IATAC: A Smart Predictor to Turn-off L2 Cache Lines

JAUME ABELLA

Universitat Politècnica de Catalunya-Barcelona

ANTONIO GONZÁLEZ and XAVIER VERA

Intel Barcelona Research Center, Intel Labs-UPC

and

MICHAEL F. P. O'BOYLE

University of Edinburgh

As technology evolves, power dissipation increases and cooling systems become more complex and expensive. There are two main sources of power dissipation in a processor: dynamic power and leakage. Dynamic power has been the most significant factor, but leakage will become increasingly significant in future. It is predicted that leakage will shortly be the most significant cost as it grows at about a $5\times$ rate per generation. Thus, reducing leakage is essential for future processor design. Since large caches occupy most of the area, they are one of the leakiest structures in the chip and hence, a main source of energy consumption for future processors.

This paper introduces *IATAC* (inter-access time per access count), a new hardware technique to reduce cache leakage for L2 caches. *IATAC* dynamically adapts the cache size to the program requirements turning off cache lines whose content is not likely to be reused. Our evaluation shows that this approach outperforms all previous state-of-the-art techniques. *IATAC* turns off 65% of the cache lines across different L2 cache configurations with a very small performance degradation of around 2%.

Jaume Abella and Antonio González were supported by the Ministry of Education and Science under grants AP2002-3677, TIN2004-07739-C02-01, and TIN2004-03072, the CICYT project TIC2001-0995-C02-01, Feder funds, and Intel Corporation.

Part of this work was done while Xavier Vera was a Ph.D. student at the Department of Computer Science and Electronics, Mälardalens Högskola.

Antonio González is also with Computer Architecture Department, Universitat Politècnica de Catalunya-Barcelona, Spain.

Xavier Vera was supported by Intel Corporation and VR grant no. 2001-2575.

Authors' addresses: J. Abella, Computer Architecture Department, Universitat Politècnica de Catalunya-Barcelona, Jordi Girona 1-3, Barcelona 08034, Spain; email: jabella@ac.upc.edu; A. González, and X. Vera, Intel Barcelona Research Center, Intel Labs, UPC, Jordi Girona 29, Barcelona 08034, Spain; email: antonio@ac.upc.edu, javierx.vera@intel.com; Michael F. P. O'Boyle, Institute for Computing Systems Architecture, School of Informatics, University of Edinburgh, Edinburgh, UK; email: mob@inf.ed.ac.uk.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2005 ACM 1544-3566/05/0300-055 \$5.00

Categories and Subject Descriptors: B3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C1.1 [**Processor Architectures**]: Single Data Stream Architectures—*RISC/CISC, VLIW architectures*

General Terms: Design, Performance

Additional Key Words and Phrases: Cache memories, low power, L2 cache, turning off cache lines

1. INTRODUCTION

Technology and microarchitecture evolution is driving microprocessors toward higher clock frequencies and higher integration scale. These two factors translate into higher power dissipation, which calls for more sophisticated and expensive cooling systems. Reducing the power dissipation can be beneficial not only in terms of energy savings, but also for reducing cooling costs or increasing performance for a given thermal solution.

Dynamic power and leakage are main sources of power dissipation in a processor. While dynamic power is directly related to the activity of the circuits, leakage depends on the amount of powered-on transistors and their physical characteristics. Dynamic power has been the more significant factor, but leakage is predicted to take over as it grows exponentially [SIA 2001]. L2 caches occupy a very large fraction of the area of most of the high-performance processors. Thus, the L2 cache represents an important source of the leakage for the chip and new techniques to reduce L2 cache leakage are needed.

Previous work proposed schemes to selectively turn off cache lines that are not expected to be needed in the future. Techniques to turn off cache lines are primarily designed for L1 caches; however, L2 caches have significantly different characteristics. L1 caches act as a filter for the L2 cache such that the L2 cache is only accessed by a relatively small number of the total L1 cache accesses. As we show in the evaluation section, L1-oriented approaches, when applied to L2, perform poorly due to this difference in behavior.

This paper introduces *IATAC* (inter-access time per access count), a new approach to reducing the leakage of L2 caches by turning off those cache lines that are not expected to be reused. For each cache line, it keeps track of the inter-access time and the number of accesses, providing decay times for each individual line. This mechanism is shown to perform significantly better than all previous approaches in terms of metrics that combine energy and execution time.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents our approach and two state-of-the-art mechanisms. The framework used for the evaluation as well as the results are presented in Section 4. Finally, Section 5 summarizes the main conclusions of this work.

2. RELATED WORK

This section describes previous work on low-power techniques for caches. The different approaches have been classified into two categories: low-level and high-level techniques. Low-level techniques are those that reduce power dissipation using mainly circuit techniques, while high-level techniques are mainly based on new cache organizations.

2.1 Low-Level Techniques

Several papers [Gonzalez et al. 1997; Itoh et al. 1995; Kuroda et al. 1998] investigate the effects on performance and power of tuning supply and threshold voltages. Abella and González [2003] study how to combine cache modules with different voltages for high performance and low power. Heo et al. [2002] present a circuit technique to reduce the leakage of bitlines by means of leakage-biased bitlines. They show that this technique makes the instruction cache (among other structures) less leaky.

Powell et al. [2000] propose gated- V_{DD} circuit technique, which allows cache lines to be gated but lose their contents. Gated- V_{DD} technique is based on using a transistor to gate the supply voltage (V_{DD}) or the ground path of the cache's SRAM cells. Flautner et al. [2002] propose drowsy caches. The main idea of this work is reducing the supply voltage by putting cache lines in drowsy mode (a kind of sleep mode) to reduce their leakage without losing their contents. Two supply voltages are required for each cache line, which is an overhead in the design of the processor. Agarwal et al. [2002] propose using gated-ground for turning off cache lines but still preserving their contents. This kind of circuits requires only one supply voltage, but they highlight that for technologies under 100 nm, small variations in the threshold voltage may destroy the contents of the cell. Thus, if a very high-precision threshold voltage cannot be achieved during the fabrication process, the stability of the cells cannot be guaranteed, making this technique nonviable. Kim et al. [2004] propose superdrowsy caches. They behave similarly to drowsy caches but only one V_{DD} is required. The main drawback of this approach is that cells in drowsy state are much more susceptible to soft errors [Li et al. 2004].

2.2 High-Level Techniques

Ghose and Kamble [1999] study the effects of using subbanking, multiple line buffers, and bitline segmentation to reduce dynamic power dissipation in caches. Su and Despain [1995] investigate vertical and horizontal cache partitioning, as well as Gray code addressing to reduce dynamic power. Hezavei et al. [2000] study the effectiveness of different low power SRAM design strategies such as divided bitline, pulsed wordline, and isolated bitline. Energy savings can also be achieved by dynamically [Balasubramonian et al. 2000; Dropsho et al. 2002] or statically [Albonesi 1999; Zhang et al. 2002, 2003] reconfiguring cache characteristics such as cache size, cache associativity, and cache active ways. Li et al. [2002] observe that L2 cache subblocks that are present in L1 can be turned off.

There are also different approaches based on changing the cache system organization. Kin et al. [1997] propose using a small filter cache in front of the conventional L1 cache to serve most of the accesses without accessing the L1 data cache. Yang and Gupta [2002] present a data cache design where frequent values are encoded to reduce the power dissipated when these data are accessed. Huang et al. [2001] propose a cache organization with different specialized modules for different types of accesses. Inoue et al. [1999] investigate the use of way predictors to access just one way in set-associative caches. Powell et al. [2001]

propose using way-prediction and selective direct mapping for nonconflicting accesses. This approach is based on the performance-oriented schemes by Batson and Vijaykumar [2001] and by Calder et al. [1996].

Buyuktosunoglu et al. [2003] propose an algorithm to gate fetch and issue to reduce the number of accesses to the instruction cache and other structures.

Zhou et al. [2001] and Kaxiras et al. [2001] have recently proposed different techniques to reduce leakage by switching off cache lines whose content is not expected to be reused. These techniques are described in detail in Section 3 as they have similar aims to our approach. We compare to them in Section 4 to evaluate the merits of our approach.

The IATAC mechanism presented in this paper saves energy by turning off L2 cache lines. Most of the work presented in this section is orthogonal to our approach, and many of them can be easily combined with our approach.

3. PREDICTORS FOR LARGE CACHES

In this section we introduce *IATAC*, a new mechanism to turn off L2 cache lines. We assume that cache lines are turned off using the traditional gated- V_{DD} circuit technique [Powell et al. 2000], which implies losing the contents of the cache lines that are gated, as it does not have those limitations of similar techniques introduced in the related work section. By way of comparison, we also present two state-of-the-art techniques [Kaxiras et al. 2001; Zhou et al. 2001] and explain why they do not work for L2 caches. The techniques presented in this section are decay-based techniques. Given a cache line, the decay interval is the number of cycles that have to elapse since the last access before the cache line is turned off.

3.1 *IATAC*

L1 caches filter out most stride 0 accesses. Stride 0 accesses frequently appear where a program variable is repeatedly loaded or stored, which usually happens in loops. These memory references frequently hit the same cache line and rarely incur misses. Thus, the L2 cache has to serve very few references with stride 0.

Furthermore, memory references normally access different L2 cache lines and the access pattern observed in a given cache line is likely to be observed in other cache lines. Similarly, patterns observed in other cache lines are also likely to be observed, in future, for this cache line. Hence, local cache line predictors are not expected to work well as they rely on using the information of a given cache line to only make predictions for this cache line; no information is shared across different cache lines.

In order to have an accurate prediction, we need a mechanism to perform global prediction of decay intervals. We have observed that patterns can be classified using the number of accesses to cache lines before they are replaced. Figure 1 shows the *time between hits* and the *time before miss* for four representative programs. Only representative access counts (corresponding to at least 2% of the occurrences) are plotted; other access counts are very rare so we neglect them. We observe that in some cases the *time between hits* remains the same for the different patterns (Applu always has a very small *time between*

Fig. 1. Average time between hits to cache lines (*time between hits*) and the average time from the last access to replacement (*time before miss*) against varying numbers of accesses. The results correspond to four representative programs (note logarithmic scale).

hits), whereas other programs show quite different *time between hits* for the different access counts (Facerec, Gcc, Parser). It can also be observed that the *time between hits* is shorter than the *time before miss*, which suggests turning off cache lines after waiting the typical *time between hits*. Another important observation is that the time we have to wait before turning off a cache line varies from one access count to another. For instance, we have to wait at least 500K cycles (the maximum *time between hits* for access counts larger than 1) before turning off a cache line that has been accessed once in Gcc (*time between hits* for two accesses in Gcc plot of Figure 1). After this number of cycles without any access, it is very unlikely that it will be accessed again. But if the cache line has been accessed five times, waiting 175K cycles (the maximum *time between hits* for access counts larger than 5, which corresponds to *time between hits* for seven accesses in Gcc plot of Figure 1) is enough.

We propose *IATAC*, which is a mechanism to turn off L2 cache lines. *IATAC* observes the interaccess time for each line, it classifies this information depending on the number of accesses to the line, and it makes predictions based on the global history.

The underlying idea can be illustrated with an example. Let us assume that a given cache line has been accessed for the N th time. To know if this is the last access or not, a decay interval is calculated based upon the interaccess time for

Fig. 2. Structures required for the *IATAC* mechanism for a 4 MB (512 KB) L2 cache.

those patterns with more than N accesses. Thus, if cache lines with more than N accesses usually have no more than K cycles between consecutive hits, it is convenient to use this value (K) as the decay interval for the accessed line. It is highly likely that either the cache line is accessed again within K cycles or it is not accessed anymore. Thus, it can be safely turned off after K cycles if no further access occurs. We turn off the data part but not the tag, which is always turned on. Clearly, recording all this information has a nonnegligible overhead. However, we will show that this overhead pays off due to the high accuracy of the mechanism.

Figure 2 depicts the structures required to implement *IATAC*. For those parameters with two values, the first one corresponds to the configuration for large caches (4 MB in our evaluation), while the second one (in parenthesis) corresponds to medium-size caches (512 KB in our evaluation). The following are the fields for each cache line:

- *wrong*: a field indicating whether the cache line was prematurely turned off.
- *onoff*: a field indicating whether the cache line is on or off.
- *decay*: the number of cycles that have to elapse to turn off the line.
- *thits*: a field to store the maximum time observed between consecutive hits¹ to the cache line.
- *elapsed*: a counter for the number of cycles elapsed since the line was last accessed.
- *counter*: a counter to keep track of the number of accesses to the cache line.

Conceptually, only two global structures are required. Both of them need to have as many entries as the maximum number of accesses we want to consider. At this point we assume *MAX_ACCESS* entries. The global structures are as follows:

- *acumcounter*: counts how many cache lines have been replaced for each value of the access counter.
- *globaldecay*: stores the decay required for each value of the access counter.

¹Including the miss that fetches the line before the first hit.

```

If there is a hit in the tag:
(1) counter = counter + 1;
(2) if (elapsed > thits)
(2)   thits = elapsed;
(2) elapsed = 0
(3) if (not onoff) wrong = 1;
(4) decay = 1;
(4) for (i=counter+1; i<=MAX_ACCESS; i=i+1)
(4)   if (acumcounter[i] > MIN_COUNT)
(4)     decay = MAX(decay, globaldecay[i]);

If there is a miss in the tag:
(5) if (thits > globaldecay[counter])
(5)   globaldecay[counter] = globaldecay[counter] x 2;
(5) else if (thits x 2 < globaldecay[counter])
(5)   globaldecay[counter] = globaldecay[counter] / 2;
(6) acumcounter[counter] = acumcounter[counter] + 1;
(6) if (acumcounter[counter] >= MAX_COUNT)
(6)   for (i=1; i<=MAX_ACCESS; i=i+1)
(6)     acumcounter[i] = acumcounter[i] / 2;
(7) decay = 1;
(7) for (i=counter+1; i<=MAX_ACCESS; i=i+1)
(7)   if (acumcounter[i] > MIN_COUNT)
(7)     decay = MAX(decay, globaldecay[i]);
(8) onoff = 1;
(8) wrong = 0;
(8) counter = 1;
(8) thits = 0;
(8) elapsed = 0;

```

Fig. 3. Algorithm of the IATAC mechanism.

Figure 3 details the mechanism. As it illustrates, different actions are taken depending on whether there is a hit or a miss. Note that if there is an access to a turned off line and there is a match in the tag, it is considered to be a hit by the algorithm even if the data cannot be served from the cache.

When there is a hit, the line's *counter* is incremented (1). The maximum time elapsed between hits is also updated (2). If the line was off, the *wrong* field is set indicating that there was a missprediction and the line cannot be turned off until it is replaced (3). Finally, a new decay is provided to the line. Conceptually, in this step the *decay* is set to the maximum value of the *globaldecay* (4), considering only those positions that correspond to access counts greater than *counter*, but only if these values are representative; that is only if these access counts have recently appeared frequently enough.

Different actions are taken when there is a miss. First of all, the interaccess time for cache lines having exactly *counter* accesses (*globaldecay[counter]*) is updated. As shown in step (5) of Figure 3, if *thits* is greater than the *counter* position of *globaldecay*, the interaccess time is multiplied by 2. But if *thits* is lower than half of the *counter* position of *globaldecay*, the interaccess time is divided by 2. This way we update the decay for future predictions, adjusting it to the recent history. The second action (6) is updating *acumcounter* that keeps track of which access counts are significant. The *counter* position is incremented, and if it reaches the maximum value, all positions are divided by 2. This way

acumcounter indicates which access counts have recently appeared and how many times with respect to the other access counts. The *decay* field (7) is updated in the same way as the case of having a hit. Thus, steps (4) and (7) are exactly the same. Finally, the other cache line's fields are initialized (8).

It should be noted that, for any cache line, when *elapsed* reaches the value stored in *decay*, if *wrong* is not set, the cache line is turned off. Finally, the tag array of the cache is always on.

3.1.1 Implementation. Some issues have to be addressed to make the implementation of the *IATAC* mechanism feasible. We use hierarchical counters [Kaxiras et al. 2001] to keep track of the cycles elapsed since each cache line was last accessed. Decay intervals are in the range of tens or hundreds of thousands of cycles. Such large decay intervals make it impractical for the counters to count cycles (too many bits would be required). Instead, it is necessary for the counters to tick at much coarser level. The solution is to use a hierarchical counter mechanism where a single global cycle counter is set up to provide the ticks for smaller cache-line counters. For instance, the global counter may send a tick to the local counters every 1000 cycles, as assumed in our experiments. If we use Gray coding for the local counters only one bit per counter will change state at any time.

thits and *elapsed* fields should have a maximum value in order to determine the number of bits required. We have observed empirically that, values higher than 8192 do not provide any advantage, and even for a medium-size cache, 1024 is enough. Hence, these fields require 13 bits each if the maximum is 8192 or 10 bits if the maximum is 1024. The *decay* field and the entries of *globaldecay* do not require as many bits. They can be only a power-of-2, since *globaldecay* positions are set initially to 1 and increased (decreased) multiplying (dividing) by 2, and *decay* is set to one of the values stored in *globaldecay*. There are 13 or 10 different powers-of-2 depending on whether the maximum value is 8192 or 1024, respectively. In any case 4 bits are enough to encode that number of different values. The maximum number of accesses (*MAX_ACCESS* in Figure 3) that are considered sets the length of *globaldecay* and *acumcounter* as well as the size of the *counter* field. Setting *MAX_ACCESS* to values greater than 32 does not provide higher accuracy. For medium-size caches, this value can be set to 16. Lower values of *MAX_ACCESS* make *IATAC* less aggressive and opportunities to save energy are lost. Thus, the length of *globaldecay* and *acumcounter* is *MAX_ACCESS*, and *counter* requires $\log_2(\text{MAX_ACCESS})$ bits. *MAX_ACCESS* value is strongly related to the L2 cache size.

In the implementation we use an additional structure that we call *maxglobaldecay*. It has as many positions as *globaldecay*. Each position *P* records the maximum value of *globaldecay* between positions *P* + 1 and *MAX_ACCESS* for those positions whose corresponding entry in *acumcounter* is greater than or equal to *MIN_COUNT*. Thus, we do not need to compute any maximum when a line is accessed since it has been previously computed. The experiments show that one position of *globaldecay* is updated once every 9 misses to L2 cache, and *maxglobaldecay* is updated once every 3500 misses.

The last parameters to describe are *MAX_COUNT* and *MIN_COUNT*. *MAX_COUNT* is the upper bound of *acumcounter* entries. While none of the entries of *acumcounter* reaches this value, the number of occurrences of the different values of *counter* are recorded. Thus, using large values for *MAX_COUNT* makes observation periods longer. If these periods are too long, it may be the case that *acumcounter* stores old records mixed with current ones. Conversely, if these periods are too short, it may be that *acumcounter* is frequently decreased (dividing all positions by 2) and the counters of infrequent values may be set to values under *MIN_COUNT*. In this case, their interaccess times are ignored to compute the decay and the mechanism, therefore, becomes less accurate. Setting *MAX_COUNT* to 1024 provides a good trade-off. Thus, each position of *acumcounter* requires 10 bits to store any value between 0 and *MAX_COUNT*.

We have set *MIN_COUNT* to 8 in our experiments. That means that only access counts with an *acumcounter* value equal or higher than 8 will be considered to make predictions. Lower values make *IATAC* more conservative since we always select the maximum decay and rare access counts' decays are considered. On the other hand, larger values make *IATAC* more aggressive because only those frequently occurring access counts are considered.

MAX_COUNT and *MIN_COUNT* must be set in concert. For instance, if we decide to set *MAX_COUNT* to 16,384 (1024×16), we must scale *MIN_COUNT* in the same way (8×16) to prevent rare access counts being considered. If we use a larger value for *MAX_COUNT* and do not increase *MIN_COUNT*, we have that *acumcounter* positions are divided by two less frequently and there is more time for infrequent access counts to reach *MIN_COUNT* and be considered.

3.1.2 Additional Comments. We have used hierarchical counters for the decay intervals because they require less hardware than conventional counters and expend less energy. The loss of accuracy introduced by this kind of counters is negligible since the decay intervals normally are some orders of magnitude higher than the update period for the counters (1000 cycles).

The logic to compute the decay interval for a given line may take a few cycles, but it is done after serving the cache access. Thus, the cache latency and access time are not affected. Delaying the computation of the decay interval by a few cycles has little impact on power and performance (decay intervals normally are between 10K and 1000K cycles).

IATAC and the compared mechanisms consider all accesses to the L2 cache, including those due to copying back L1 dirty lines. Considering or not this kind of accesses does not change the results. We also assume that multiple updates to the structures of *IATAC* (and the other mechanisms) can be done immediately. If L2 cache accesses are hits, they just update the line's local information. Global structures are affected only by misses. If these updates have to be done sequentially, a small queue with as many entries as memory outstanding misses will suffice to update the predictor properly. Delaying updates by a few cycles has a negligible impact on energy savings since cache lines are turned off for tens or hundreds of thousands of cycles.

3.2 State-of-the-Art Approaches

This section describes two state-of-the-art approaches, which also rely on turning off cache lines to save leakage energy. They are extensively evaluated against our proposed technique, *IATAC*, in Section 4.

3.2.1 Cache Decay. This work [Hu et al. 2002; Kaxiras et al. 2001] relies on the observation that for L1 data caches it is very common to have all the hits to a cache line concentrated in a short period of time. They are then followed by a long period without any access to the cache line, which is then finally replaced. Based on this observation they propose two mechanisms.

3.2.1.1 Fixed Decay Interval. The first mechanism assumes a fixed decay interval for the whole cache and program execution. We will refer to this mechanism as *decayN* in the rest of the paper where *N* stands for the decay interval. For instance, if the decay interval is set to 64,000 cycles, the mechanism is referred to as *decay64K*. *DecayN* provides high power savings with a small performance degradation for L1 data caches, since cache lines are turned off soon after the last hit.

Different programs may achieve maximum leakage reduction with different decay intervals. The reason for this fact is that memory intensive programs have shorter interhit intervals as well as shorter inactive intervals than nonmemory intensive programs. Additionally, different cache lines may require different decay intervals. This fact is particularly significant for irregular applications where the access patterns to distinct cache lines are quite different.

Due to the much larger size of L2 caches, data take longer to be replaced compared to the L1 data cache. Furthermore, the time elapsed between hits to a given cache line is quite irregular and it may be even longer than the inactive period after the last hit. Therefore, a long decay interval may not turn off cache lines whose data will not be used again before being replaced. However, a short decay may turn off cache lines too early, increasing the number of misses.

3.2.1.2 Adaptive Decay Interval. The second mechanism improves upon the first one in the sense that it tries to find individual decay intervals for each cache line which are adapted dynamically. This mechanism (we will refer to it as *decayAdap*) tries to choose the smallest possible decay interval for each cache line individually. A L1 cache line normally has a sequence of hits for a brief period, and then remains unused for a long time until it is replaced. *DecayAdap* increases the decay when a miss occurs soon after turning off a line as this access would have been a hit if the cache line was not turned off prematurely. On the other hand, if the miss takes place long after turning off the line, the decay is decreased as this access would have been a hit, even if the cache line was not turned off. If the time elapsed after turning off the cache line and the corresponding miss is neither too short but not too large, then the decay interval remains the same.

This mechanism shows good results for L1 data caches, but it has some drawbacks for L2 caches. First of all, *decayAdap* uses local history for the predictions. Since L1 caches serve most of the accesses with stride zero, the L2 cache has

Fig. 4. Mechanism to update the decay interval for the adaptive mode control.

to serve memory instructions with nonzero strides, which normally access different lines. Hence, the information about the behavior of a particular line is useful for other cache lines if accessed by the same memory instructions and, therefore, global history can make much more accurate predictions. Another important drawback of *decayAdap* resides in the fact that, instead of keeping the tags on to realize if a cache line has been turned off so early or not, it tries to infer the cause of misses. If periods between consecutive hits are long enough, cache lines are prematurely turned off and the prediction, which is wrong, is considered successful. Hence, instead of increasing the decay interval to correct the wrong prediction, it is decreased, and the next prediction is also likely to be wrong.

3.2.2 Adaptive Mode Control. The adaptive mode control [Zhou et al. 2001] is also a decay-based technique that uses a global decay interval for the whole cache. This mechanism is similar to *decayN* but it allows N to be adapted dynamically depending on the fraction of extra misses introduced by the mechanism. This mechanism does not turn off the tag array, which helps to detect whether a miss is incurred or not by their mechanism. Obviously, not turning off the tag array implies that some leakage cannot be saved.

The implementation of this mechanism is similar to that of the *decayN*, but in this case the decay is updated periodically after a certain number of cycles. Figure 4 shows how this mechanism works. Ideal misses stands for the number of misses that would have occurred regardless of a line's sleep/active status, whereas sleep misses are those specifically caused by sleep-mode lines. A sleep miss occurs when a matching tag is found but the data portion is in sleep mode. At the end of the period the target error is computed, which corresponds to the ideal number of misses multiplied by a performance factor (PF).

- If the number of sleep misses is higher than 1.5 times the target error, the decay interval (N) is increased in order to be less aggressive. The new decay is $2 \times N$.
- If the number of sleep misses is below 0.5 times the target error, the decay interval is decreased. The new decay interval is $N/2$.

The performance factor (PF) is set to a power-of-2 (potentially negative) and it does not change during the program execution. We will refer to this technique as *AMCPF* in the rest of the paper. For instance, if PF is $1/4$ we will refer to it as *AMC0.25*.

This mechanism is shown to be very efficient for L1 caches (both data and instructions), but it has some limitations for the L2 cache. For instance, as we show in Section 4, the decay is not homogeneous across all the cache lines. Thus, assuming the same decay for all the cache lines may miss significant energy savings and/or degrade performance due to extra misses. Another intrinsic problem of this technique resides in the fact that it is targeting a given error. For some programs, maximum energy savings can be achieved without increasing the miss ratio. Let us assume that the current decay interval is N , PF is $1/2$ (or smaller), and the typical period between two consecutive accesses to a given cache line is always in the range $(N: 2 \times N)$. During this period there will be as many misses as cache accesses, and thus the new decay interval will be set to $2 \times N$. During the following period there will not be sleep misses, making the decay interval decrease to N . Again there will be as many misses as cache accesses, and the pattern repeats. At the end, during half of the program execution there are as many misses as accesses, which causes significant performance degradation.

4. EVALUATION

This section presents the results for the proposed mechanism as well as those used for comparison purposes. First we describe the tools used for the evaluation, the benchmarks considered and the processor configuration. Later we discuss the results in terms of performance and energy.

4.1 Experimental Framework

Dynamic power and performance results are derived from CACTI 3.0 [Shivakumar and Jouppi 2001], which is a timing, power, and area model for cache memories, and an enhanced version of Wattch [Brooks et al. 2000b], which is an architecture-level power and performance simulator based on SimpleScalar [Burger and Austin 1999]. The main enhancements are the separation of the reorder buffer and the issue queue, and modeling ports for the register file. Leakage is modeled taking into consideration the number of bit cells that are active for each mechanism and the 3% area increase [Powell et al. 2000] of the cache cells due to the logic needed to turn off the cache lines. This increase is assumed to result in a 3% additional leakage. Section 4.4 shows the detailed energy model.

Table I shows the processor configuration. Two different sizes (512 KB and 4 MB) for the L2 cache have been evaluated. Since using different latencies for both cache sizes does not provide significant insights, we assume the same latency for both of them. We consider 100-cycle memory latency, since it is expected that future processors will have very large off-chip L3 caches with latencies much lower than main memory [Sinharoy 2004].

We have used the whole Spec2000 benchmark suite [SPEC 2000] with the *ref* input data set. We have simulated 1 billion instructions for each benchmark

Table I. Processor Configuration

Fetch, decode, commit width: 8 instructions
Issue width: 8 INT + 8FP instructions
Branch predictor: Hybrid 2K Gshare, 2K bimodal, 1K selector
BTB: 2048 entries, 4-way
L1 Icache: 64 KB, 2-way, 32 byte line (1 cycle)
L1 Dcache: 8 KB, 4-way, 32 byte line, 4 R/W ports (2 cycles)
L2 unified cache: 512 KB/4 MB, 4-way, 64 byte line (10 cycles hit, 100 cycles miss, 2 cycles interchunk)
ITLB: 128 entries fully-associative (1 cycle)
DTLB: 128 entries fully-associative (1 cycle)
Fetch queue: 64 entries
Issue queue: 128 INT + 128 FP entries
Reorder buffer: 256 entries
Load/store queue: 128 entries
Register file: 160 INT + 160 FP registers
INT functional units:
6 ALU (1 cycle)
3 mult/div (3 cycles mult, 20 cycles nonpipelined div)
FP functional units:
4 ALU (2 cycles)
2 mult/div (4 cycles mult, 12 cycles nonpipelined div)

Fig. 5. IPC degradation for the different mechanisms.

after skipping the initialization part and warming up caches for 100 million instructions. The benchmarks have been compiled with the HP/Alpha compiler with `-O4 -non_shared` flags.

4.2 Configurations Evaluated

For the 512 KB cache, results are reported for *IATAC* as well as for different configurations of the related approaches: *decay64K*, *decay128K*, *decay256K*, *decay512K*, *decay1024K*, *decayAdap*, *AMC0.5*, *AMC0.25*, *AMC0.125*, *AMC0.0625*, and *AMC0.03125*. We chose these configurations because lower values of N for *decayN* degrade significantly performance (see Figure 5), whereas higher values of N make *decayN* too conservative (see Figure 6). The situation for *AMCPF* is different: higher values of PF are excessively aggressive and the performance loss is high (see Figure 5), but lower values of PF hardly change

Fig. 6. L2 turn-off cache line ratio for the different mechanisms.

Fig. 7. L2 miss ratio for the different mechanisms.

the performance results (IPC and turn-off ratio keep the same). As stated in Section 3, there are some pathologic (and quite frequent) situations where *AMCPF* loses significant performance whatever the *PF* value is, and thus, lower values of *PF* do not change the results. For the 4 MB cache, we report results for the proposed *IATAC* mechanism. We also report results for *decay512K*, *decay1024K*, *decay2048K*, *decay4096K*, *decay8192K*, *decayAdap*, *AMC0.5*, *AMC0.25*, *AMC0.125*, *AMC0.0625*, and *AMC0.03125*. The baseline starts with the whole cache turned off, and it never turns off any cache line once it has been accessed.

4.3 Performance Evaluation

In this section we present results showing IPC degradation, the ratio of cache lines turned off, and L2 miss ratio for the different mechanisms.

Figure 5 shows the performance degradation for both cache sizes for all mechanisms. Performance of individual benchmarks has been averaged using the harmonic mean. Similarly, Figure 6 shows the turn-off ratios and Figure 7 the miss ratios for the different mechanisms.

In Figure 6 *ideal* stands for the oracle mechanism that turns off every cache line right after the last hit. We observe that the baseline has a turn-off ratio higher than 0. That is so because some programs (*perlbnk* and *eon* among

others) do not fully utilize the L2 cache. Since the simulations start with the whole cache turned off, some cache lines are never accessed and thus never turned on.

We observe that for *decayN*, increasing N reduces the performance degradation at the price of having a more conservative mechanism. That translates into a lower miss ratio and lower turn-off ratio. *AMCPF* shows a similar trend when PF is decreased, but eventually the results stabilize and lower values for PF hardly change the results. For the 512 KB cache, PF values lower than 0.03125 obtain results very close to those of *AMC0.03125*. The same trend is observed for the 4 MB cache. As outlined in Section 3.2.2, *AMCPF* targets a given error and thus, if an optimal situation is achieved (maximum power savings and no extra misses) the mechanism is self-tuned and becomes more aggressive, degrading performance significantly most of the times. We also observe that *AMC* techniques perform relatively close to *IATAC* for the 4 MB cache in terms of turn-off ratio, but not for the 512 KB cache. This is so because the 4 MB L2 cache is underutilized for many programs leading to these high turn-off ratios.

For the 512 KB cache it can be seen that other mechanisms produce a low IPC loss, similarly to *IATAC*. However, *IATAC* achieves a much higher turn-off ratio than all the other mechanisms with an IPC degradation of 1.4%. Only *decay1024K* loses less performance, but its turn-off ratio is extremely low.

We observe that the performance loss for *decayAdap* for a 4 MB L2 cache is huge. As stated by Hu et al. [2002]: “the idea is to start with a short decay interval, detect whether it was a mistake, and adjust the decay interval accordingly.” For many programs, once the decay interval is elapsed, a significantly greater period of time elapses before there is an access to a particular cache line, which would be a hit if not were for the fact that it is turned off. Since the tags are off, the mechanism does not realize that it has been a decay miss, and instead, since the time elapsed after turning the line off is quite large, the mechanism decides that it was not a decay miss and tries to make the decay interval shorter. Thus, most of the L2 cache lines are off, but the performance is severely harmed. For smaller caches *decayAdap* does not lose as much performance because cache lines are reused more frequently and, even if the decay interval is small, it is easy to quickly have an access to the turned off cache line and decide that it was turned off prematurely. We must state that *decayAdap* was proposed for L1 caches, and the authors did not evaluate it for L2 caches. We have included this mechanism in the evaluation for the sake of comparison.

It can be observed that *IATAC* has a similar or even higher miss ratio and lower performance degradation than some versions of *decayN*. Our approach slightly increases the miss ratios in many programs, which produce a small performance loss in each program. *DecayAdap*, *decayN*, and *AMCPF* usually increase the miss ratio significantly for a few programs. That translates into significant performance degradation for these programs and also for the harmonic mean. In order to show this fact, we show detailed results for all benchmarks for *decay512K*, *decayAdap*, *AMC0.0625*, and *IATAC*. Figure 8 shows the IPC degradation, whereas Figure 9 shows the number of misses (in logarithmic scale). It can be seen that *IATAC* is the worst behaving mechanism in terms of

Fig. 8. IPC loss for spec2000 benchmarks and a 512 KB L2 cache.

Fig. 9. Number of misses for spec2000 benchmarks and a 512 KB L2 cache.

miss ratio and IPC only for the swim benchmark. Swim is a program with high ILP that has a large number of misses to the L2 cache. Increasing the number of misses has negligible performance impact as their latency is overlapped with other misses. *IATAC* has also a larger number of misses for the perlbnk program, but there are only 200 misses. The figures show that there is a high correlation between significantly increasing the number of misses and degrading the performance. For instance, *decayAdap* is the worst behaving technique for ammp, crafty, eon, fma3d, gzip, mesa, and vortex programs in terms of misses and IPC. In general, as expected, a higher number of misses means a higher performance loss.

For the 4 MB L2 cache, it can be observed that the turn-off ratio is significantly higher than for the 512 KB cache for some versions of *decayAdap*, *decayN*, and *AMCPF* techniques, but the performance loss is also higher. Only *decay4096K* and *decay8192K* produce performance degradation slightly lower than *IATAC*, but at the expense of a much lower turn-off ratio. *IATAC* is the only mechanism achieving high turn-off ratio and low-performance degradation for medium and large cache sizes.

4.3.1 Dynamic Behavior. In order to show how *IATAC* adapts decay values dynamically, we present some results describing the predictive behavior of the *IATAC* scheme for the 512 KB L2 cache. Results for the 4 MB cache show the same trend. Figure 10 shows the standard deviation normalized with respect

Fig. 10. Decay interval coefficient of variation.

to the mean (coefficient of variation, or CV for short) for the predicted decay intervals, computed in two different ways:

- *Generation*: every time a prediction is performed, the CV of the latest prediction for every cache line is computed. These values are averaged. This gives an indication of the variability of decay intervals across different lines at a given point in time. We have observed that the result is greater than 100% for half of the programs. That confirms that assuming the same decay interval simultaneously for all the cache lines is not worth doing since different lines require different decay intervals. That explains why *decayN* and *AMCPF* do not work well for the L2 cache.
- *Cache line*: we compute the CV for the predictions of each individual cache line. Hence, we get as many values as cache lines. The final result is the average of all these values. This gives an indication of the variability of the decay intervals across time for a given cache line. We have observed that this value is quite high, indicating that using local history for a given cache line to make new predictions is unreliable.

It is not frequent the case that the predictions for a given cache line are constant, and at some point in time, they change to a different value that also remains constant. In fact, we have observed that *cache line* CV is high because decay intervals for a given cache line vary frequently. This explains why *decayAdap* performs poorly for the L2 cache. Additionally, *decayAdap* does not keep the tags on, and tries to infer the cause of misses instead of detecting whether or not they have been produced by an early deactivation of the cache line. While this approach is beneficial for L1 caches, it is inaccurate for the L2 cache.

4.4 Energy-Delay Efficiency

We have shown that *IATAC* can turn off a significant portion of the L2 cache at the expense of a low performance degradation. However, it does require some additional hardware. In this section we evaluate, for the 4 MB L2 cache, the impact of the different approaches using appropriate power-performance metrics

Table II. Energy Model

Energy distribution for the whole chip: 50% dynamic + 50% leakage
Fraction of leakage of the L2 cache: 33% (512 KB) or 50% (4 MB)
Dynamic energy includes <ul style="list-style-type: none"> (1) Dynamic energy increase due to larger execution time for the different mechanisms (2) Extra L2 misses (3) Overhead due to additional structures
Leakage energy includes <ul style="list-style-type: none"> (4) Leakage increase due to larger execution time for the different mechanisms (5) 3% extra leakage overhead for the L2 cache due to gated-V_{dd} technique (6) Overhead due to additional structures

[Brooks et al. 2000a]. Thus, we consider energy, EDP (energy-delay product) and ED^2P (energy-delay² product).

The SIA roadmap [SIA 2001] and Heo et al. [2002] among others estimate that the leakage will represent about 50% of the energy consumption of the chip for 70 nm technology. Others predict even a higher percentage [SYNOPSIS 2002]. In our evaluations we assume 50%. Additionally, we are conservative and assume that L2 cache leakage may represent as much as 50% of the total leakage energy, even though other authors [Li et al. 2002] report up to 75% contribution. Table II summarizes the energy model that has been considered.

All techniques introduce some extra misses (2) when cache lines are turned off too early. The consequent increase in execution time impacts both dynamic (1) and leakage (4) energy. The larger the execution time, the higher the leakage energy consumed in all the structures but the L2 cache, some of whose lines are turned off. Also the dynamic energy grows due to the activity that there is every cycle, like selecting instructions from the issue queue. The dynamic energy overhead (3) for the additional structures is the number of bits read and written multiplied by the energy consumption of accessing a bit of the L2 cache. The energy per L2 bit is obtained by dividing the energy consumption of an L2 access by the number of bits accessed. Gated- V_{DD} technique (5) requires a 3% area overhead to be implemented. We assume that increasing the L2 area by 3% increases its leakage by the same percentage. Finally, the leakage overhead (6) due to additional structures is computed in a similar way. We assume that each bit of the additional structures has the same leakage as a bit in the L2 cache. For instance, for the 512 KB L2 cache *IATAC* requires 288 bits for the global structures and 31 bits for each cache line. Each cache line has 560 bits (48 tag + 512 data), and there are 8192 cache lines. Thus, the L2 cache has 4,587,520 bits and the additional structures require 254,240 bits, which is 5.5% leakage overhead. We also consider the effects of tags leakage for those approaches that do not turn them off (*AMCPF*, *IATAC*).

We have not taken into account the extra off-chip energy consumption. On one hand, it is hard to estimate the energy consumption of an off-chip access. On the other hand, even if the extra off-chip energy consumption is higher than the on-chip energy savings, the main benefit of reducing the on-chip energy consumption is that temperature can be reduced and the cooling system can be simpler or a higher performance can be achieved for a given thermal solution.

Table III. Energy Breakdown for *IATAC* Mechanism

	<i>IATAC</i> (512 KB L2)	<i>IATAC</i> (4 MB L2)
A. Dynamic energy	50.0%	50.0%
B. Dynamic overhead due to IPC loss	0.6%	0.8%
C. Dynamic overhead due to extra L2 misses	0.5%	0.6%
D. Dynamic overhead due to additional structures	0.6%	0.8%
Total dynamic energy	51.7%	52.2%
E. Leakage energy	50.0%	50.0%
F. Leakage energy savings	−11.2%	−16.0%
G. Leakage overhead due to IPC loss	0.5%	0.9%
H. Leakage overhead due to gated-Vdd	0.6%	0.8%
I. Leakage overhead due to additional structures	1.0%	1.6%
Total leakage energy	40.9%	37.3%
Total chip energy	92.6%	89.5%
Total chip energy savings	7.4%	10.5%

Fig. 11. Energy consumption for the different mechanisms.

Table III shows the energy breakdown for the *whole chip* when *IATAC* is used. It can be observed that the energy savings are very high and thus, the overhead (dynamic and static) for our mechanism is more than compensated. In the table, B (G) stands for the extra dynamic energy (leakage) due to having a larger execution time. C accounts for the extra dynamic energy in the L2 cache to write data that were evicted too early, and to read data that has to be written back to the memory again. D is the dynamic energy of the structures required by the *IATAC* mechanism. F corresponds to the leakage energy savings in the L2 cache. H accounts for the extra leakage energy required by the Gated- V_{DD} technique as stated before. I is the leakage of the extra bits per cache line and structures required by the *IATAC* mechanism.

Figures 11, 12 and 13 show the relative energy, EDP, and ED^2P required by the different mechanisms. It can be seen that *IATAC* is clearly the best performing technique in terms of the last two metrics, which combine energy and performance. *IATAC* has higher accuracy to identify useless cache lines at the expense of a very small performance degradation. Note that *decay512K* and *decay1024K* save slightly more energy than *IATAC* for the 4 MB cache, but

Fig. 12. EDP for the different mechanisms.

Fig. 13. ED²P for the different mechanisms.

that comes at the expense of high IPC degradation. In terms of ED²P, *IATAC* performs 7% (6%) better than the baseline and 7% (4%) better than the second best technique for the 512 KB (4 MB) L2 cache.

To summarize, *IATAC* achieves the highest energy savings, and its performance loss is very low. Those mechanisms with energy savings close to those of *IATAC* have higher performance degradation. Overall, in terms of ED²P, *IATAC* outperforms all other mechanisms. In this work, we focus on reducing L2 leakage, which is 16.7% (25%) of the whole chip energy for the 512 KB (4 MB) L2 cache. The *ideal* mechanism that turns off each cache line after its last use saves 13.4% (17.8%) of the whole chip energy whereas *IATAC* saves as much as 7.4% (10.5%) energy, which is more than half of the ideal mechanism energy savings. *IATAC* achieves these high-energy savings at the expense of very low-performance loss.

5. CONCLUSIONS AND FUTURE WORK

This paper introduces *IATAC*, a new microarchitectural technique to reduce cache leakage for L2 caches, which is expected to be an important source of energy consumption in future processors. *IATAC* dynamically adapts the decay interval for each cache line individually based on global history. To the best of our knowledge, this is the first approach that is based on identifying

global access patterns to predict the appropriate decay interval for each individual cache line. We show that considering the number of accesses to the cache lines and the decay intervals together is a promising approach to developing predictors to turn off cache lines. *IATAC* outperforms all other previous approaches since its turn-off ratio is very high (around 65%) and its performance loss is low. Besides, it comes at the expense of very small additional hardware. *IATAC* provides the best ED²P across all mechanisms for different cache sizes. For instance, *IATAC* improves ED²P by 7% (6%) for a 512 KB (4 MB) L2 cache.

Finally, there is still some room for improvement. The turn-off ratio achieved by *IATAC* is close to the *ideal*, but that comes at the expense of a small performance degradation. Although performance loss is small, it has significant impact on the ED²P metric. Thus, new predictors that achieve a similar level of accuracy as *IATAC* but further reduce the IPC loss will be interesting. Additionally, enabling the compiler to provide some hints to the hardware predictor, such as the expected number of accesses to a cache line, may improve the accuracy of the predictor.

REFERENCES

- ABELLA, J. AND GONZÁLEZ, A. 2003. Power efficient data cache designs. In *Proceedings of the IEEE 21st International Conference on Computer Design (ICCD'03)*, San Jose, CA, October 2003, 8–13.
- AGARWAL, A., LI, H., AND ROY, K. 2002. DRG-cache: A data retention gated-ground cache for low power. In *Proceedings of the ACM/IEEE 39th Design Automation Conference (DAC'02)*, New Orleans, LA, June 2002, 473–478.
- ALBONESI, D. H. 1999. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the ACM/IEEE 32nd International Symposium on Microarchitecture (MICRO'99)*, Haifa (Israel), Nov. 1999, 248–259.
- BALASUBRAMONIAN, R., ALBONESI, D., BUYUKTOSUNOGLU, A., AND DWARKADAS, S. 2000. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the ACM/IEEE 33rd International Symposium on Microarchitecture (MICRO'00)*, Monterey, CA, Dec. 2000, 245–257.
- BATSON, B. AND VIJAYKUMAR, T. N. 2001. Reactive-associative caches. In *Proceedings of the IEEE 10th International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, Barcelona, Spain, Sept. 2001, 49–60.
- BROOKS, D. M., BOSE, P., SCHUSTER, S. E., JACOBSON, H., KUDVA, P. N., BUYUKTOSUNOGLU, A., WELLMAN, J. D., ZYUBAN, V., GUPTA, M., AND COOK, P. W. 2000a. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro* 20, 6 (Nov.-Dec.), 26–44.
- BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000b. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the ACM/IEEE 27th International Symposium on Computer Architecture (ISCA'00)*, Vancouver, Canada, June 2000, 83–94.
- BURGER, D. AND AUSTIN, T. 1999. *The simpleScalar tool set*, version 3.0. Technical Report, Computer Sciences Department, University of Wisconsin-Madison, 1999.
- BUYUKTOSUNOGLU, A., KARKHANIS, T., ALBONESI, D. H., AND BOSE, P. 2003. Energy efficient co-adaptive instruction fetch and issue. In *Proceedings of the ACM/IEEE 30th International Symposium on Computer Architecture (ISCA'03)*, San Diego, CA, June 2003, 147–156.
- CALDER, B., GRUNWALD, J., AND EMER, J. 1996. Predictive sequential associative cache. In *Proceedings of the IEEE 2nd International Symposium on High-Performance Computer Architecture (HPCA'96)*, San Jose, CA, Feb. 1996, 244–253.
- DROPSHO, S., BUYUKTOSUNOGLU, A., BALASUBRAMONIAN, R., ALBONESI, D. H., DWARKADAS, S., SEMERARO, G., MAGKLIS, G., AND SCOTT, M. L. 2002. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Proceedings of the IEEE 11th International Conference on Parallel Architectures and Compilation Techniques (PACT'02)*, Charlottesville, VA, Sept. 2002, 141–152.

- FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., AND MUDGE, T. 2002. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the ACM/IEEE 29th International Symposium on Computer Architecture (ISCA'02)*, Anchorage, AK, May 2002, 148–157.
- GHOSE, K. AND KAMBLE, M. B. 1999. Reducing power in superscalar processor caches using sub-banking, multiple line buffers and bit-line segmentation. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'99)*, San Diego, CA, Aug. 1999, 70–75.
- GONZALEZ, R., GORDON, B. M., AND HOROWITZ, M. A. 1997. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits* 32, 8 (Aug.), 1210–1216.
- HEO, S., BARR, K., HAMPTON, M., AND ASANOVIC, K. 2002. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proceedings of the ACM/IEEE 29th International Symposium on Computer Architecture (ISCA'02)*, Anchorage, AK, May 2002, 137–147.
- HEZAVEI, J., VIJAYKRISHNAN, N., AND IRWIN, M. J. 2000. A comparative study of power efficient SRAM designs. In *Proceedings of the ACM 10th Great Lakes Symposium on VLSI (GLSVLSI'00)*, Evanston, IL, Mar. 2000, 117–122.
- HU Z., KAXIRAS, S., AND MARTONOSI, M. 2002. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. *ACM Transactions on Computer Systems* 20, 2 (May), 161–190.
- HUANG, M., RENAULT, J., YOO, S. M., AND TORRELLAS, J. 2001. L1 data cache decomposition for energy efficiency. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'01)*, Huntington Beach, CA, Aug. 2001, 10–15.
- INOUE, K., ISHIHARA, T., AND MURAKAMI, K. 1999. Way-predictive set-associative cache for high performance and low energy consumption. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'99)*, San Diego, CA, Aug. 1999, 273–275.
- ITO, K., SASAKI, K., AND NAKAGOME, Y. 1995. Trends in low-power RAM circuit technologies. *Proceedings of the IEEE* 83, 4 (Apr.), 524–543.
- KAXIRAS, S., HU Z., AND MARTONOSI, M. 2001. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the ACM/IEEE 28th International Symposium on Computer Architecture (ISCA'01)*, Göteborg, Sweden, June 2001, 240–251.
- KIM, N. S., FLAUTNER, K., BLAAUW, D., AND MUDGE, T. 2004. Single-VDD and Single-VT super-drowsy techniques for low-leakage high-performance instruction caches. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'04)*, Newport Beach, CA, Aug. 2004, 54–57.
- KIN J., GUPTA, M., AND MANGIONE-SMITH, W. H. 1997. The filter cache: An energy efficient memory structure. In *Proceedings of the ACM/IEEE 30th International Symposium on Microarchitecture (MICRO'97)*, Research Triangle Park, North Carolina, Dec. 1997, 184–193.
- KURODA, T., SUZUKI, K., MITA, S., FUJITA, T., YAMANE, F., SANO, F., CHIBA, A., WATANABE, Y., MATSUDA, K., MAEDA, T., SAKURAI, T., AND FURUYAMA, T. 1998. Variable supply-voltage scheme for low-power high-speed CMOS digital design. *IEEE Journal of Solid-State Circuits* 33, 3, (Mar.), 454–462.
- LI L., DEGALAHAL, V., VIJAYKRISHNAN, N., KANDEMIR, M., AND IRWIN, M. J. 2004. Soft error and energy consumption interactions: A data cache perspective. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'04)*, Newport Beach, CA, Aug. 2004, 132–137.
- LI L., KADAYIF, I., TSAI, Y.-F., VIJAYKRISHNAN, N., KANDEMIR, M., IRWIN, M. J., AND SIVASUBRAMANIAM, A. 2002. Leakage energy management in cache hierarchies. In *Proceedings of the IEEE 11th International Conference in Parallel Architectures and Compilation Techniques (PACT'02)*, Charlottesville, VA, Sept. 2002, 131–140.
- POWELL, M., AGARWAL, A., VIJAYKUMAR, T. N., FALSAFI, B., AND ROY, K. 2001. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the ACM/IEEE 34th International Symposium on Microarchitecture (MICRO'01)*, Austin, TX, Dec. 2001, 54–65.
- POWELL, M., YANG, S. H., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. N. 2000. Gated-V_{DD}: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'00)*, Rapallo (Italy), July 2000, 90–95.
- SHIVAKUMAR, P. AND JOUPPI, N. P. 2001. *CACTI 3.0: An Integrated Cache Timing, Power and Area Model*. Research Report 2001/2, WRL, Palo Alto, CA (USA), 2001.

- SINHARROY, B. 2004. Power5 Architecture and Systems. Keynote at *the IEEE 10th International Symposium on High-Performance Computer Architecture (HPCA'04)*, Madrid, Spain, Feb. 2004.
- SU, C. L. AND DESPAIN, A. M. 1995. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'95)*, Dana Pt., CA, Apr. 1995, 63–68.
- YANG, J. AND GUPTA, R. 2002. Energy efficient frequent value data cache design. In *Proceedings of the ACM/IEEE 35th International Symposium on Microarchitecture (MICRO'02)*, Istanbul, Turkey, Nov. 2002, 197–207.
- ZHANG, C., VAHID, F., AND NAJJAR, W. 2003. A highly configurable cache architecture for embedded systems. In *Proceedings of the ACM/IEEE 30th International Symposium on Computer Architecture (ISCA'03)*, San Diego, CA, June 2003, 136–146.
- ZHANG, W., HU J. S., DEGALAHAL, V., KANDEMIR, M., VIJAYKRISHNAN, N., AND IRWIN, M. J. 2002. Compiler-directed instruction cache leakage optimization. In *Proceedings of the ACM/IEEE 35th International Symposium on Microarchitecture (MICRO'02)*, Istanbul, Turkey, Nov. 2002, 208–218.
- ZHOU, H., TOBUREN, M., ROTENBERG, E., AND CONTE, T. 2001. Adaptive mode control: A static-power-efficient cache design. In *Proceedings of the IEEE 10th International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, Barcelona, Spain, Sept. 2001, 61–70.
- SYNOPSYS 2002. *Managing Power in Ultra Deep Submicron ASIC/IC Design*. Synopsys White Paper, May 2002.
- SIA 2001. Semiconductor Industry Association (SIA). International Technology Roadmap for Semiconductors 2001. <http://public.itrs.net/files/2001ITRS/>
- SPEC. 2000. <http://www.specbench.org/osg/cpu2000/>

Received November 2004; revised February 2005; accepted February 2005