Prefetching-Aware Cache Line Turnoff for Saving Leakage Energy *

Ismail Kadayif
Dept. of Computer Engineering
Canakkale Onsekiz Mart University
Canakkale 17100, TR
kadayif@comu.edu.tr

Mahmut Kandemir
Dept. of Computer Sci. & Eng.
Pennsylvania State University
University Park, PA 16802, USA
kandemir@cse.psu.edu

Feihui Li
Dept. of Computer Sci. & Eng.
Pennsylvania State University
University Park, PA 16802, USA
feli@cse.psu.edu

Abstract— While numerous prior studies focused on performance and energy optimizations for caches, their interactions have received much less attention. This paper studies this interaction and demonstrates how performance and energy optimizations can affect each other. More importantly, we propose three optimization schemes that turn off cache lines in a prefetching-sensitive manner. These schemes treat prefetched cache lines differently from the lines brought to the cache in a normal way (i.e., through a load operation) in turning off the cache lines. Our experiments with applications from the SPEC2000 suite indicate that the proposed approaches save significant leakage energy with very small degradation on performance.

I. Introduction

Caches are critical components from both performance and energy viewpoints, and are being increasingly employed in mobile and embedded environments. From the performance angle, they sit on the critical path of execution, and their hit/miss characteristics usually determine the overall performance of an application. From the energy angle, they are responsible from up to 42% of overall on-chip energy consumption [11]. Therefore, optimizing their performance and energy characteristics is very important and is expected to be even more so in the future.

Due to importance of cache memories, they have been explicit target of many previous optimizations from both performance [12, 6] and power angles [3, 8, 7]. While these techniques have been evaluated thoroughly in isolation (from both performance and power perspectives in many cases), their interaction with each other, when they co-exist together in the same system, took relatively much less attention in the past. For example, it is not clear how data/instruction prefetching would interact with cache line turn-off, used for leakage reduction. Studying this interaction is critical since many embedded environments today demand both high-performance and low-power. Without capturing the power impact of performance optimizations and performance impact of power optimizations, one will not be able to perform the necessary tradeoffs in designing and optimizing an embedded system.

This paper has two major goals. First, focusing on a specific performance optimization (prefetching) and a specific power optimization (cache line turn-off), it presents energy and performance results, emphasizing on how these two optimizations interact with each other from both performance and power angles. Second, it proposes three novel cache line prefetching techniques that take prefetching employed by the hardware into account. The proposed approaches exploit the knowledge on prefetched lines by employing a different decay interval (time frame after which the cache line is turned off) for prefetched lines. In other words, in the proposed strategies, the prefetched lines and the normal lines (i.e., the lines brought into the cache through execution of load operations) use different thresholds. This in turn minimizes the potential negative impact of prefetching on energy consumption. In more detail, prefetching brings data/instructions from main memory to cache before they are actually needed. But, more importantly, when a cache line is placed into a low-leakage mode (using a cache line turnoff strategy), prefetching data/instructions into it forces it to be transitioned to the normal operation mode (active mode), thereby affecting leakage behavior as well. Our goal in this paper is to study these interactions using a set of benchmarks in a two-level cache hierarchy, and quantify the potential benefits of exploiting this interaction using different prefetching-aware cache line turn-off schemes. We present experimental data – using a simulation environment and the SPEC2000 benchmarks – that emphasize the importance of capturing the interactions between performance and energy optimizations, and show how the proposed optimization schemes improve power-performance tradeoff when prefetching and cache line turnoff co-exist in the same system. The experiments also indicate that the performance overheads caused by the proposed schemes are very low.

This paper is structured as follows. The next section summarizes the particular prefetching and cache line turnoff schemes studied in this paper, and explains their interaction qualitatively. Section III gives our experimental setup, and Section IV presents results from our implementation. In Section V, we propose three optimization strategies which treat the prefetched cache lines differently than those brought into cache in normal way to save further leakage energy in caches. We conclude the paper in Section VI with a summary of our major observations.

II. Preliminaries: Prefetching and Cache Line Turnoff

There are a number of prefetching techniques in literature. For example Lai, et al. [10] use trace of memory references to predict when a block becomes dead, and exploits the ad-

*This work is supported in part by NSF Career Award 0093082 and a grant from GSRC.

0-7803-9451-8/06/$20.00 ©2006 IEEE.

dress correlation to predict which subsequent block to prefetch. In this study we used a hardware-based technique called the *tagged prefetch* [12], which is implemented in several commercial architectures including HP PA7200 [5]. It is based on one block lookahead, which initiates a prefetch for block b+1 when block b is accessed under two different scenarios. First, if there is a miss when b is accessed. Second, if b is brought into cache via prefetching and it is accessed for the first time. We used this approach in an L1-L2 cache hierarchy for both data and instruction prefetching.

While prefetching targets at improving performance, the leakage control mechanisms try to reduce the energy consumption. An important requirement to reduce leakage energy using either a state-destroying (cache line turnoff) or a state-preserving leakage control mechanism is the ability to identify unused resources (cache blocks).[1] Kaxiras et al. [8] present a state-destroying leakage energy reduction technique for cache memories. This technique, called *cache decay,* is based on the idea that a cache block (line) that is not used for a sufficiently long period of time can be considered dead. More specifically, with each cache block, they associate a small 4-state FSM (finite state machine). The FSM steps through these states as long as the cache block is not being accessed. When the last state is reached, the cache block is turned off. Li et al. [9] propose several architectural techniques that exploit data duplication across the different levels of cache hierarchy. They employ both state-preserving (data-retaining) and state-destroying leakage control mechanisms for the L2 sub-blocks when their data also exist in L1. Among their strategies, S-SP-Lazy (Speculative, State-Preserving, and Lazy) generates the best leakage energy savings. In this strategy, when a data is brought from L2 to L1, the corresponding L2 sub-block is put in a state preserving leakage control mode. In this study, the cache decay technique and the S-SP-Lazy technique are integrated to save leakage energy in the L1-L2 cache hierarchy. Specifically, we use cache decay for L1, but both cache decay and S-SP-Lazy for L2. The L2 cache is energy-managed at the sub-block granularity and the sub-block is put into state-preserving leakage mode immediately after it is brought into the L1 cache. Further, if the sub-block is not accessed for a sufficiently long period of time, it is transitioned to the state-destroying mode. If all the sub-blocks of an L2 block are in the state-destroying mode, the block is invalidated and subsequently becomes a candidate for LRU replacement.

From both performance and leakage energy viewpoints, it is important to ensure that the prefetched cache lines are used before being replaced. Therefore, we can divide prefetches in two groups, namely, *useful* and *non-useful*. If a prefetched cache line is discarded from the cache without being accessed, this means that it is prefetched unnecessarily. In this paper, we call this a non-useful prefetching, as opposed to useful prefetches whose data are used at least once before being displaced from the cache. Non-useful prefetches can cause performance degradation due to keeping the bus busy and potentially introducing extra cache misses. Further, they increase the both dynamic energy consumption (because of unnecessary cache access) and leakage energy consumption (if the prefetched block is brought into a cache line which is in the leakage-saving mode).

A cache line can be in one of the three power modes (states): active (AC) mode (consuming full leakage power), state preserving (SP) low-power mode (we assume that, in this mode, the cache line consumes 10% of the leakage energy of active mode as in [9]), and state destroying (SD) power mode (no energy consumption at all).

TABLE I
BASE CONFIGURATION.

Processor Core	
Functional Units	4 integer and 4 FP ALUs 1 integer multiplier/divider 1 FP multiplier/divider
LSQ Size	64 Instructions
LRU Size	64 Instructions
Fetch/Decode/Issue/Commit Width	4 instructions/cycle
Fetch Queue Size	4 Instructions
Cache and Memory Hierarchy	
L1 Instruction/Data Cache	32KB, 32 byte blocks 2-way, 1 cycle latency
L2 Cache	1MB unified, 128 byte blocks 2-way, 10 cycle latency
Data/Instruction TLB	128 entries, full-associative, 30 cycle miss latency
Memory	100 cycle latency
Energy Management	
Technology	0.07 micron
Supply Voltage	1.0V
Dynamic Energy per L1 Access	0.186nJ
Leakage Energy per L1 Block/ L2 Sub-block per Active Cycle	0.182pJ
Leakage Energy per L2 Sub-block per Standby Cycle (state preserving)	0.018pJ
Leakage Energy per L1 block/ L2 Sub-block per Standby Cycle (state-destroying)	0pJ

III. EXPERIMENTAL SETUP

We used SimpleScalar 3.0 [4] to implement our prefetching and leakage-saving optimization strategies, and study their interactions. SimpleScalar is a tool-set to simulate application programs on a range of processors and systems using fast execution-driven simulation. In this work, we used the sim-outorder component. Table I lists the simulation parameters used for our base configuration. Note that, embedded systems are increasingly using multiple issue processors. We used the 70nm technology and energy models from CACTI 3.2 [2] to get the dynamic energies of accessing L1 and L2 caches. We define the *leakage factor* represented with parameter k as follows: the ratio between the leakage energy per cycle of the entire L1 cache and the dynamic energy consumed per access. In our study we assume k=1. (Larger k values reflect the future designs, and although we only provide results for k=1 due to lack of space, for larger k values our schemes accomplish more savings in overall energy.) Further, we assume that the leakage energy of an L2 sub-block is equal to that of the L1 block. We used five randomly-selected benchmarks from the SPEC2000 suit [1] in our experiments. Since it takes a long time to simulate any benchmark from the SPEC2000 suit when it runs to completion, we fast forwarded the first 300 million instructions, and then simulated the next 200 million instructions.

[1] When there is no confusion, we use the term "cache line turnoff" to cover for both state-destroying and state-preserving mechanisms.

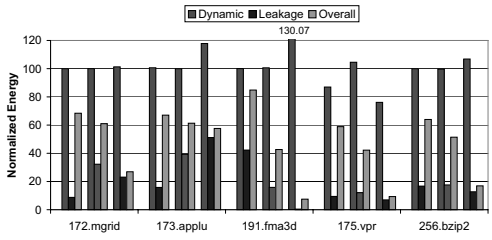

Fig. 1. The normalized energy consumption of optimized codes when k=1.

IV. Base Results

In the remainder of this paper, we refer to a benchmark optimized by prefetching and leakage control mechanisms as *optimized*. In this section we give energy and performance results for optimized codes.

A. Energy Savings

The normalized energy consumption of the optimized codes with respect to energy consumption of the original codes are given in Figure 1. In dynamic energy calculations, we conservatively assumed that each prefetch attempt (even for the blocks already in the cache) consumes some dynamic energy, amount of which is equal to the dynamic energy consumed per access. In the figure, for each benchmark, the first and the second group of bars are for the L1 instruction cache, and the L1 data cache, respectively, whereas the last group of bars is for the L2 cache. The bars in each group, from left to right, show the dynamic energy, leakage energy, and overall energy of the corresponding cache. As can be seen from this figure, for all benchmarks, the dynamic energy overhead introduced by the prefetching mechanism and the leakage control mechanism together is so small that can be omitted for L1 caches. This is not the case for the L2 cache though. Our optimizations increase the dynamic energy of the L2 cache 17.83%, and 30.07% for 173.applu, and 191.fma3d, respectively. Interestingly, the 175.vpr benchmark benefits, as far as the dynamic energy consumption in the L1 instruction and L2 caches are concerned, from the optimizations. The reason of this is that the prefetching in this benchmark reduces the cache misses for the caches in question dramatically, thereby lowering the number of total cache accesses. When we look at the leakage results, we see that, for each benchmark there is a considerable saving in all caches. An important observation from Figure 1 is that the leakage energy consumption dominates the dynamic energy consumption in the L2 cache even after cache line turnoff. This can be attributed to the fact that leakage energy consumption was extremely large for this cache in the original codes.

B. Execution Cycles

While prefetching potentially increases the performance by bringing cache blocks early in caches, a leakage control mechanism usually degrades performance since there is an overhead to be incurred when a cache line is transitioned from the state preserving/state destroying leakage mode to the full power (i.e., active) mode before being accessed. In our experiments, we assumed that it takes one extra cycle to bring up a cache line into the active mode. Table II shows the execution cycles for the optimized codes, normalized with respect to the corresponding execution cycles of the original codes. On an average, we have 21.62% performance improvement for the optimized codes.

TABLE II
Normalized execution cycles.

172.mgrid	173.applu	191.fma3d	175.vpr	256.bzip2
65.07%	76.08%	118.22%	54.67%	67.83%

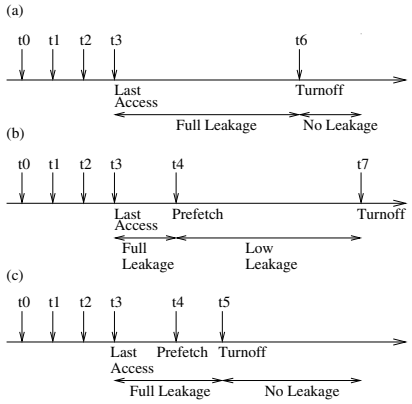

Fig. 2. (a) No prefetching. Full leakage power is consumed during the period [t3,t6]. (b) S-SP strategy. There is a prefetch at time t4, and full leakage power is consumed only during the period [t3,t4]. (c) L-SD strategy. Prefetched line is transitioned into SD leakage control mode at t5 (i.e., after a small decay period after being prefetched), so consuming full leakage power during the period [t3,t5]. However, there is no leakage consumption after t5.

V. Customizing Cache Line Turnoff for Prefetched Lines

Until this point in our experimental evaluation, all the cache lines are treated exactly the same way, whether they have been brought into the cache through prefetching or through a normal load operation. In particular, a prefetched cache line is put in SD power mode if it is not touched during the decay interval. In this section, we discuss three different optimization strategies that treat the prefetched cache lines *differently* from the normal (non-prefetched) cache lines. The first strategy places the prefetched cache lines into the SP mode immediately after the prefetching is performed. In the rest of this paper, we refer to this scheme as the S-SP approach (which stands for Speculative and State Preserving). The second scheme, on the other hand, employs a new decay period for the prefetched cache lines to put them into the SD power mode; we call it L-SD (Lazy and State Destroying). As we will explain shortly, the decay period for the prefetched cache lines in L-SD is much shorter than that for the normal cache lines. The last strategy treats a prefetched cache line based on the characteristics of the previous prefetch into the same line. We refer to the third scheme as the P-H (Predictive and Hybrid). It is predictive in

a sense that it tries to figure out whether the prefetching into the line in question would be useful or not. Furthermore, this strategy is hybrid as far as deciding the power status of the prefetched cache line (after prefetching) is concerned, since the power status of the cache line (after prefetching) can be either AC or SP (i.e., both are possible).

A. S-SP Scheme

This scheme places the prefetched cache line in the SP leakage control mode speculatively, immediately after the prefetching is complete. In this way, if the cache line is prefetched unnecessarily (i.e., the cache line would be discarded from the cache without being accessed at all), the line in question will be in SP leakage control mode until it is being replaced (or during the decay period), instead of being in the AC leakage control mode consuming full leakage power. Also, with this strategy, some non-useful prefetches are not bad at all in terms of leakage energy; on the contrary, they may contribute to the leakage energy savings, which can be explained as follows. Figure 2(a) shows the lifetime of a typical cache line. At time t0, a new data is brought into the line in question, and at time t3 the last access to the data occurs. Suppose that the cache line has not been accessed during the decay period, it is transitioned to the SD leakage control mode at time t6, so consuming no leakage power after t6. Figure 2(b) illustrates the cache line in question when the S-SP strategy is used. Assume that the data is brought into the cache line at t4 by prefetching, and at the same time the cache line is transitioned to the SP mode. Consequently, the cache line will consume full leakage power only during the period [t3,t4], instead of [t3,t6], and consume low leakage power during the period [t4,t7], resulting in leakage energy saving. If the original data in that cache line is not referenced later by the program, this prefetch (although it is not useful in itself) can translate to overall power savings.

The drawback of this strategy is that it puts not only the non-useful prefetched cache lines in SP leakage control mode but also the useful prefetched cache lines. So, when the useful prefetched cache line is accessed, the line needs one extra cycle to be waken up, causing performance degradation.

B. L-SD Scheme

In this scheme, we define a new decay period for the cache lines brought into the cache via prefetching. Because of the principle of locality, it is fair to assume that the useful prefetched cache lines, in general, would be accessed within a short period of time after they are brought into the cache. Therefore, it makes sense to put the prefetched cache line into SD leakage control mode if it is not accessed within this short decay period (anticipating that it will not be accessed at all). To explain this strategy better, let us consider the Figure 2(c). The cache line is brought into the cache via prefetching at time t4. If the cache line is not accessed within the period [t4,t5] (note that t5-t4 is equal to new cache decay period), it is placed in the SD leakage control mode at time t5. Consequently, the non-usefully prefetched cache line will consume full leakage power only from t4 to t5 instead of from t4 to t7 (t7-t4 is equal to the decay period of the normal cache lines). This results in reducing overhead of non-useful prefetched cache lines.

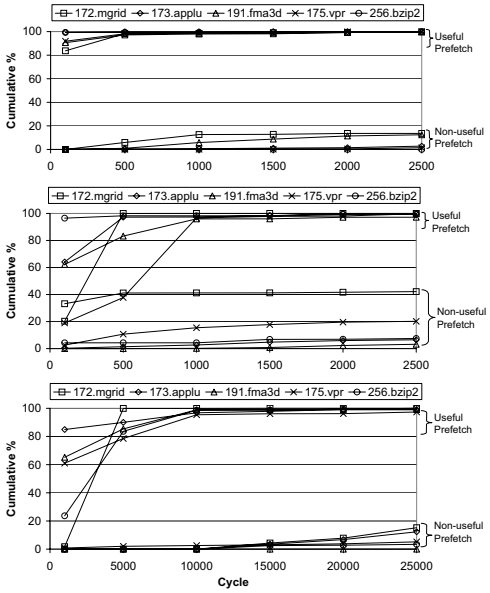

Fig. 3. Cumulative distribution of access interval of prefetched cache lines. The top and middle graphs correspond to the L1 instruction and data caches, respectively, and the bottom graph is for the L2 cache. The upper lines in each graph show the access intervals for the useful prefetched cache lines, while the lower lines indicate the access intervals for the non-useful prefetched lines.

B.1 Finding the Decay Period for Prefetched Lines

In L-SD scheme it is very important to choose a suitable decay period for the prefetched cache lines. If we choose a very short decay period, a usefully-prefetched line may be turned off too early before it is accessed, resulting in performance and energy degradation. On the other hand, if the decay period is chosen too long, a non-useful prefetched cache line will be in the AC leakage control mode unnecessarily long, increasing the leakage energy overhead. To decide a suitable cache decay period for prefetched cache lines, we conducted a set of experiments, which we explain below.

The cumulative distribution of the access intervals for the prefetched L1 instruction, L1 data, and L2 lines are shown in Figure 3. In all three graphs, the upper lines capture the access intervals for the useful prefetched cache lines, whereas the lower lines depict the access intervals for the non-useful prefetched cache lines. From this figure, we can make the following observations. First, in general, the useful prefetched lines in the L1-L2 hierarchy are touched within very short interval after being prefetched. For example, on average, 98.84% (97.65%) of the useful prefetched L1 instruction (L1 data) cache lines are touched within a 1K cycles interval after they are brought in the corresponding cache via prefetching. Furthermore, within an interval of 10K cycles after being brought into the L2 cache, 97.95% of the useful prefetched cache lines are touched. These results are compatible with the rule of spatial locality, that is, when an instruction/data is accessed it is very likely that the neighboring instruction/data will be accessed within a short period of time. Second, the non-useful prefetched lines spend considerable amount of time before being discarded from the cache.

Based on these two observations, one can reduce the decay

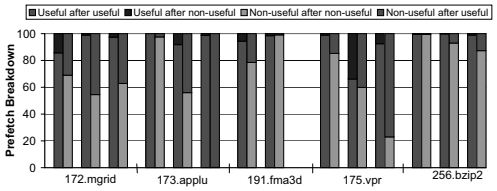

Fig. 4. Breakdown of the useful prefetches into useful after useful and useful after non-useful cases, and breakdown of non-useful prefetches into non-useful after non-useful and non-useful after useful cases.

interval of prefetched cache lines to cut down the leakage energy overhead of the non-useful prefetched cache lines, with negligible degradation in performance, by allowing them to spend less time in the AC leakage control mode. For the rest of our experimental analysis, we selected the decay period of the prefetched cache lines as 1K cycles for the L1 instruction and data caches, and 10K cycles for the L2 cache.

C. P-H Scheme

Before going on to the details of this strategy, let us explain some experimental results consolidating the idea behind it. First, we tried to categorize the useful/non-useful prefetches based upon the previous prefetch into the corresponding cache line. If the prefetch is useful after a useful prefetch into the same line, we call it *useful after useful*. Similar definitions can be made for *useful after non-useful*, *non-useful after non-useful*, and *non-useful after useful* prefetches.

Figure 4 shows the breakdown of the useful/non-useful prefetches into categories. For each benchmark, there are three groups of bars; the first and second correspond to the L1 instruction and data caches, respectively, while the third one corresponds to the L2 unified cache. As can be seen from the figure, for all caches, if the prefetch is useful, the next prefetch into the same line will more likely be useful. Furthermore, except for L2 cache with the 173.applu and 175.vpr benchmarks, if the data is brought into a line via prefetching and replaced before being accessed, the next data brought into the same line via prefetching will likely be discarded before being accessed. Second, we tried to see to what extent we can predict the time when the prefetched data will be accessed by just looking at the access interval of the previous data brought into the same line via prefetching. What we mean by "access interval" is the difference between the time when the data is brought into cache via prefetching and the time when the data is actually accessed. Figure 5 shows the cumulative distribution of differences between the access intervals of two consecutive useful prefetches into the same lines. We can observe that, for all caches in our architecture, the access interval of a prefetched cache line can be predicted quite accurately within a range based on the access interval of the previous prefetch into the same line. For each prefetched line, the P-H strategy checks whether the previous prefetch into the same line was useful. If the previous prefetch was non-useful, the line is placed in SP leakage mode (as in the S-SP scheme) in case the current prefetch would be useful. Otherwise, the prefetch is assumed to be useful and its leakage mode is determined by considering the behavior of the previous useful prefetch into the same line. If the access

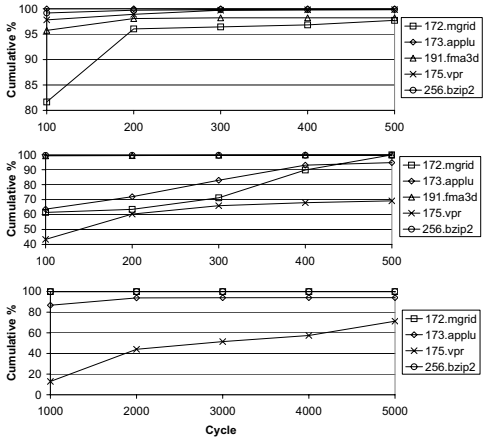

Fig. 5. Cumulative distribution of the differences between the access intervals of the two successive useful prefetches into the same cache line. The top and middle graphs are for the L1 instruction and data caches, and the lower graph is for the L2 cache.

interval of the previous prefetch is smaller than a $threshold$ value, the prefetched data is assumed to be accessed within very short interval after the prefetching is complete; so, it is placed in the AC leakage mode so that it will be ready when it is needed (as the threshold value, we used 200 cycles for both the L1 instruction and data caches, and 2000 cycles for the L2 cache). If this is not the case, the prefetched line is placed in SP leakage mode for a period, which is smaller than the access interval of the previous prefetched data into the same line by the threshold. Tuning the value of this threshold parameter is critical since it affects both power and performance. After the interval, if the line is still in the SP mode (i.e., the line is not accessed within the interval), it is placed into the AC leakage mode so that it would be ready when it is required. If the prefetched line is not accessed during the decay period used for the prefetched lines after it is prefetched (regardless of the outcome of the previous prefetch), it is placed into SD leakage mode as in the L-SD scheme to save further leakage energy. In our experiments, as the decay period for the prefetched cache lines, we adopted the same values we used in the L-SD strategy, i.e., 1K/10K cycles for L1/L2 caches.

D. Leakage Energy Savings

In this section, we first give the results regarding the leakage energy overhead caused by non-useful prefetches and the maximum leakage savings when an *oracle* predictor is used, and then give the leakage energy savings achieved through the L-SD, S-SP, and P-H schemes described above. For maximum leakage savings, we assume that we have an oracle that can predict precisely whether a prefetch is useful or not. The percentage of leakage overheads introduced by non-useful prefetches and the leakage savings achieved via the oracle in the L1-L2 cache hierarchy for the optimized codes are shown in Figure 6. For each benchmark, the first, second, and third groups of bars correspond to the L1 instruction cache, L1 data cache, and L2 cache, respectively. In each group, the first bar illustrates the overhead incurred by non-useful prefetches, the second bar depicts the leakage saving obtained using the oracle. One can see that, for any benchmark, in general, the

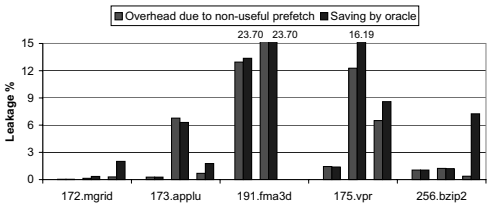

Fig. 6. Leakage energy overheads due to non-useful prefetches and the leakage savings by the oracle.

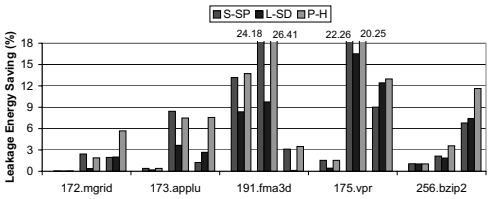

Fig. 7. Leakage energy savings with the L-SD, S-SP, and P-H schemes.

leakage energy saving for any component is larger than the overhead introduced by the non-useful prefetches. Moreover, the 191.fma3d benchmark has 23.70% leakage overhead and saving in the L1 data cache, the largest percentage of leakage overhead and saving in any component in the L1-L2 cache hierarchy among all our benchmark codes. On the other hand, the same benchmark does not incur any overhead in the L2 cache since all its prefetches into L2 are useful.

Figure 7 shows the leakage energy savings in the L1-L2 cache hierarchy when the L-SD, S-SP, and P-H schemes are applied. All the savings are given as percentages with respect to the leakage energy consumption of the optimized codes (i.e., the codes optimized by prefetching and leakage control mechanisms). For each benchmark, there are three groups of bars, where the first and second groups are for the leakage savings in the L1 instruction and L1 data caches, and the third group is for the leakage savings in the unified L2 cache. In each group, the first bar shows the leakage energy saving when the L-SD scheme is used, whereas the second and third bars indicate the leakage energy savings with the S-SP and P-H schemes, respectively. From this figure, we can make the following observations. First, the S-SP scheme outperforms the L-SD scheme in both the L1 instruction and L1 data cache leakage energy savings. This is due to fact that S-SP scheme puts the prefetched cache lines in SP leakage control mode immediately after the prefetch operation is complete, and these L1 lines spend at most 10K cycles (not a long duration at all) before being turned off. On the other hand, the L-SD scheme waits for 1K cycles to put the prefetched lines in the L1 caches in the SD mode, resulting in more leakage energy consumption. Second, as opposed to the case with the L1 caches, in general, the L-SD scheme does a better job than the S-SP scheme in saving the L2 leakage energy. This can be attributed to the fact that the L-SD scheme transitions the prefetched cache line to the SD leakage control mode in a relatively short period of time (10K cycles) after the prefetch is complete, thereby resulting in considerable leakage saving. On the other hand, with the S-SP scheme, a prefetched cache line may spend quite a long time (about 1M cycle) in the SP leakage control mode if it is not accessed. This makes the S-SP scheme less efficient than the L-SD scheme as far as the savings in L2 are concerned. As shown in the figure, the behavior of the L-SD scheme in L2 energy saving is better than that of the S-SP scheme for all the benchmarks except 191.fma3d. The different behavior observed with this benchmark is due to the fact that there is no non-useful prefetches in L2 for this benchmark; so, the L-SD scheme can not take advantage of them. Our next observation is that, the P-H scheme outperforms the L-SD scheme in leakage savings for all caches in the hierarchy. Fourth, the performance of the L-SD scheme depends on the number of the non-useful prefetches. If the number of non-useful prefetches are low as in the 191.fma3d benchmark with the L2 cache, its performance will be very poor. However, this is not the case for the P-H scheme since it may place the usefully-prefetched cache lines into the SP leakage power mode. Due to space limitation we are not giving the performance details here. The average performance overhead is around 1% for each scheme, and their performance ranking is S-SP, P-H, and L-SD.

VI. Conclusions

This paper makes two important contributions. First, it presents a detailed quantification of the power-performance interactions between prefetching and cache line turnoff. Second, based on this quantification, it proposes and evaluates three different cache line turn off schemes that treat prefetched lines differently from the lines brought into the cache through a normal load operation.

References

[1] Spec cpu2000 benchmark. http://www.spec.org/.

[2] http://research.compaq.com/wrl/people/jouppi/CACTI.html

[3] B. Batson and T. N. Vijaykumar. Reactive associative caches. In Proc. *International Conference on Parallel Architectures and Compilation Techniques,* 2001.

[4] D.C Burger and T. M. Austin. The SimpleScalar toolset, version 2.0, Tech. Rep. 1342, *Dept. of Computer Science,* UW, June 1997.

[5] K. K. Chan. Design of the HP PA 7200 cpu. *Hewlett-packard J.,* vol. 47, no. 1, pp. 25-33.

[6] R. Cooksey, S. Jourdan, and D. Grunwald. A stateless contend-directed data prefetching mechanism. In Proc. *International Conference on Architectural Support for Programming Languages and Operating Systems,* October 2002.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In Proc. *International Symposium on Computer Architecture,* 2002.

[8] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting general behavior to reduce cache leakage power. In Proc. *the 28th Annual International Symposium on Computer Architecture,* 2001.

[9] L. Li, I. Kadayif, Y. F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam. Leakage energy management in cache hierarchies. In Proc. *International Conference on Parallel Architectures and Compilation Techniques,* September 2002.

[10] A. C. Lai, C. Fide, and B. Falsafi. Dead-block prediction and dead-block correlating prefetchers. In Proc. *International Symposium on Computer Architecture,* 2001.

[11] J. Montenaro et al. 160 mHz 32b 0.5w CMOS RISC Microprocessor. In Proc. *International Solid State Circuits Conference,* 1996.

[12] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Computing Surveys,* vol. 32, no. 2, June 2000.