gshare branch predictor for both unbanked and banked predictors.

This policy is called "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.

4.3.2 *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 wake up 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 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 1). 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."

4.3.3 *Results.* Figure 11 details the active ratio and branch misprediction rate for naive decay, which always wake up any rows that are accessed, with a 21264-style hybrid predictor. We see that even though the active ratios are higher (worse) 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 multitable structure, we compute it over all prediction and chooser bits in the structure. Overall, as Figure 12

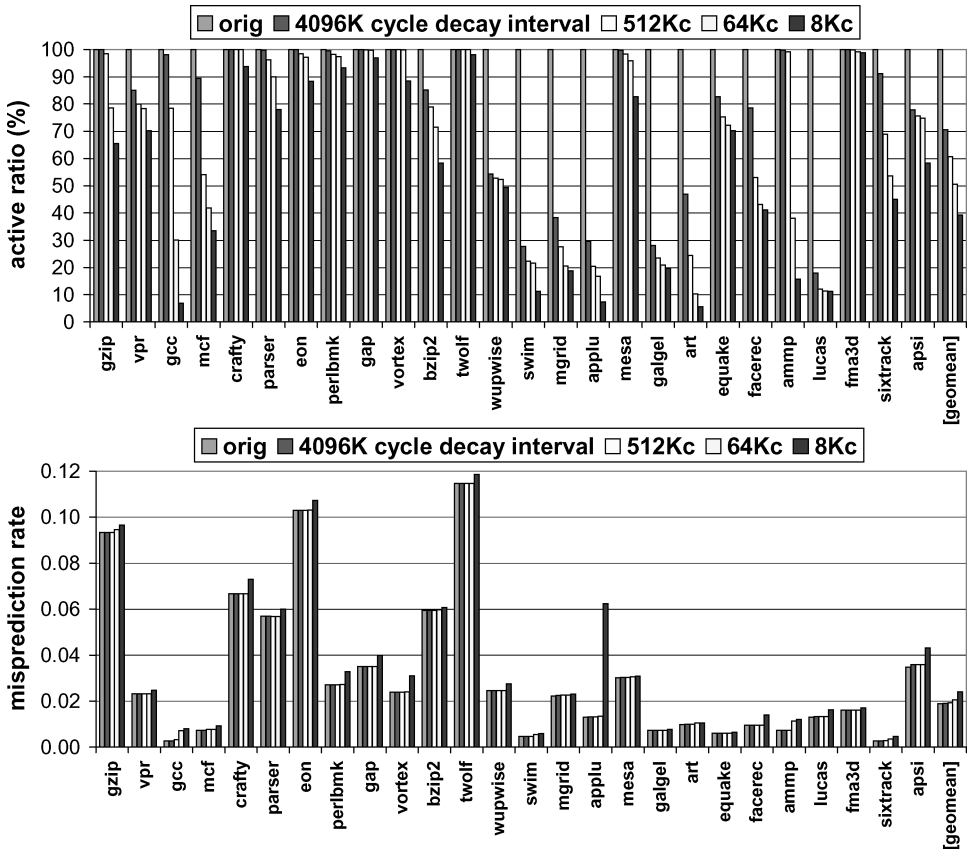

Fig. 11. Active ratio (Top) and misprediction rate (Bottom) for 21264's hybrid predictor.

shows, naive decay realizes strong reductions in energy savings—40% for a 64k-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. We achieve this by exploiting information from the chooser. The chooser points to a particular component for a reason—that is, one component is performing better than the other, and thus we only need information from the superior component.

4.4 Decay with the BTB

In addition to the structure for predicting the direction of conditional branches, a BTB [Holgate and Ibbett 1980; Losq 1982] 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

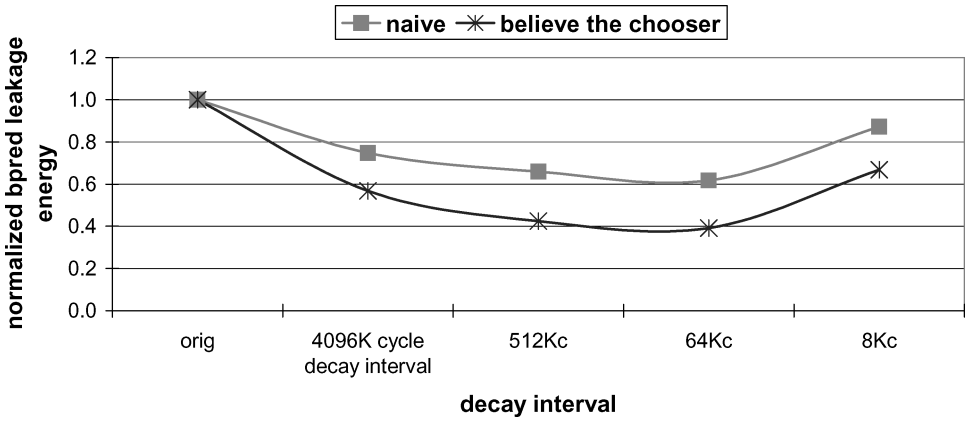

Fig. 12. Normalized leakage energy of 21264-style hybrid predictor with naive and "believe the chooser" wake up policies.

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 (Figure 13).

We evaluated a 2048-entry, 4-way associative BTB, which appears in the Intel PIII processor [Diefendorff 1999]. Except for *vortex*, *perlbmk*, and *crafty*, most benchmarks have a very low active ratio, below 20% or even below 10%. This is no surprise, considering that the static footprints of most benchmarks in Table I 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 8k cycles or less. Overall, with decay intervals of 64k cycles or more, about 90% of the BTB leakage energy can be saved.

4.5 Summary and Discussion

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 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. [2000] and Parikh et al. [2002] suggest 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 predictor, but the power savings achieved with

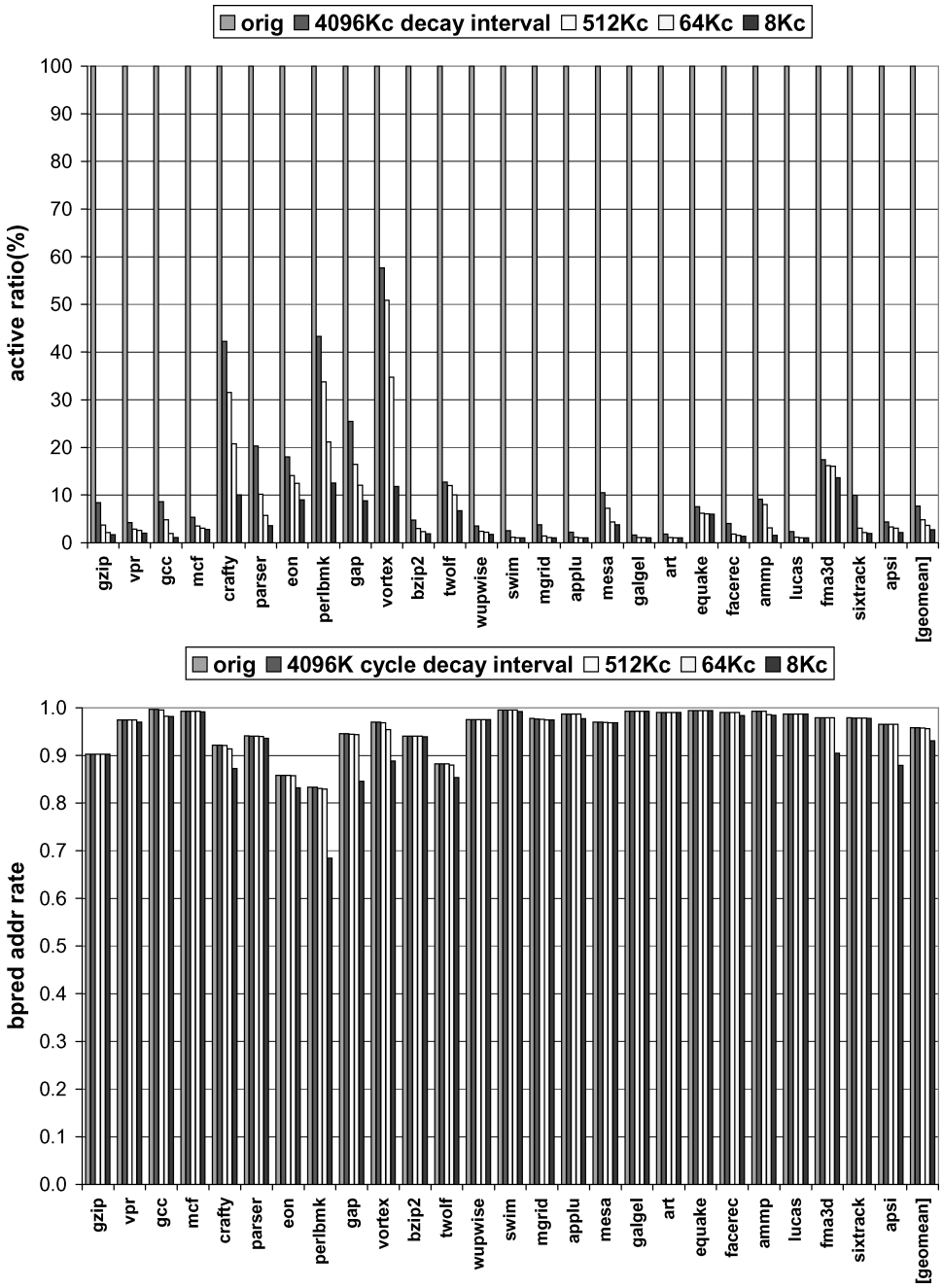

Fig. 13. Active ratio (Top) and accuracy (Bottom) for the BTB with different decay intervals.

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 [Parikh et al. 2002]. Overall, as branch prediction accuracy and execution time were increased negligibly, the reduction in leakage energy consumed by the branch-predictor unit causes the processor to save energy as a whole.

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 to 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 in 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. [2001] show that for caches, even better decay rates can be achieved with decay intervals that adapt to changing program behavior. Other work by Zhou et al. [2001], Velusamy et al. [2002], and Sankaranarayanan and Skadron [2004] also improved on cache decay by selecting better policies for adapting the decay interval. While studying adaptive branch-predictor decay was beyond the scope of this paper, it remains worth investigating because our data showed that some benchmarks suffered no performance loss even with a short, 8k-cycle decay interval (Figure 14) (and would get even more benefit from decay), while others suffered drastically.

A further consideration is that banked and multitable organizations provide substantial benefits in reduced *dynamic* energy, by reducing word- and bit-line lengths. Furthermore, when decay is applied to multitable 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, also 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.

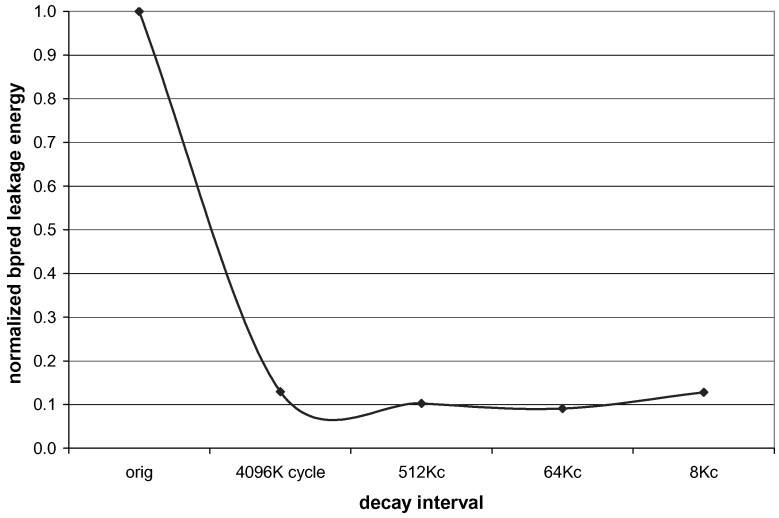

Fig. 14. Normalized leakage energy for the BTB with different decay intervals.

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 [Lipasti et al. 1996] and prefetch address predictors [Lai et al. 2001] are organized similarly to branch predictors, an interesting future effort will be to apply strategies found in this paper to these structures.

5. DECAY WITH 4T RAM CELLS

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.

5.1 The Quasi-Static 4T Cell

Basic 4T DRAM cells are well established and described in introductory VLSI textbooks and various articles [Lyons et al. 1987; Noda et al. 1998; Wolf 1998]. 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 two-third of the area. For example, under a 0.18 μm process, an ordinary 6T cell occupies an area of 5.51 μm^2, whereas a 4T cell would occupy an area of 3.8 μm^2 (i.e., 69% of the area [Diodato et al. 1998]).

Essentially, the 4T cell is just as fast as a 6T cell. While our data demonstrate a slight speed disadvantage, the difference is so small that coupled with the

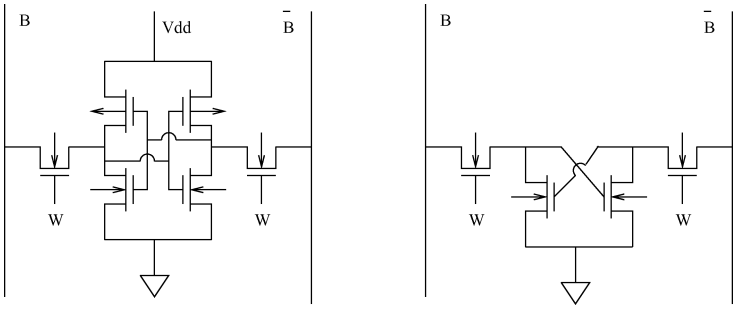

Fig. 15. Circuit diagrams of the 6T SRAM cell (left) and the 4T quasi-static RAM cell (right).

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 because 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.

Precharging, in both SRAM and DRAM designs, saps some of the power savings; any charge loss, power dissipation, or cell behavior that is a result of precharging is accounted for in our simulation. There are various forms of precharge (full-V_{dd}, half-V_{dd}, 2/3-V_{dd}) and they vary from those that are normally on to those that are normally off. We use full-V_{dd} in our simulations. The precharge control circuitry in the simulation is normally off; the precharge is on for 33% of the time and then off at the specific moment of a read or write. Whether the precharge transistors are on or off, residual charge on the bit-line does leak through the access transistors and eventually is lost to the ground node. Even when all residual charge is removed from the bit-line, the leakage of the precharge transistors supply a low level of current flow through the access transistors.

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 1.5 V 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 30 mV (for 1.5 V designs).

Thus 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.

5.2 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 0.16 μm 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 achievable 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 (σ) ($\gg$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 six-σ 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. 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 3.3 V I/O transistors—readily available in COM2 technology—to replace the 1.5 V transistors while maintaining 1.5 V 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 3.3 V transistors while operating them at 1.5 V 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) that requires its transistors to be fully biased to work correctly.

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

Table IV. Hold Times (in ns) for 1.5 and 3.3 V Versions of 4T Cells at Different Operating Temperatures

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

For a 1 GHz (1 ns cycle-time) processor, one can also consider these retention times as cycle counts.

Table V. Read Energies (in fJ) for Nominal 1.5 and 3.3 V 4T Cells at 85°C for Different States of Decay (i.e., Different Internal Voltages)

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

Values include energy dissipated in the cell, precharge circuits, and sense amplifier. The 3.3 V 4T cells are operated at 1.5 V. An internal voltage of 1.5 V is not available in the 1.5 V 4T because the 4T cell internal node voltages are constrained by the cut-off region of the MOSFETS used as access transistors. This affect could be overcome by overdriving the wordlines, as in conventional DRAM design.

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 100 mV. We used 100 mV as our criteria for the minimum voltage at which 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, as shown and verified in our circuit models, produces 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.

5.3 Sense Amplifier Considerations

One implication of the 4T cell is that it becomes progressively more expensive to read as it decays (similar to the behavior of DRAM). As the cell decays, the sense amp must work harder to detect and amplify the differential signal on the bitlines. 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 1.5 and 3.3 V devices) and for various stages of decay. Table V gives the read energy for the 4T cell at various internal voltages.

5.4 Locality Considerations

Granularity is also relevant in the 4T design but here it stems from the way 4T cells are refreshed. As reading a single cell asserts the wordline and refreshes all the cells in the accompanying row, cells that would have decayed if left alone get refreshed coincidentally by nearby active cells. This leads to the observation that even 4T cells with short retention times may not actually lose data as quickly if the row size is long enough. Thus retention time selection goes together with locality granularity because large row granularities make the apparent rate of refresh much higher. (Segmented wordlines would allow more selective refresh but these designs are outside the scope of this paper.)

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.

5.5 Metastability

A key issue regarding 4T structures is the fact that metastability problems may be a concern 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 [Kaxiras et al. 2001], 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.

5.6 Reliability, Determinism, and Controlled Operation

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. [1987] describe a refresh mechanism that on a periodic basis momentarily strobes the address lines after the bitlines 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 bitlines 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, and (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 reenable refresh, reverting to nondecay mode. One method to achieve this is by using on-chip sensors to automatically trigger the refresh when necessary.

5.7 Trends 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. For example, simulating under SPICE using a future technology (65 nm), at 85°C a 3.3 V 4T cell has a 1.4 μs retention time, translating into 1400 processor cycles at 1 GHz. However, according to the ITRS roadmap, by the time 65 nm is reached the processor clock should be approximately 7.5 GHz, translating into an effective retention time of just over 10,000 cycles. Coupled with architectural techniques to extend the retention time, even with the impact of process technology, we can still field an effective retention time for decay. 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;
—precharge and access transistors leakage is a mechanism of pure charge loss and complete power waste in a 6T-SRAM;
—precharge and access transistor leakage tends to replenish the internal node charge lost due to the leakage of other transistors in the 4T-DRAM.

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 reasonable middle ground between 6T and 1T.

5.8 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. One way to affect retention times is to add devices such as resistors or capacitors to the basic 4T cell [Diodato et al. 2001]. 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.

The ability to control retention times is key to the applicability of 4T cells to branch-predictor designs, as the retention time is dependent on temperature. Thus, one can adapt the natural decay interval (in cycles) more closely to the desired decay interval in response to the given temperature range. As described below, refresh circuitry can be included to be used as a fail-safe if the temperature exceeds a certain threshold.

In addition, retention times can be controlled using architectural techniques, such as using multiple refreshes to prolong retention time, accomplished either statically or adaptively. Moreover, as described above, the structure of the cache may make the apparent rate of refresh much higher; longer row granularities can compensate for a shorter decay interval. One thus can design the structure to orient towards longer rows for an increased decay interval (and a larger safeguard against performance loss) or shorter rows for greater leakage-energy savings.

6. RESULTS FOR 4T-BASED BRANCH PREDICTORS

To illustrate the effects of the increase in the misprediction rate due to decay, we simulated a processor with the direction predictor and BTB built with 4T structures. We simulated under the worst-case scenario: small static leakage in comparison to dynamic energy, which correlates closely with the COM2 process. Thus we use slow-decaying 3.3 V 4T cells in our design, operating at 85°C, leaving us a decay interval of 57,200 cycles for both the direction predictor and the BTB.

These values can be adjusted using the techniques as described in Section 5; for example, if the decay interval of 57,200 cycles is too long, we can instead use 1.5 V 4T cells, which have a shorter decay interval (8000 cycles), and extend that retention time appropriately.

6.1 4T Decay Results

6.1.1 *Gshare.* Figure 16 shows the normalized execution time and misprediction rate of the 4T branch predictor versus a standard (nondecaying) branch predictor. We see that the misprediction rate and thus the execution time are virtually unchanged. In fact, due to the random effects of reading the decayed

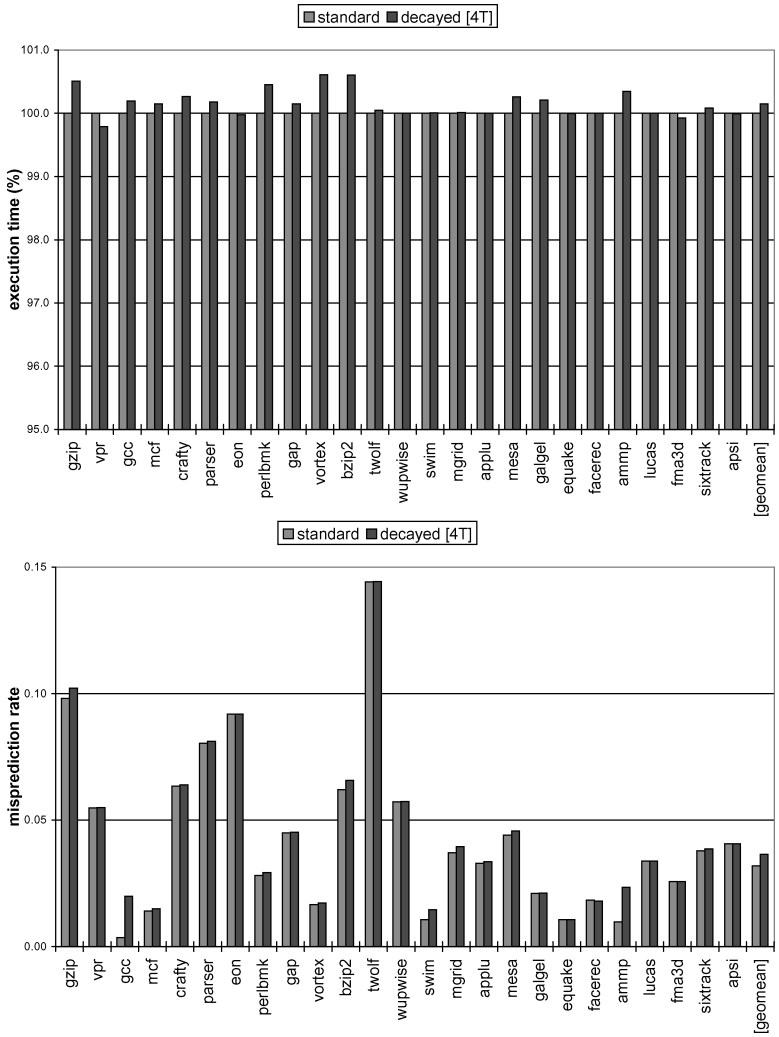

Fig. 16. Normalized execution time (Top) and misprediction rate (Bottom) of standard and 4T predictors. 4T predictors produce minimal performance losses.

values, a few benchmarks actually improve 2–3%. Overall, the accuracy was affected less than 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, nondecaying 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 nondecayed 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

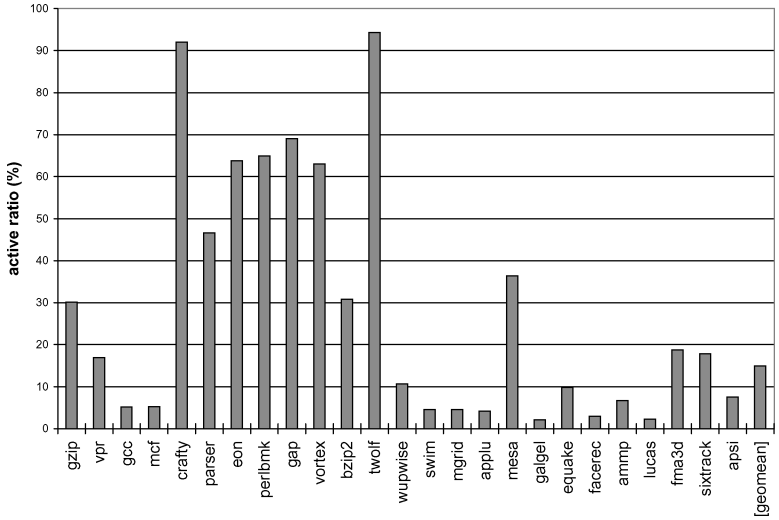

Fig. 17. Active ratio of a 4T-based predictor.

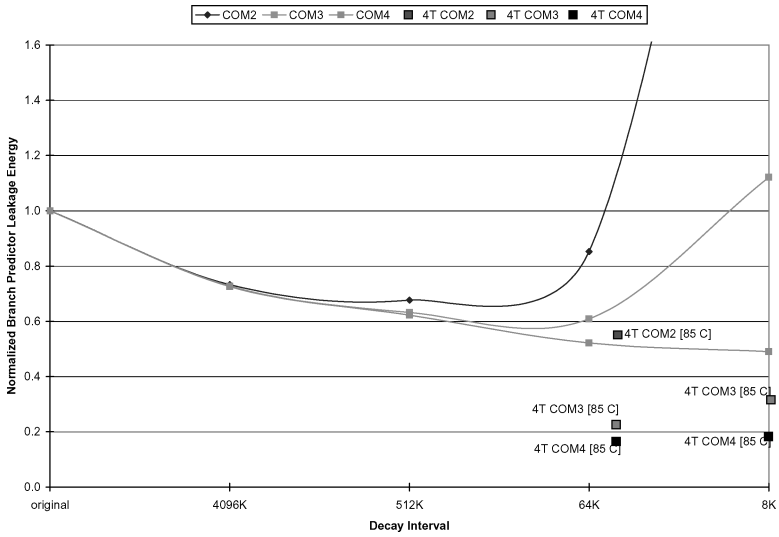

Fig. 18. Normalized leakage energy for the 4T branch predictor at 57.2k and 8k cycles.

on the same row, however, 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 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 lower 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,

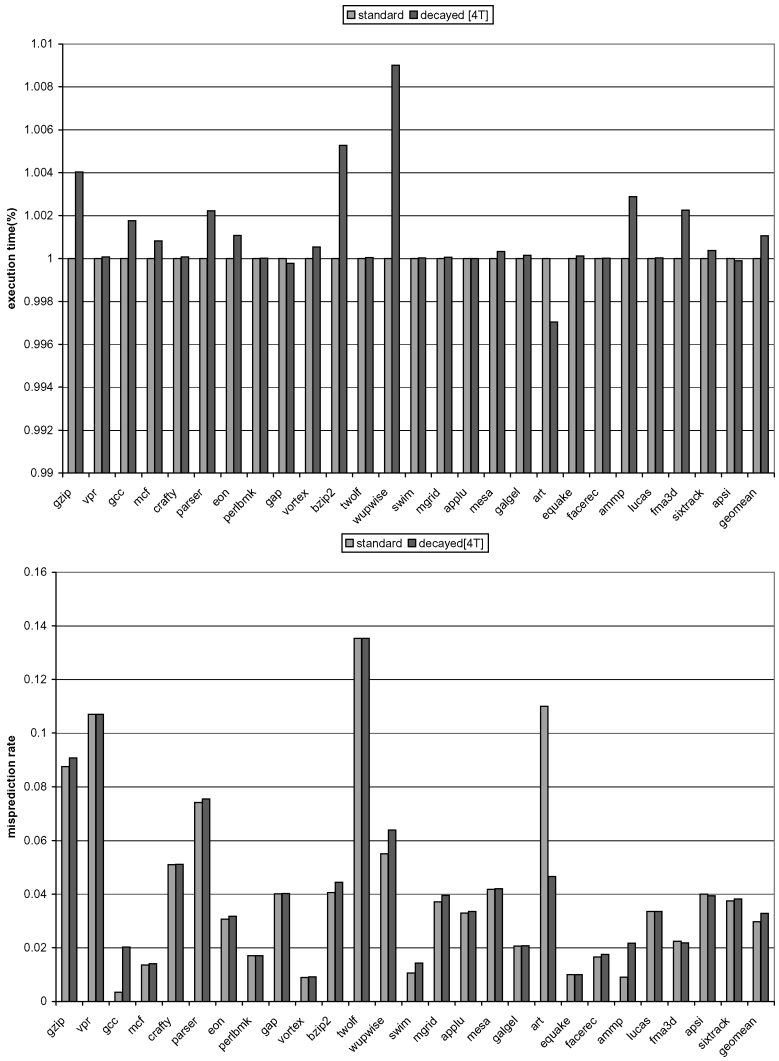

Fig. 19. Normalized execution time (Top) and misprediction rate (Bottom) of standard and 4T bimode predictors. 4T predictors produce minimal performance losses.

a 6T-based decaying predictor on a COM3 process would actually consume more energy than a standard, nondecaying 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 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.

6.1.2 *Bimode Results.* Figure 19 shows the normalized execution time and misprediction rate of a more complex predictor, the bimode. In this configuration

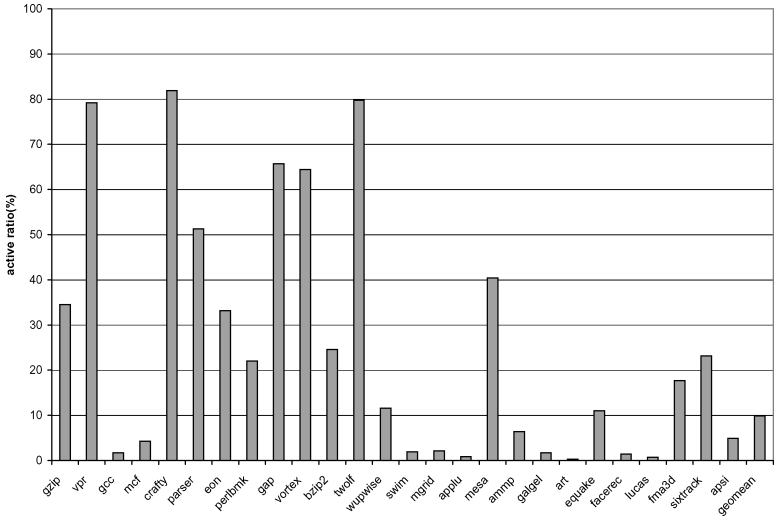

Fig. 20. Active ratio of a 4T-based bimode predictor.

we used an aggressive bimode predictor composed of two 16k PHTs activated by an 8k selector. We see that execution time is moved very little; misprediction rate is increased roughly 0.3%, translating into a performance loss of 0.1%. The complexity of the predictor helps alleviate the performance loss due to the misprediction, as a single static branch covers a larger area than, for example, a bimodal predictor, which raises the apparent refresh rate. This is confirmed by the fact that bimode predictors suffer less as the decay interval is decreased compared to the gshare. The results for varying the decay interval of gshare will be discussed in Section 6.2.

Figure 20 shows the active ratio of the bimode predictor. The active ratio appears to be smaller than that of the gshare because the active ratio is an average of all three components of the bimode—one of which is a very simple bimodal predictor. While the bimodal component is half the size of the PHTs, its active ratio is significantly smaller than that of the PHTs and thus lowers the overall active ratio.

6.1.3 *BTB Results.* The results for the 4T BTB are similar to the 6T BTB. 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.

6.2 Sensitivity to Decay Interval

Figure 21 shows the impact of decay interval over execution time and predictor accuracy. Shorter (aggressive) decay intervals increase the decay ratio, but at the cost of performance. For example, at a 256-cycle decay interval, the active ratio is less than 1%, but the execution time is increased 11% due to the branch

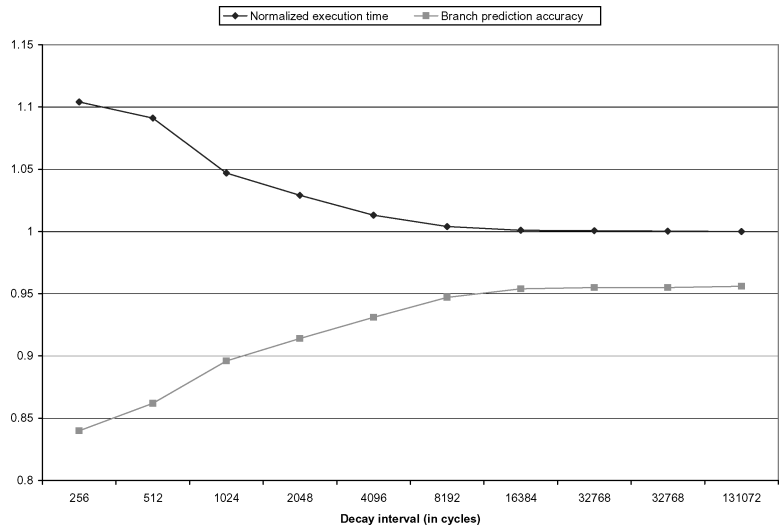

Fig. 21. Normalized execution time and branch-predictor accuracy for the 16k gshare predictor at varying decay intervals.

predictor suffering a 10% drop in accuracy. Overall, the normalized leakage energy for the branch predictor is increased 7 times.

While temperature can cause the 4T cell to leak more rapidly (and thus shorten the decay interval), it is important to restate that a randomized value in the directional counters may still be correct, and thus the effect of accidentally decaying branch predictors is not as severe as decaying cache lines. More, as mentioned earlier, retention time can be controlled using the techniques mentioned in Section 5. For example, if the temperature is beyond some threshold such that cells are leaking too quickly, a stand-by refresh circuit can be enabled to return the branch predictor into a nondecaying mode.

6.3 Summary and Discussion

We see that using quasi-static 4T cells allows us to build naturally decaying branch predictors with minimal impact on performance. Furthermore, predictors built using 4T-based decay offer additional benefits over those already shown with decay with 6T cells. We show that 4T RAM cells are as much as one-third smaller in area and 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.

Thus our 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 naturally suited to exploiting decay in large structures over our benchmarks: 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. Our results in Section 5.7 show that our technique will be viable even in future technologies.

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.

7. 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. [2001] described an instruction cache organization that disables ways of a set-associative cache to match the capacity currently needed by the executing program.

Several papers [Powell et al. 2000; Kaxiras et al. 2001; Hu et al. 2002] 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. [2002a] extended decay techniques to the branch predictor. Most of the above techniques assume technologies such as gated-V_{dd} or supply-voltage gating to control whether or not a portion of the structure is enabled or disabled. More recently, Hanson et al. [2001] presented a comparison of leakage-control techniques, 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 [Roy 1998] save more overall energy for structures that should have fast lookup times (like first-level caches).

Zhou et al. [2003] proposed turning off only the data portion of the cache, leaving the tag portion active in order to monitor and exploit miss information. Furthermore, using adaptive cache decay as an example, Velusamy et al. [2002] applied formal feedback control mechanisms to improve leakage savings by implementing an adaptive controller which varied decay intervals dynamically.

Sankaranarayanan and Skadron [2004] also improved on cache decay by selecting better policies for adapting the decay interval. Flautner et al. [2002] 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. Extending gated-V_{dd}, Agarwal et al. [2002] DRG-Caches use data-retentive leakage-aware cells to reduce leakage energy. Alternatively, Azizi et al. [2002] introduced an asymmetric dual-V_t SRAM cell. Another leakage control technique saving the cache state was described by Heo et al. [2002]. Combining state-preserving and state-destroying techniques, Li et al. [2002] 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. [2002] presented techniques using the compiler to improve instruction cache leakage.

A separate, architectural way of improving energy consumption performance of structures such as the cache was proposed by Balasubramonian et al. [2000] and Yang et al. [2002], in which the cache is reconfigured in order to best support the working set of the data with as little wasted cache as needed. Resizable caches are indeed orthogonal to decay techniques and also take advantage of the fact that some applications require very little either the cache or from the branch predictor. However, the decay approach is simple, requires very little hardware, and can potentially use very small operating intervals. Still, there is nothing preventing a resizable cache from being built with decay, and combining the benefits of both.

Taking a different approach, Juang et al. [2002] looked at eliminating V_{dd} altogether in replacing the standard SRAM cell with a dynamic, quasi-static 4T cell. Using the quasi-static cell allows for an easily built, naturally decaying cache. In addition to caches, Hu et al. [2002b] described ways of saving leakage in branch predictors by implementing them with quasi-static cells. Finally, Hu et al. [2003] 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.

8. CONCLUSIONS

In this paper, we examine implementations of decay-based leakage control using quasi-static memory cells. Cache decay, first proposed in Kaxiras et al. [2000], and Kaxiras et al. [2001], 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.

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 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} [Yang et al. 2001], 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 6T cell design and gated-V_{dd}. Specifically, while gated-V_{dd} tries to shut off cache lines, the two load transistors in 6T cells are merely wasting charge. To resolve this conflict, we proposed using 4T 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 4T decay implementation achieves significant savings in leakage power. While 4T decay typically offers greater savings than 6T 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 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 now only 7–10% 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. In future processors, as branch predictors potentially will grow in size very quickly, outpacing the growth of the cache, the power consumption of the branch predictor will be an important problem. Most broadly, the paper prompts a rethinking of how transient data can best be exploited in designing power-efficient processors.

ACKNOWLEDGMENTS

We would like to thank Polychronis Xekalakis and Yan Zhang for their help in simulating the 4T cell under future technologies in SPICE.

Margaret Martonosi and Doug Clark's research on energy-efficient processors is supported in part by NSF ITR grant CCR-0086031. In addition, Margaret Martonosi gratefully acknowledges funding and equipment donations from Intel and IBM.

Kevin Skadron's research was supported in part by NSF grants CCR-0105626, an NSF CAREER award (CCR-0133634), and a grant from Intel MRL.

Stefanos Kaxiras would like to thank Intel for equipment donations.

We also thank the anonymous reviewers for their detailed, helpful suggestions.

REFERENCES

AGARWAL, A. ET AL. 2002. DRG-Cache: a data retention gated-ground cache for low power. In *Proceedings of the 39th Design Automation Conference*.

AZIZI, N. ET AL. 2002. Low-leakage asymmetric-cell SRAM. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

BALASUBRAMONIAN, R. ET AL. 2000. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd International Symposium on Microarchitecture*.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architecture-level power analysis and optimizations. In *Proceedings of the ISCA-27*.

BURGER, D. C. AND AUSTIN, T. M. 1997. The SimpleScalar tool set, Version 2.0. *Computer Architecture News 25*, 3 (June), 13–25.

CHANG, P.-Y., HAO, E., AND PATT, Y. N. 1995. Alternative implementations of hybrid branch predictors. In *Proceedings of the Micro-28*. 252–57.

DIEFENDORFF, K. 1999. Pentium III = Pentium II + SSE. *Microprocessor Report*.

DIODATO, P. ET AL. 1998. Merged DRAM-LOGIC in the Year 2001. In *Proceedings of the IEEE International Workshop on Memory, Technology, Design, and Testing*.

DIODATO, P. ET AL. 2001. Embedded DRAM: An element and circuit evaluation. In *IEEE Custom Integrated Circuits Conference*.

DIODATO, P. W. 2001. Personal communication.

FLAUTNER, K. ET AL. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th International Symposium on Computer Architecture*.

GWENNAP, L. 1996. Digital 21264 sets new standard. *Microprocessor Report*, 11–16.

HANAMURA, S. ET AL. 1987. A 256K CMOS SRAM with internal refresh. In *The 1987 IEEE International Solid-State Circuits Conference*.

HANSON, H. ET AL. 2001. Static energy reduction techniques for microprocessor caches. In *Proceedings of the 2001 International Conference on Computer Design*. 276–83.

HEO, S. ET AL. 2002. Dynamic fine-grain leakage reduction using leakage-biased bit lines. In *Proceedings of the 29th International Symposium on Computer Architecture*.

HOLGATE, R. W. AND IBBETT, R. N. 1980. An analysis of instruction fetching strategies in pipelined computers. *IEEE Transactions on Computers C-29*, 4 (Apr.), 325–329.

HU, Z. ET AL. 2002a. Applying decay strategies to branch predictors for leakage energy savings. In *Proceedings of the 2002 International Conference on Computer Design*.

HU, Z. ET AL. 2002b. 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*.

HU, Z., JUANG, P., SKADRON, K., CLARK, D., AND MARTONOSI, M. 2001. Applying Decay Strategies to Branch Predictors for Leakage Energy Savings. Tech. rep., CS-2001-24, University of Virginia.

HU, Z., KAXIRAS, S., AND MARTONOSI, M. 2002. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. *ACM Trans. Comput. Syst*.

HU, Z., KAXIRAS, S., AND MARTONOSI, M. 2003. Timekeeping techniques for predicting and optimizing memory behavior. In *The 2003 IEEE International Solid-State Circuits Conference*.

BUTTS, J. A. AND SOHI, G. 2000. A static power model for architects. In *Proceedings of the 33rd International Symposium on Microarchitecture*.

JIMÉNEZ, D. A., KECKLER, S. W., AND LIN, C. 2000. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd International Symposium on Microarchitecture*. 67–77.

JUANG, P. ET AL. 2002. Implementing decay techniques using 4T quasi-static memory cells. *Comput. Arch. Lett*.

KAXIRAS, S., HU, Z., ET AL. 2000. Cache-line decay: A mechanism to reduce cache leakage power. In *Workshop on Power-Aware Computer Systems (PACS)*. In conjunction with ASPLOS-IX.

KAXIRAS, S., HU, Z., AND MARTONOSI, M. 2001. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th International Symposium on Computer Architecture*.

KESHARVARZI, A. ET AL. 1997. Intrinsic iddq: Origins, reduction, and applications in deep sub-μm low-power CMOS IC's. In *Proceedings of the IEEE International Test Conference*. 146–155.

LAI, A., FIDE, C., AND FALSAFI, B. 2001. Dead-block prediction and dead-block correlating prefetchers. In *Proceedings of the 28th International Symposium on Computer Architecture*.

LI, L. ET AL. 2002. Leakage energy management in cache hiearchies. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*.

LI, Y. ET AL. 2004. State-preserving vs. non-state-preserving leakage control in caches. In *Proceedings of the 2004 Design, Automation and Test in Europe (DATE)*.

LIPASTI, M., WILKERSON, C. B., AND SHEN, J. P. 1996. Value locality and load value prediction. In *Proceedings of the ASPLOS-VII*. 138–47.

LOSQ, J. J. 1982. Generalized history table for branch prediction. *IBM Tech. Discl. Bull. 25*, 1 (June), 99–101.

LYONS, R. ET AL. 1987. CMOS static memory with a new four-transistor memory cell. In *Proceedings of the 1987 Stanford Conference On Advanced Research in VLSI*. 111–132.

MCFARLING, S. 1993. Combining branch predictors. Tech. Note TN-36, DEC WRL.

NODA, K. ET AL. 1998. A 1.9 μm^2 loadless CMOS four-transistor SRAM cell in a 0.18 μm logic technology. *IEDM Tech. Dig.*, 847–850.

PARIKH, D., SKADRON, K., ZHANG, Y., BARCELLA, M., AND STAN, M. 2002. Power issues related to branch prediction. In *Proceedings of the HPCA-8*. 233–244.

POWELL, M. ET AL. 2000. Gated-V_{dd}: A circuit technique to reduce leakage in cache memories. In *Proceedings of the International Symposium on Low Power Electronics and Design*.

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

SANKARANARAYANAN, K. AND SKADRON, K. 2004. Profile-based adaptation for cache decay. *ACM Trans. Archit. Code Optim*. in press.

SCHUSTER, S., TERMAN, L., AND FRANCH, R. 1987. A 4-device CMOS static RAM cell using subthreshold conduction. In *Symposium on VLSI Technology, Systems, and Applications*.

SEMICONDUCTOR INDUSTRY ASSOCIATION. 2001. From website: The international technology roadmap for semiconductors. Available at `http://public.itrs.net/Files/2001ITRS/Home.htm`.

SEZNEC, A. ET AL. 2002. Design tradeoffs for the alpha EV8 conditional branch predictor. In *Proceedings of the 2002 International Symposium on Computer Architecture*.

SMITH, J. E. 1981. A study of branch prediction strategies. In *Proceedings of the 8th International Symposium on Computer Architecture*. 135–48.

SONG, P. 1997. UltraSparc-3 aims at MP servers. *Microprocessor Report*, 29–34.

THE STANDARD PERFORMANCE EVALUATION CORPORATION. 2000. Available at `http://www.spec.org`.

VELUSAMY, S. ET AL. 2002. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*. In conjunction with ISCA-29).

WOLF, W. 1998. *Modern VLSI Design: Systems on Silicon*. Prentice Hall. Prentice-Hall.

YANG, S. ET AL. 2002. Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*.

YANG, S.-H. ET AL. 2001. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*.

ZHANG, W. ET AL. 2002. Compiler-directed instruction cache leakage optimization. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*.

ZHOU, H. ET AL. 2003. Adaptive mode control: A static-power-efficient cache design. *ACM Trans. Embedded Comput. Syst.* Special issue on Power-Aware Embedded Computing.

ZHOU, H., TOBUREN, M., ROTENBERG, E., AND CONTE, T. 2001. Adaptive mode control: A static-power-efficient cache design. In *Proceedings 2001 International Conference on Parallel Architectures and Compilation*.

Received April 2003; revised December 2003 and April 2004; accepted April 2004