LRU-SEQ: A Novel Replacement Policy for Transition Energy Reduction in Instruction Caches

†*Praveen Kalla, †‡Xiaobo Sharon Hu
Dept. of CSE, Univ. of Notre Dame.
{*nkalla, shu*}*@cse.nd.edu*

Jörg Henkel
NEC Laboratories America Inc.
henkel@nec-labs.com

Abstract

Leakage energy will be the major energy consumer in future deep sub-micron designs. Especially the memory sub-system of future SOCs will be negatively affected by this trend. In order to reduce the leakage energy, memory banks are transitioned to a low-energy state when possible. This transition itself costs some energy which is termed as the transition energy. In this paper we present, as the first approach of its kind, a novel energy saving replacement policy called LRU-SEQ for instruction caches. Evaluation of the policy on various architectures in a system-level environment has shown that upto 23% energy savings can be obtained. Considering the negligible hardware impact, LRU-SEQ offers a viable choice for an energy saving policy.

1 Introduction

Static power consumption is becoming increasingly important as silicon process parameters shrink. Process technologies are fast approaching 40 nm and static power consumption would be orders of magnitude higher by 2010 [9] [20]. Ever thinning gate-oxides in CMOS transistors has introduced gate-leakage (through tunneling) in addition to sub-threshold leakage (due to lower supply voltages). Alarming data is presented in [2] that leakage power would jump from 50% to 460% of active power as technology moves from 65 nm to 45 nm, respectively. Thus it becomes crucial to integrate circuit-level leakage-current reduction techniques and architecture-level design techniques into system-level design flows in order to tackle static power consumption.

Over the past decade researchers have been increasingly paying attention to static power reduction in logic circuits([18] [6] [19] [11],etc). These circuit-level techniques, while successful in reducing leakage-power costs, bring new challenges, i.e. *transition-energy* costs which are incurred when circuits switch to shut-down modes. Since caches consume a considerable amount of area on a chip, they would also account for a significant contribution to transition energy costs.

In this paper we present as the first approach of its kind a novel energy saving replacement policy called LRU-SEQ for instruction caches that will effectively reduce static energy consumption and as such, reduce total cache energy by 20%-28% (with an average of 23%). The paper is structured as follows: related work is described in Sec. 2, motivation for *transition-energy* reduction through architecture-level policies is presented in Sec. 3 and our novel LRU-SEQ cache organization is presented in Sec. 4. Our evaluation platform is presented in Sec. 5 followed by results in Sec. 6. We conclude in Sec. 7.

2 Related Work

Circuit-level leakage reduction techniques can be broadly classified into two categories. *State-preserving* techniques try to reduce static power consumption while preserving the contents of the logic cell, for example, the contents in an SRAM cell. DVS [21] selects between two voltage levels for active and sleep modes. [8] uses ABC-MTCMOS [18] to control the backgate bias to increase threshold voltage to save leakage power. Bit-line leakage control techniques are presented in [6] [14].

State-destroying techniques tend to shut down under-utilized portions of the instruction caches. These portions are identified by tracking miss-rates [22] [23], temporal activity [10], data redundancy [16]. [22] uses a gated-Vdd approach (a *state-preserving* version is presented in [1]).

While all the previous approaches concentrate on tracking down unused/redundant parts of the cache, two related parameters that have been introduced by these circuit techniques have not been tackled at an *architectural level*. These are *transition energy* (E_{Tr}) and *break-even* time. The energy overhead for a circuit to transition to/from a low-power shut-down/sleep mode is termed as the *transition energy*. The minimum number of cycles for which the circuit has to remain in a low-power mode to overcome this energy overhead is termed as the *break-even* time. As circuit-level leakage-reduction techniques become more advanced and faster (in terms of their latency), circuits will transition to/from low-power modes more often. This will eventually increase the significance of the contribution of E_{tr} to the total energy costs.

In this paper, we introduce a novel architecture-level policy that reduces the transition energy in cache sub-banks by redirecting *sequential cache fills* to the last bank accessed. By regrouping sequential accesses in such a manner, the policy not only reduces inter-bank transitions, but also increases the chances that a bank can be shut for a longer period and thus meet the requirements of the break-even time. Thus leakage energy is minimized. We describe the motivation for such a policy in the next section followed by the implications of such a policy on the hit-rate and the its hardware implementation.

3 Motivation and Basic Idea

In contrast to traditional approaches that employ heuristics to determine less frequently used cache regions, we revisit the replacement policy. Thus, the cache activity (at the granularity of a cache sub-bank) is altered in such a way that *any* circuit-level leakage-reduction technique can profit from it. The approach of a policy change is motivated by the following two observations (we will concentrate on only *state-preserving* leakage-reduction techniques in this paper).

Observation 1:*(E_{Tr}:)* Transitions to/from sleep modes usually involve charging or discharging the substrate (as in ABC-MTCMOS), or discharging pre-charged circuits as in [6]. E_{Tr} has been assumed to be in the range of 1 to 10% in [3], [4]. But as in [6], transition energy can be almost equal to the read-energy for the cache under certain criteria. Hence, transition energy can be quite significant (and as trends indicate, will significantly increase) compared to the active energy. Thus, it is clear that a policy which **minimizes transitions** (inter-bank transitions in our case) would help in **minimizing** the transition energy costs.

†This work is supported in part by NSF under grant numbers MIP-9701416 and CCR02-08992.

‡Part of the work by this author was carried out during her sabbatical visit to HongKong University of Science and Technology.

*Part of the work by this author was carried out as an intern at NEC Laboratories America.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

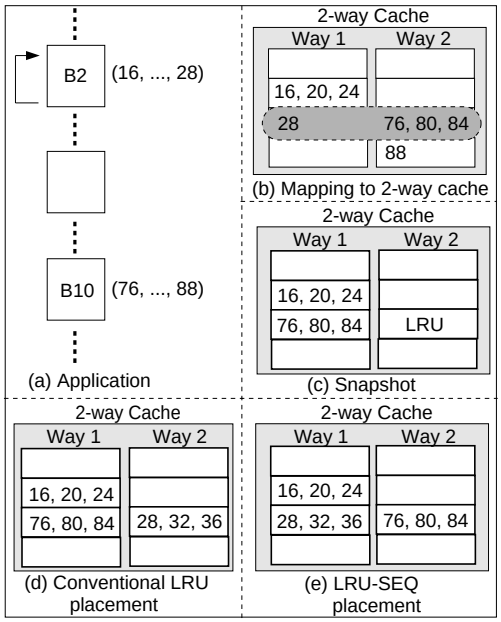

Figure 1: Motivational example for LRU-SEQ

Observation 2:*(break-even time:)* Because of the transition energy overheads as presented in the previous paragraph, circuits need to be put to sleep mode for a minimum time if there is to be any net gains. A break-even time of 200 cycles for 180nm and an optimistic 1 cycle for a 70nm technology have been presented in [6]. In [3], at least 17 cycles are needed to break-even with the transition energy overhead for a dual-Vt domino logic circuit. Thus a policy that **extends sleep times beyond break-even time** for a cache bank would greatly help in *offsetting* the transition energy overheads.

The above observations suggest that a *policy* (or) an *implementation* that affects the placement of memory blocks in a cache bank, could be tweaked in order to *save and offset* E_{Tr} overheads. Block placement in caches can be affected by address re-mapping, etc. but such *implementation* schemes will incur high hardware and timing overheads. Instead, in this article, we explore the effects of the *replacement policy*.

Example: We present the following example which illustrates the motivation behind our new policy (LRU-SEQ): (Refer to Fig. 1). Consider an application with basic blocks as shown in *(a)*. Assume that the basic block $B2$ (address range: 16-28) is a loop which executes 50 times once it starts. Furthermore, consider a 2-way set-associative instruction cache to which the basic blocks $B2$ and $B10$ (address range: 76-88) would map as shown in *(b)*. $B2$ has a partial conflict at address 28 with $B10$. Now consider a run-time snapshot of the cache as shown in *(c)*. At this point, the instruction at address 24 has been executed and instruction at address 28 has to be fetched from memory. Assume that *way-2* has the LRU position for that cache line.

The conventional LRU policy would place the fetched instructions in *way-2 (d)*. A subsequent loop execution would result in 100 bank transitions. Also, assuming that a) leakage-reduction techniques have been applied at the granularity of a bank, and b) a bank is put to sleep as soon as another bank is activated, the break-even time should be less than 3 cycles (wrt *way-2*) in order to yield an advantage (16, 20, 24 $\rightarrow$ 28 $\rightarrow$ 16, 20, 24, ...). Now, consider a policy that would constrain a *sequential-fill* to the same bank which was previously accessed. This would place the instruction at 28 in the same bank as (16, 20, 24), thus eliminating the bank's transitions. Such a placement has also automatically eliminated policy-constraints imposed by the break-even time.

It has to be noted that the motivation behind LRU-SEQ is valid only for instruction caches. Its validity for data caches has to be evaluated and is out of the scope of this paper. We now present a formal definition of our policy as well as the cache organization.

```
LRU-SEQ Policy
FOR ( every cache access ) DO
   IF ( current access == HIT )  THEN
         P_way = C_way
   Else
         IF  ( abs( C_line -- P_line) <= SEQ_DST ) THEN
               // a near-neighbor
               C_way = P_way
         ELSE
               // a far-neighbor
               C_way = LRU_way
         END
   END
      // traditional LRU update
   Update LRU state for ( C_line / C_way ).
END
```

Figure 2: LRU-SEQ policy

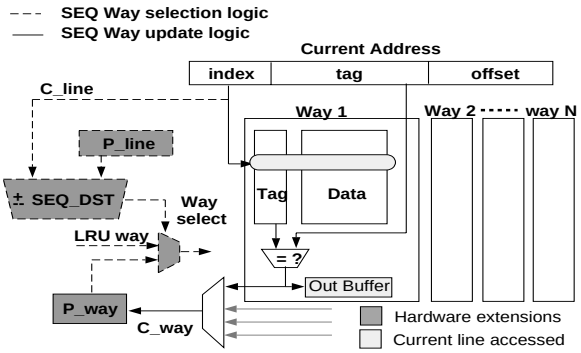

Figure 3: LRU-SEQ cache organization

4 The LRU-SEQ Cache

This section describes the LRU-SEQ policy along with the assumptions and some necessary hardware modifications.

System/Cache architecture: We assume that a *set-associative* cache is structurally divided into banks, where each *set/way* corresponds to a bank(s) (Fig. 3). We have considered two environments: **SysTRACE:** A single-issue processor with blocking caches, no prefetching or branch predictors, and perfect data cache (as our evaluations are concerned with the instruction caches). This scenario is emulating a SPARC environment. The second environment is **SysSIM :** It is emulated by the SimpleScalar [13] tool and thus referring to MIPS and Pentium architectures with branch-predictors and out-of-order execution cores. Note that we do not impose any assumptions regarding the leakage reduction technique or its effects on parameters like cache timing, etc. The intention is to keep the implementation independent from the LRU-SEQ policy. A formal definition of LRU-SEQ policy is presented next.

LRU-SEQ policy: Assuming that the *way* and *line* (index) accessed in the previous cache access cycle (a read or a write) is stored as P_{way} and P_{line}, and the current access is denoted by

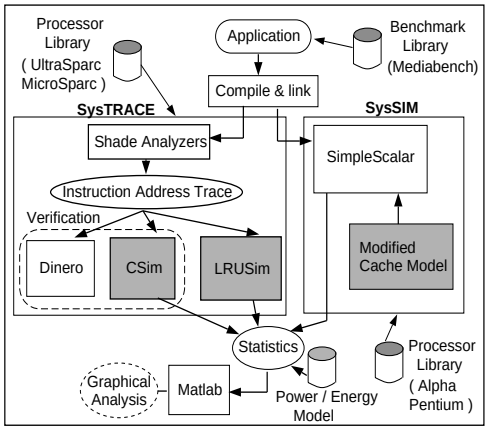

Figure 4: LRU-SEQ Evaluation Framework

C_{way} and C_{line}, respectively, LRU-SEQ operates as shown in Fig. 2. SEQ_DST is the maximum distance (in cache lines) between P_{line} and C_{line} beyond which traditional LRU-way selection takes over from LRU-SEQ way-selection. The reason for still using LRU state for *far-neighbors* (a jump beyond SEQ_DST) is to retain in part or whole, the temporal-locality advantages inherent to the LRU policy as well as to avoid scenarios where cache fills are concentrated at a single bank (which would undermine the advantages of the associativity offered by the cache).

Hardware modifications: In order to implement our policy, we need some logic to track the cache *way* and cache *line* accessed in the previous cache access cycle. This data is stored within two additional registers as shown in Fig. 3 (details shown are for a single *way* in the cache for simplicity). During a cache hit, P_{way} is updated. During a cache miss, C_{line} is compared to P_{line} to determine whether LRU/LRU-SEQ *way* have to be applied. Since this logic is activated during a cache miss, any imposed delay is overlapped with memory cycles. As a summary, there is no timing disadvantage and the energy overhead for this additional operation is virtually not noticeable as a percentage of the whole energy consumption of the cache. The next section evaluates our methodology in various scenarios with diverse parameters.

5 Evaluation of LRU-SEQ

This section describes our evaluation platform for LRU-SEQ, the tool flow and the energy model.

5.1 The evaluation platform

We have embedded two different flows in our platform (Fig. 4). The left side depicts a flow that is trace-based i.e. an address trace is generated and afterwards analyzed whereas the right side shows a cycle-based simulation flow.

SysTRACE involves the use of the Shade analyzers [12] and various cache simulators. We developed Csim and LRUSim in order to monitor various activities in caches employing conventional and LRU-SEQ replacement policies. The Dinero cache simulator was simultaneously used for verification purposes. The motivation behind the flow is to isolate the effects of the replacement policy from those due to system-level configurations (cache hierarchies, branch predictor configurations, re-order buffers, etc.) on the run-time of an application.

SysSIM comprises of SimpleScalar tools [13] embedded along with modified cache simulators. The goal behind this flow is to investigate a comprehensive impact of all system parameters (system configuration and replacement policy) on the run-time of an application.

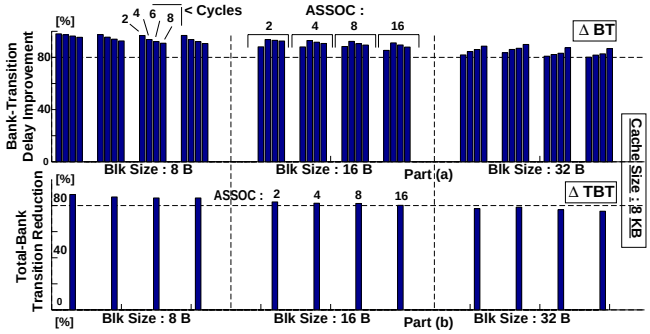

Figure 5: Bank transition reduction by LRU-SEQ for *rasta*

We have used the Mediabench [17] benchmark set to evaluate our LRU-SEQ replacement policy through our platform.

5.2 Energy model

As mentioned earlier, it is the major goal to explore the energy/power advantages of the our policy. Therefore, we employ a cycle-level energy model according to Eqns. 1 and 2. The total energy of the cache, E_{cache} is determined as the sum of the energies consumed by each individual bank in each cycle. The energy consumed by a bank during a certain cycle, E_{bank} is one of three components: E_{active} (read/write access), E_{idle} (buffered access) and E_{sleep} (low-leakage mode). Any banks that transition to/from sleep modes are charged with E_{Tr}.

$$E_{cache} = \sum_{cycles} \sum_{banks} E_{bank} + N_{bank.trans} * E_{Tr} \quad (1)$$

$$E_{bank} = E_{active} \mid E_{sleep} \mid E_{idle} \quad (2)$$

Using this energy model, we make the following assumptions regarding bank usage: (1) All the banks associated with a *way* are activated. Note also that the energy model can be easily modified to take into account any further sub-division. (2) Whenever a bank is accessed, the contents are buffered such that subsequent accesses to the same cache line can be fended by the buffer. However, the bank is not transitioned to a sleep mode (depending on the circuit-level techniques available this may or may not be possible without an adverse impact on cycle time and energy [11] [6] [3]). During those cycles, E_{idle} is charged to the bank. (3) We have charged each bank equal E_{Tr} costs for both types of transitions (cost to wake-up, and to sleep). This may or may not be the case according to the circuit-level technique and the model can be easily modified to take that into account.

6 Results

In this section, we present the experiments conducted using our LRU-SEQ policy. For lack of space, most of the discussions refer to the **SysTRACE** set-up (see Sec. 4 and Sec. 5) unless otherwise stated. However, a summary of the results obtained through **SysSIM** is also given (see Sec. 6.5).

6.1 Transition energy and break-even time

LRU-SEQ has been designed to save energy as explained earlier. This is directly achieved by reduction in *total bank transitions* (TBTs) and indirectly by increasing the *inter-bank transition time* (IBT time). Fig. 5 illustrates this effect. For clarity, we present the data for a cache size of 8 KB, across different associativities (2, 4, 8, 16) and across different block sizes (8 B, 16 B, 32 B). *Part(a)* shows LRU-SEQ's increased IBT times (sleep times) compared to LRU. Eqn. 3 captures the information shown in this plot. $IBT_{\mathbf{n}}$ is the number of IBTs

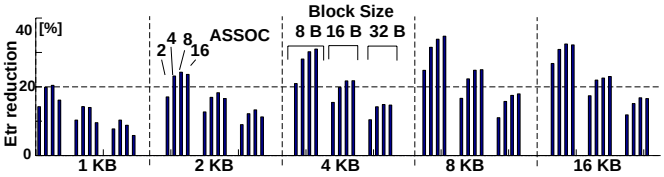

Figure 6: E_{Tr} savings by LRU-SEQ for *rasta*

that occur within **n** cycles. ΔBT_n captures the reduction offered by LRU-SEQ. The plot shows the data for (2, 4, 8, and 10) cycles across the different cache configurations. It can be seen that the minimum improvement is already about 80%. Thus in at least 80% of the IBTs, the cache banks will be able to overcome their energy costs using LRU-SEQ, depending upon the break-even time (2, 4, 8 or 10 cycles) which is, dependent upon the technology.

$$\Delta BT_n = \frac{[(IBT_n)_{LRU} - (IBT_n)_{LRU.SEQ}] * 100}{(IBT_n)_{LRU}} \quad (3)$$

The excellent improvement in BT_n is due to two factors: One is the *absolute* reduction of $(IBT)_n$ (as discussed) and the other is due to the reduction of the *total* number of inter-bank transitions (TBTs). That means: 1) many transition events are completely eliminated and 2) the remaining events are pushed farther apart in time. Both effects are supporting the goal of energy reduction. This is understandable as the LRU-SEQ policy favors to cluster activities in banks.

The improvement in TBTs is plotted in *Part (b)* of Fig. 5. Col. 3 of Tab. 1 summarizes the TBT *reduction* for all the benchmarks. It can be seen that, on an *average* there is a 71-93% reduction in total transitions across the benchmarks. Referring to our energy model (Sec. 5.2), this translates accordingly to savings in E_{Tr}. These savings are plotted in Fig. 6 (where the contribution of E_{Tr} is given by Eqn. 4. ($N_{Tr}, N_{active}, N_{idle}$) are the number of (transition, active and idle) events in the cache. It is seen that up to 35% of the total cache energy can be saved by the LRU-SEQ policy (assuming that E_{Tr} is comparable to E_{active} and E_{idle} which is a fairly reasonable assumptions). Col. 4 of Tab. 1 presents this data for all the benchmarks averaged over all cache configurations. It can be seen that LRU-SEQ has the potential to save considerable amount of cache energy compared to LRU (20-28%).

$$Contribution(E_{Tr}) = \frac{N_{Tr} * 100}{N_{active} + N_{idle} + N_{Tr}} \quad (4)$$

These energy savings would not be meaningful unless we look at the impact on the hit-ratio which effects the total system performance.

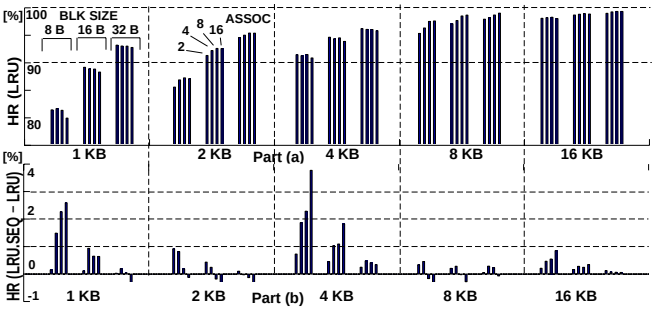

Figure 7: Hit-ratio evaluation for LRU-SEQ for *rasta*

6.2 Effects on the hit ratio

The major performance characteristic for any replacement policy is the cache hit ratio. This is shown in Fig. 7 for different cache configurations (example: *rasta* benchmark). *Part (a)* illustrates the variation in hit-ratio for a conventional LRU policy. Results are plotted for caches from 1 KB to 16 KB, with block sizes from 8 B to 32 B with different associativities from 2-way to 16-way. *Part (b)* shows the difference in the hit-ratios: $\Delta HR_{(\text{LRU.SEQ - LRU})}$. It can be seen that in many cases, LRU-SEQ offers even a slightly better hit-ratio compared to LRU.

Col.2 of Tab. 1 gives comprehensive ΔHR data for all the applications averaged over all cache configurations. On the average, LRU-SEQ performed indeed better by 0.46% for *rasta*. LRU-SEQ had an average variation of 0.46 to -0.04% over all benchmarks. From this data, it can be seen that LRU-SEQ with a $SEQ\_DST = 1$ does not trade-off the energy savings against performance. Indeed, the performance even increases slightly in many cases. Next we study the effect of varying SEQ_DST.

Table 1: Improvement by LRU-SEQ (Cache Size: 4KB - 16KB; Block Size: 8B - 32B; Assoc: 2 - 8; SEQ_DST : 1); (*HR: Hit Ratio, TBT: Total Bank Transitions, E_{Tr}: Transition Energy*)

Bench--marks	ΔHR [%] (avg)	ΔTBT [%] (avg)	$\Delta Rel(E_{Tr})$ [%] (min, max, avg)		
rasta	0.46	81.11	10.42	33.85	21.10
epic-e	-0.00	93.30	13.64	44.79	28.56
epic-u	-0.01	78.54	6.04	38.30	23.63
jpeg-d	-0.00	85.74	10.59	39.40	23.80
jpeg-e	0.01	76.32	11.25	37.00	22.87
mesa-m	-0.04	74.81	11.59	34.52	20.98
mesa-o	0.03	71.22	7.15	31.60	20.15
adpc-e	0.02	82.33	0.52	39.62	25.27

6.3 Effects when varying SEQ_DST

Previously, we have presented SEQ_DST as a possible parameter in our policy. The idea behind this parameter is to group farther and farther branch destinations under sequential fills (to the same *way*). We want to investigate the effect on the hit-ratio and the advantages in terms of increase in the reduction of bank transitions (ΔBT).

Fig. 8 presents $\Delta HR_{(\text{LRU.SEQ-LRU})}$ as SEQ_DST is varied from 1 to 4 (data was collected upto a SEQ_DST of 64, but are not presented for space constraints). Plots on the left show ΔHR. The top plot corresponds to a SEQ_DST of 1, the next to a SEQ_DST of 2 and so on. Y-axis denotes ΔHR while x-axis denotes the test cases (all cache configurations: Size 4KB - 16 KB; Block Size 8B - 16 B; Assoc 2 - 8; for all the benchmarks). The range of ΔHR is given near the lower left corner as R(a,b). Plots on the right depict the % decrease in bank-transitions as SEQ_DST is varied.

It can be seen that there is very little increase in ΔBT as SEQ_DST increases. However the effect on ΔHR is more apparent. Let us we define a *"good"* value for SEQ_DST as one for which: $-1 \leq \Delta HR \leq +1$. Reviewing the values of the standard deviation as SEQ_DST is increased, we found that a $SEQ\_DST \geq 2$ can be detrimental (for 95% test coverage). If we look further at the range of variation (R) in ΔHR, LRU-SEQ has the best performance for SEQ_DST of 1 (-0.5 vs. -3).

In the following, we combine the *sequential-fill* methodology with different policies and observe the effects on hit-ratio and bank-transition reduction.

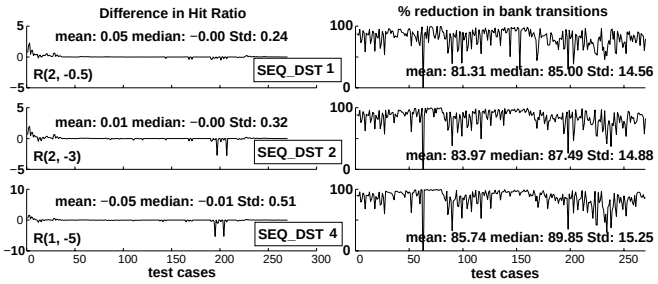

Figure 8: Evaluation of SEQ_DST for LRU-SEQ

6.4 PLRU-SEQ and Random-SEQ

Given the implementation complexity of LRU, pseudo-LRU (PLRU) is often adopted (x86 through Pentium [7]). We applied the principles behind LRU-SEQ to this policy and analyzed plots similar to those in Fig. 8. Our analysis showed that: a SEQ_DST of 1 provides the best performance with a bank transition reduction of 82% and a maximum variation in hit-ratio of (2, -0.5) (PLRU.SEQ - PLRU) when compared to regular PLRU implementations. Analysis for applying the methodology to the random replacement policy (Sparc family [15]) showed that: a SEQ_DST of 1 has a 82% reduction in bank transitions and a maximum variation in hit-ratio of (0, -0.5) (Rnd.SEQ - Rnd).

6.5 Effects of system configuration using SysSIM

In the previous sections an address trace generator (Shade) was used to evaluate the new replacement policy. In this section we present the results obtained on the SimpleScalar framework [13]. The goal behind these tests is to obtain an idea of the relationship between the effect on hit-ratio and the corresponding effect on the run-time due to various system-level parameters.

Results for various instruction-cache configurations (Size: 4KB-16KB, Line Size: 8B-32B, Assoc: 2-8, SEQ_DST: 1-64), and system-level configurations of the SimpleScalar machine (Data cache: default, 4 KB & 1 MB; in-order/out-of-order execution cores; minimal resource, i.e., single IU, FPU and default configurations) are presented in Fig. 9. Three metrics are presented: 1) Gain in Hit-ratio (Col.1), 2) % Gain in run-time (Col. 2) and 3) Bank transition reduction (last column). R(a,b) values denote the range of the data; S(a,b,c) denote the triplet (mean, median, standard deviation) and S(a,b) denotes the (mean, median). It can be seen that a SEQ_DST of 1 or 2 is preferable. The second column shows that the effect on the run-time on the average is 0.36%.

The system-level study presented in this section thus confirms our observations in the previous sections that LRU-SEQ policy with a SEQ-DST of 1 or 2 is excellent for reducing transition energy effects with a minimal impact on the run-time.

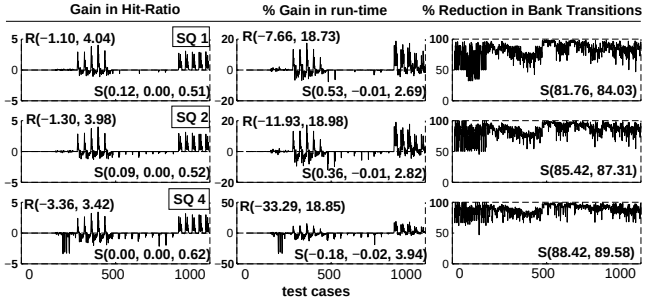

Figure 9: $LRU - SEQ$ for the Simplescalar architecture

7 Conclusions

We presented a novel replacement policy LRU-SEQ for instruction caches. The policy exploits the fact that reducing inter-bank transitions and increasing the sleep time for the banks between transitions can save a considerable amount of cache energy. This holds especially for future silicon technologies where static power consumption is expected to dominate through leakage power consumption. We obtain cache energy savings with an average of 23% across all benchmarks. The policy has been carefully studied within different system set-ups (SysTrace and SysSIM) as well as for a large set of parameters like SEQ_DIST and cache parameters (like cache size, associativity, block size etc.). Beyond comparing LRU-SEQ to LRU, we have also studied pseudo random and random replacement policies as they are used by some processor architectures. Despite the energy savings of 23% in average, our policy does not incur any noticeable performance penalty (though the cache access patterns change) as shown and the additional hardware required within the cache is minimum. It can be concluded that the energy savings come almost for free. LRU-SEQ is currently only validated for instruction caches. Future work will include the validation of data caches, too.

References

[1] A. Agarwal, H. Li and K. Roy, "DRG-cache: a data retention gated-ground cache for low power", *DAC, 2002*, pp. 473-478.

[2] B. Dole et al, "Transistor elements for 30nm physical gate lengths and beyond", *Intel Technology Journal*, Vol. 6, Issue 2, May 2002.

[3] S. Dropsho et al,"Managing static leakage energy in microprocessor functional units", *MICRO 2002*, Page(s): 321 -332.

[4] K. Flautner et al,"Drowsy caches: simple techniques for reducing leakage power", *ISCA 2002* Page(s): 148 -157.

[5] H. Hanson et al., "Static energy reduction techniques for microprocessor caches", *Proc. ICCD 2001*.

[6] S. Heo, K. Barr, M. Hampton and K. Asanovic, "Dynamic fine-grain leakage reduction using leakage-biased bitlines", *ISCA-29*, May 2002.

[7] Embedded Intel486 Processor Family Developer's Manual.

[8] T. Ishihara and K. Asada,"An Architectural Level Energy Reduction Technique For Deep-Submicron Cache Memories", *ASP-DAC/VLSI Design 2002*, p. 282.

[9] "International Technology Roadmap for Semiconductors 2002 Update", http://public.itrs.net/.

[10] S. Kaxiras et al,"Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power", *ISCA 2001*, p.0240.

[11] C. H. Kim and K. Roy, "Dynamic Vt SRAM: a leakage tolerant cache memory for low voltage microprocessors", *ISLPED 2002*, pp.251-254.

[12] R. F. Cmelik and D. Keppel. "Shade: A fast instruction-set simulator for execution profiling.", *Technical Report 93-06-06*, Dept. of CSE, University of Washington, June 1993.

[13] www.simplescalar.com

[14] K. Soontae et al,"Predictive precharging for bitline leakage energy reduction", *IEEE ASIC/SOC Conference 2002*, pp.36-40.

[15] UltraSparc III Cu User Manual.

[16] L. Li et al,"Leakage Energy Management in Cache Hierarchies", *PACT 2002*.

[17] Mediabench, http://www.cs.ucla.edu/ leec/mediabench/.

[18] K. Nii et. al,"A low power SRAM using auto-backgate-controlled MT-CMOS", *ISLPED 1998*, pp.293 -298.

[19] M. Powell et al,"Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories", *ISLPED 2000*, pp. 90-95.

[20] G. Sery, S. Borkar and V. De, "Life is CMOS: why chase the life after?", *DAC 2002*, pp. 78-83.

[21] K. N. Sung, K. Flautner, D. Blaauw and T. Mudge,"Drowsy instruction caches - leakage power reduction using dynamic voltage scaling and cache sub-bank prediction", *MICRO 2002*, Page(s): 219 -230.

[22] S.-H.Yang et al,"An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches", *HPCA 2001*, pp. 147-158.

[23] H. Zhou et al,"Adaptive Mode Control: A Static-Power-Efficient Cache Design", *PACT 2001*, pp.61.