J Supercomput (2011) 55: 28–50
DOI 10.1007/s11227-010-0396-0

# Leakage-efficient design of value predictors through state and non-state preserving techniques

**Juan M. Cebrián · Juan L. Aragón · José M. García · Stefanos Kaxiras**

Published online: 5 March 2010


**Abstract** In the last decade computer engineers have faced changes in the way microprocessors are designed. New microprocessors do not only need to be faster than the previous generation, but also be feasible in terms of energy consumption and thermal dissipation. Recently, a new challenge appeared for computer engineers, the static power consumption. As process technology advances toward deep submicron, the static power component becomes a serious problem, especially for large on-chip array structures such as caches or prediction tables, and it must be taken into consideration. We can fight to reduce leakage power in two different ways: we can switch off the structure, reducing its leakage to zero but losing its contents (non-state preserving techniques), or we can lower its voltage (state preserving techniques), obtaining less savings but being able to restore the state of the structure in a reasonable time.

Data dependences are one of the key factors that limit performance in modern microprocessors. Value Prediction (VP) is a paradigm that exploits value locality in order to predict the output of an instruction, overcoming data dependences. The more accurate the predictor, the more performance is obtained, at the expense of becoming a potential source of power consumption and a thermal hot spot.

In this work we propose a leakage-efficient design of traditional Value Predictors (Stride, FCM, and DFCM) based on the fact that many VP entries remain unused during long periods of time before being eventually evicted. By applying both state and

J.M. Cebrián (✉) · J.L. Aragón · J.M. García
Dept. of Computer Engineering, University of Murcia, Murcia 30100, Spain
e-mail: jcebrian@ditec.um.es

J.L. Aragón
e-mail: jlaragon@ditec.um.es

J.M. García
e-mail: jmgarcia@ditec.um.es

S. Kaxiras
Dept. of Electrical and Computer Engineering, University of Patras, Rio, 26500 Patras, Greece
e-mail: kaxiras@ee.upatras.gr

non-state preserving techniques, the unused entries are disabled obtaining substantial leakage energy reductions (50–80% depending on the configuration and predictor type).

**Keywords** Power consumption · Value prediction · Low power · Leakage · Static power

## 1 Introduction

Nowadays, when we think about the design of embedded microprocessors we always think in terms of energy consumption and power dissipation, especially in the case of battery-operated devices. This trend is also moving to the high performance domain where operating costs and thermal constrains are becoming a serious problem. There are two sources of power dissipation: *dynamic* and *static* power (power dissipated regardless of activity, even when transistors are not switching). For several generations, static power (leakage) has been just a small fraction of the overall power consumption in microprocessors, and it was not considered a major concern [13, 14]. The continued CMOS scaling allows a lower supply voltage which has the positive effect of reducing the dynamic power component ($P_d \propto V_{dd}^2$). However, using smaller geometries, along with very small threshold voltages, has the additional effect of increasing leakage loss exponentially, which leads to static power beginning to dominate the overall power consumption as process technology drops below 65 nm [1, 7].

Several proposals can be found in literature at both circuit and architecture levels to reduce the leakage power. Those proposals can be classified into *State* and *Non-state preserving* techniques. *Non-state preserving* techniques focus on reducing the leakage power by switching off unused portions of large array structures. Cache Decay [12] selectively turns individual data cache lines off if they have not been used for a long time, reducing leakage energy at the expense of losing the contents of the cache line. *State preserving* techniques such as *drowsy* [6] try to reduce leakage without losing the contents of the structure by using different supply voltages according to the state of the structure's entry. The leakage reduction is not as high as in decay scheme, but restoring the structure's contents is more power-efficient in drowsy than in decay.

On the other hand, data dependences are one of the key factors that limit performance in current high-performance superscalar microprocessors. Recent works suggested that Value Prediction (VP) can overcome the limits imposed by data dependences [8, 9, 17, 19]. Value prediction is a technique that predicts the output of an instruction as soon as it is fetched allowing subsequent instructions that depend on that result to execute using the predicted value. These instructions must be validated once the actual values are computed. More recently, VP has also been successfully used to perform early load retirements in high performance processors [15]. However, the use of value prediction techniques has not been widespread, despite the speedup provided (15% on average as reported in [3]), mainly due to complexity-delay issues. Note that unlike other prediction structures, such as branch predictors, the access

time in VPs is not crucial. Firstly, the predicted value is not needed until the instruction has reached its issue stage, and secondly, current high performance processors typically implement deeper pipelines (14 stages or more) which effectively hide the VP latency. In addition, the use of VP structures incurs additional dynamic and static power dissipation. The continuous access to the prediction tables in almost each clock cycle may result in a thermal hot spot, increasing the leakage power of the structure, as it also happens in caches and branch predictors. In modern high performance processors, due to high operating temperatures, it is crucial to reduce leakage in every possible structure. Although the VP is a small structure compared to an L2 cache, if we let it overheat (likely, as it is accessed frequently and resides quite close to the core) without any precaution to regulate its leakage, the negative effects can be quite serious. Small hot structures can leak more than larger but cooler ones and we cannot afford to allow leakage even in the smallest structures.

In previous papers we proposed the use of non-state preserving techniques for reducing leakage power in value predictors, including *Static Value Prediction Decay (SVPD)* [4] and *Adaptive Value Prediction Decay (AVPD)* [5]. These techniques either statically or dynamically locate VP entries that have not been accessed for a noticeable amount of time and switch them off to prevent leakage.

The major contributions we present in this extended work are the following:

- A novel power-performance and leakage-efficient *drowsy* approach for *Value Predictors* is introduced. This approach will locate unused entries and lower their supply voltage, retaining the data but requiring five additional cycles to access it again.
- A new Sect. 4.3 that introduces the drowsy mechanism and different ways to implement it has been added to better understand the benefits of the novel proposal.
- A detailed analysis of the experimental results obtained by the novel drowsy mechanism for reducing leakage in value predictors has been added (a new Sect. 5.2.2).
- Finally, Sect. 5.2.3 has been extended in order to provide a comparative analysis of the best drowsy mechanism against previously proposed non-state preserving mechanisms for reducing leakage in value predictors.

The *Adaptive Value Prediction Decay* (*AVPD*) approach is needed for two reasons. First, adapting the decay interval individually for individual VP entries (as opposed to cache lines) would represent significant overhead and, thus, we consider it impractical. Second, VPs are non-tagged structures and, therefore, it is infeasible to track the ideal miss rate *vs.* the induced miss rate. *AVPD* uses a global run-time decay interval, requiring no additional hardware *per entry.* To adapt this global decay interval without tags, *AVPD* uses a time-based approach to judge whether the current decay interval causes an inordinate number of entries to be prematurely shut off.

The rest of the paper is organized as follows. Section 2 provides some background and reviews some related work. Section 3 analyzes the utilization of the value prediction tables. The proposed schemes for reducing leakage power in value predictors are described in Sect. 4. Section 5 shows the experimental methodology and the leakage energy savings obtained. Finally, Sect. 6 summarizes the main conclusions of the work.

## 2 Related work

### 2.1 Approaches for reducing leakage power

Fighting to reduce leakage has been an important topic in microprocessor development in the last few years. The techniques to deal with leakage have been categorized into *non-state preserving* and *state preserving* [2, 16, 20, 23].

In the non-state preserving scenario, Powell et al. [18] proposed *gated-*$V_{DD}$ as a procedure to restrain leakage power by gating off the supply voltage of cells. This non-state preserving technique, known as *decay*, reduces the leakage power drastically at the expense of losing the cell's contents. Decay techniques must be applied very carefully since the information loss can result in an increase of the dynamic power. Kaxiras et al. [12] successfully applied decay techniques to individual cache lines in order to reduce leakage in cache structures (67% of static power consumption can be saved with minimal performance loss). This technique has also been applied to conditional branch predictors and BTB structures [10, 11].

Conversely, the state preserving scenario is lead by *drowsy* techniques which try to reduce leakage without losing the cell's contents. Drowsy caches [6] use different supply voltages according to the state of each cache line. The lines in drowsy mode use a low-voltage level that allows a leakage reduction while retaining the data, but requiring a high voltage level to access it again. Waking up from the drowsy state is similar to a pseudo-cache miss incurring in some penalty cycles (about 7 cycles) according to [16]. Of course, the leakage power savings of this mechanism are lower than the decay ones, but the additional dynamic power consumption due to the information loss is also decreased. Flautner et al. [6] showed that a drowsy cache, which is putting to sleep all cache blocks periodically, achieves 54% leakage power savings with negligible performance degradation (about 1%). The authors in [16] confronted the state and non-state preserving techniques for caches and showed that, for a fast L2 cache (5–8 latency cycles), decay techniques are superior in terms of both performance degradation and energy savings compared to drowsy ones.

As an example of an adaptive decay mechanism suited for caches, Zhou et al. [22] proposed an adaptive time-based mechanism that dynamically disables cache lines in order to reduce leakage power dissipation. The mechanism takes advantage of the cache tag array, which is never switched off, to track if there are many induced cache misses in order to adapt the length of the decay interval accordingly.

Finally, quasi-static, four-transistor (4T) memory cells are an alternative to traditional decay techniques. These cells are approximately as fast as 6T SRAM cells, but do not have connections to the supply voltage ($V_{SS}$). Conversely, the 4T cells are charged upon each access whether read or write, and slowly leak the charge over time until, eventually, the values stored are lost. Authors in [11] applied decay techniques to branch predictors by using 4T cells, removing some of the drawbacks of gated-$V_{DD}$ transistors, since any access to a 4T cell automatically reactivates it, whereas reactivating a 6T cell from the "sleep" mode is somehow more complex, requiring extra hardware involved in gating the supply voltage.

### 2.2 Value prediction overview

This section introduces the most common value predictors and their particularities. The *last value predictor* was introduced by Lipasti et al. [17]. This is the most basic prediction mechanism and, basically, it assumes that the next value produced by an instruction will be the same as the previous one. A generalization of the last value predictor leads to the *stride value predictor* (STP). Introduced by Gabbay et al. [8], STP uses the last value produced by an instruction plus a stride pattern. In a stride pattern, the difference between two consecutive values is a constant. The next predicted value is computed by adding the last value to the stride.

The *finite context method* value predictor (FCM), introduced by Sazeides et al. [19], uses the history of recent values, called the *context*, to determine the next value. This is implemented by using two-level prediction tables. The first level stores the recent history of the instructions outputs (VHT). The second level stores, for each possible context, the value which is most likely to follow that pattern (VPT). The value is predicted by using the program counter to access the VHT table and, according to the context hash function, the VPT table is accessed to get the predicted value.

Finally, the *differential finite context method* value predictor (DFCM) introduced by Goeman et al. [9], joins the two previous predictors in one structure. DFCM works like FCM (two-level prediction tables), but it stores as the context the *differences* between the outputs instead of the outputs themselves, plus the last output of the instruction. In this way, DFCM can predict stride patterns using less storage space than FCM by adding the last value to the stride associated with the context. For non-stride patterns, DFCM works just like the FCM predictor.

## 3 Problem overview: generational behavior in value prediction structures

As introduced before, there are two sources of power dissipation in all processor structures, dynamic and static power. In value prediction, the dynamic component is produced by the repeated capacitance charge and discharge on the transistor gate outputs and strongly depends on the utilization of the VP tables: the more use the more consumption. The output values of instructions can be predicted at different demanding levels: the most aggressive utilization predicts the output for all instructions traversing the pipeline (this is the worst-case scenario for power reduction techniques, and it will be the one used in the rest of this paper). Other approaches restrict the use of the value predictor to just a fraction of instructions such as long-latency instructions, load instructions that miss in the data caches, instructions that belong to the critical path, or just to predict the effective address for memory disambiguation. Therefore, restricting the VP utilization to just a fraction of selected instructions effectively reduces the dynamic power component of this structure. However, the static power component is still there: the VP structure leaks regardless of activity, especially through *subthreshold* leakage currents and *gate* leakage currents that flow even when transistors are nominally off, with increasing leakage loss for finer process technologies. For this reason, our work is focused on reducing the VP structure's static power component.

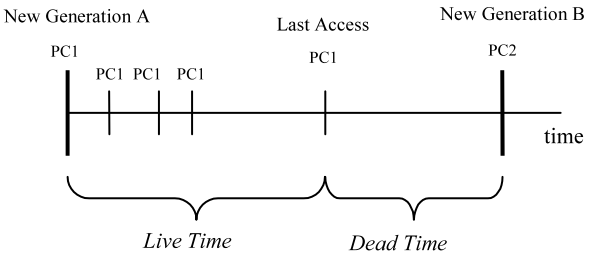


**Fig. 1** Temporal behavior of a value predictor entry

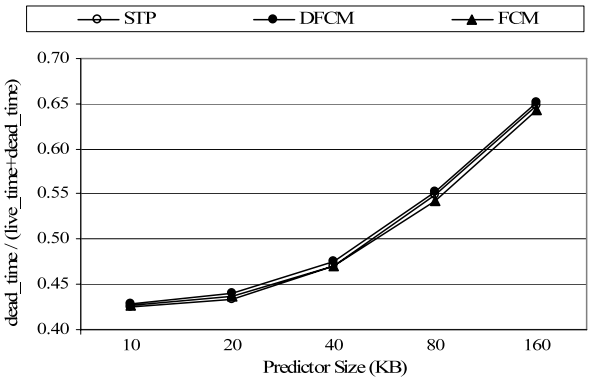


**Fig. 2** Fraction of time VP entries spend in *dead* state (SpecInt2000)

Value predictors share many similarities with caches as they are both array structures. As happened in [12], looking at the value predictor entries' behavior we can divide the stream of accesses into generations. A generation is defined as the period of time an entry is accessed by the same instruction. Entries have an initial usage time (known as live time) followed by a period of no-utilization (known as dead time) before they are accessed by a different instruction, as shown in Fig. 1. An entry's *live time* will be the period of time the entry is accessed by the same PC and its *dead time* will be the period of time between the last access with a specific PC and the first access with a new one.

In order to evaluate the generational behavior and the utilization of the VP entries, Fig. 2 shows the fraction of time each entry remains in the *dead* state[1] for the whole SpecInt2000 benchmark suite as a function of VP size. This way we can determine if disabling all the entries during the dead times will produce enough savings to justify the mechanism. It can be observed that the three evaluated value predictors—Stride, FCM and DFCM—present a similar utilization regardless of their size. For sizes around 20 KB, the average fraction of dead time is 43%, and for predictor sizes around 40 KB, the average fraction of time the entries spend in their *dead* state is 47%. Therefore, if we were able to take advantage of these *dead times* by detecting them and shutting the entries off, we could reduce the leakage energy of the value predictor structure by one half on average. However, it is important to note that this is not an upper bound on the leakage energy savings that could be achieved by decaying

[1] This fraction of time can be measured as the ratio *total dead time*/(*total live time* + *total dead time*).

VP entries. Long periods of inactive *live time* could also be detected to early shut the entry off in order to obtain further leakage savings, at the expense of slightly reducing the VP accuracy and processor performance, as we will show in the following sections.

## 4 Techniques for reducing leakage in value predictors

### 4.1 Non-state preserving scheme: static decay

We will begin introducing the non-state preserving techniques. *Static decay* is a technique that tries to locate unused VP entries in order to switch them off [5]. To decide if an entry is *unused*, a decay interval is established, i.e. the number of cycles we should wait before shutting an entry off if there are not any instructions accessing it. If we choose short decay intervals we will increase the energy savings, but we can degrade processor performance, as we can disable entries during their live time, losing their contents. On the other hand, choosing long decay intervals will decrease the chances of disabling entries during live time, but we will lose some potential energy savings from dead times. Therefore, we need to track the accesses to each VP entry in order to detect if a particular entry is accessed very frequently or, conversely, the entry has been unused for a long period of time, probably entering into a *dead* state. For the static decay scheme it is crucial to explore a wide range of *decay intervals* to precisely detect the dead states while, at the same time, not degrading the VP accuracy and, therefore, the speedup provided. Ideally, the best static decay interval is the one that minimizes the performance impact of prematurely disabling a VP entry.

The static decay mechanism will be implemented by the means of a hierarchical counter composed of a global decay counter and a two-bit saturated gray-code counter on each value predictor entry.[2] The local counters will only be incremented when the global counter reaches zero. Any access to an entry will reset its local counter. When a local counter reaches its upper limit it means that the entry has been unused for a decay interval, and can be shut off, reducing its leakage power consumption to almost zero. As cited in Sect. 1, the access time to the VP structure is not crucial but, in our study, we will use a moderate access time of 5 cycles.

To prevent VP entries from leaking we will use *gated-*$V_{DD}$ transistors [18]. These "sleep" transistors are inserted between the ground (or supply) and the cells of each VP entry, which reduces their leakage in several orders of magnitude and can be considered negligible. An alternative to using *gated-*$V_{DD}$ transistors consists of using quasi-static 4T transistors in the VP array, although similar leakage savings would be expected [11].

The length of the decay interval is controlled by the global decay counter. If we set the global decay counter to a low value, the VP entries may be disabled prematurely and leakage will be reduced drastically, but so will the hit rate of the predictor. On

[2] Using a hierarchical counter is more power-efficient since it allows accessing the local counters at a much coarser level. Accessing the local counters each cycle would be prohibitive because of the power overhead.

the other hand, if we increase the global counter too much (a long decay interval), the leakage energy savings will not be as high as their potential.

Regarding the utilization of VPs, throughout the paper we are predicting the output values for *all* instructions traversing the pipeline. However, it is important to note that this aggressive prediction scheme does not benefit a decay mechanism, either static or adaptive, since it is based on locating unused predictor entries. The more demanding use of the VP structure, the less opportunities to detect unused VP entries and the less leakage energy savings obtained from a decaying mechanism.

There are several overheads that must be considered when performing the leakage energy savings evaluation. The first component overhead takes into account the extra dynamic and static power that results from the additional hardware (a global decay interval counter as well as the two-bit local counters[3] per VP entry [5]). The second component overhead comes from the induced VP misses (when a VP entry is prematurely disabled) that increase execution time. These extra cycles that the program is running will also lead to additional static and dynamic power dissipation. Note that this second overhead is highly destructive since each extra cycle accounts for the overall processor dynamic and static power and can easily cancel whatever VP leakage energy savings provided by the decay scheme.

### 4.2 Non-state preserving scheme: adaptive value prediction decay (*AVPD*)

*Adaptive Value Prediction Decay* (*AVPD*) is, like *Static Decay*, a time-based mechanism that analyzes the VP tables to detect unused entries. If an entry is unused for a long period of time, it probably means that it has entered in a dead state, and we should proceed to turn it off. As we will see in Sect. 5.2.1, the decay interval is dependant on the application running in the processor or even on the code section being executed. During program execution there are sections of code where the VP usually hits or fails its predictions (correct and wrong predictions appear clustered depending on the program phase). We can also find program sections where the number of VP entries being accessed is abnormally low, and even identify instructions whose optimal decay interval is different from others. Therefore, if we are able to dynamically adapt the decay interval to the program needs, higher leakage energy savings could be obtained compared to statically setting the decay interval.

The *AVPD* mechanism will use basically the same implementation as the static approach. The mechanism uses a hierarchical counter composed of a *global counter* and a two-bit saturated gray-code counter for each individual value predictor entry (*local counters*) to implement the decay interval. The *AVPD* mechanism considers that each VP entry can be in one of the following three states, as shown in Fig. 3: *enabled* (both data and the local counter are enabled), *partially disabled* (data is shut off but the local counter is enabled) or *disabled* (both data and the local counter are shut off).

*AVPD* uses two additional global counters that account for: (a) the number of *partially disabled* entries (entries that change from the *enable* state to the *partially*

[3]The dynamic and static power overhead of all 2-bit local counters has been measured to be less than 2% of the total VP structure.

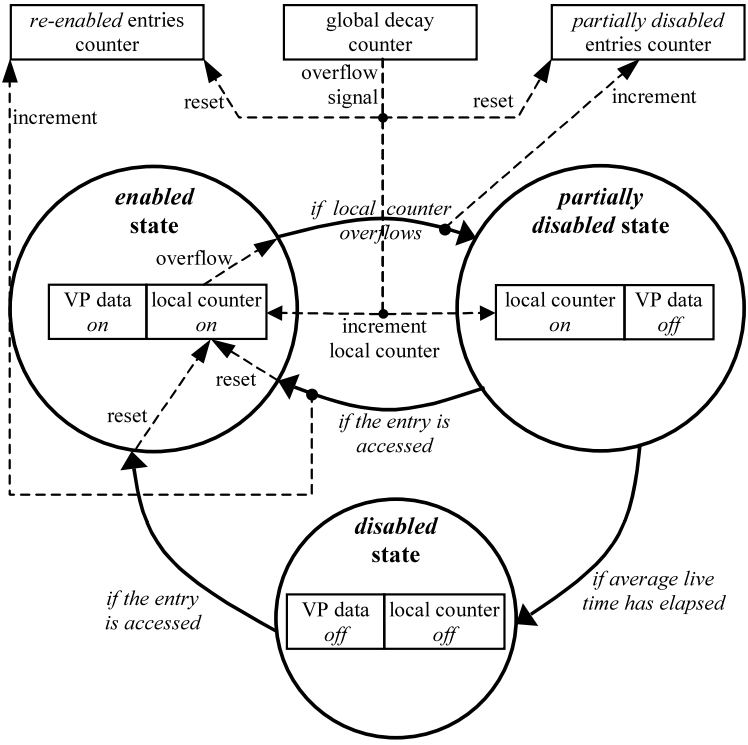


**Fig. 3** *AVPD* mechanism

*disabled* state) within the previous decay interval; and (b) the number of *re-enabled* entries (entries that change from the *partially disabled* state to the *enabled* state) within the current decay interval. After a number of cycles equal to the average live time,[4] a *reactivation ratio* is calculated as the number of *re-enabled* entries over the number of *partially disabled* entries.

As we cannot identify the instruction accessing the value predictor (because it is implemented as a non-tagged table), it is not possible to determine if there is a real generational change or if we disabled the entry during its live time. Therefore, we will use a time-based approach. We will add an intermediate state between the enabled and disabled states and account, during a short period of time, if there are many entries that are re-enabled (come back from the intermediate state to enable state). If that happens, the decay interval is too short. If there are not many re-enabled entries, we can try to lower the decay interval to increase savings. A positive effect of *AVPD* compared to the original cache decay mechanism is that prematurely disabling a VP entry is not as harmful as disabling a cache line: losing the contents of the cache line *always* leads to an extra access to L2 cache or memory to retrieve the lost

[4]The static decay experiments showed that the *average live time* is around 400 cycles for the three evaluated VPs.

information incurring in extra execution cycles. However, losing the contents of a VP entry might result—or not—in a value miss prediction on the next access to that entry, but this is exactly what would happen if we had a real generation change (which is a very common situation and one of the major limitations in traditional non-tagged VPs, where the huge number of destructive interferences dramatically shortens the generational replacement).

In addition, *AVPD* uses two predefined threshold values (*increasing threshold* and *decreasing threshold*) in order to determine whether the length of the current decay interval is correct: i.e., if the current decay interval makes VP entries to decay during their *live time* (prematurely) or during their *dead time* (as expected). Therefore, if the *reactivation ratio* is higher than the *increasing threshold*, the current decay window is too short and it is doubled since there are many entries being disabled prematurely. On the other hand, if the *reactivation ratio* is lower than the *decreasing threshold*, the current decay window is too long and it is halved since we are shutting entries off too late, losing opportunities to reduce the VP leakage. In order to make the *AVPD* mechanism easier to implement, we will use power-of-two decay intervals. VP entries are shut off, preventing them from leaking, by using *gated-*$V_{\mathrm{DD}}$ transistors [18].

The *AVPD* mechanism works as follows (see Fig. 3): in each cycle, the global decay counter is incremented by one and, when it overflows, the local counters of all VP entries in either *enabled* or *partially disabled* states are incremented. However, an access to any VP entry will result in an immediate reset of its local counter. In addition:

- For those entries in the *enabled* state (both VP data and the local counter are enabled): if the entry remains unused for a long time, its local counter will eventually overflow and the entry will change to the *partially disabled* state. The number of *partially disabled* entries is incremented.
- For those entries in the *partially disabled* state (VP data is shut off whereas the local counter is enabled): if the entry is not accessed within the average live time, it will be changed to the *disabled* state and the local counter will be also shut off. However, an access to a *partially disabled* entry will change it to the *enabled* state, increasing the number of *re-enabled* entries.
- For those entries in the *disabled* state (both VP data and the local counter are shut off): any access to the entry will change it to the *enabled* state.

Regarding the predefined values used for the *increasing* and *decreasing* thresholds, it is important to note that setting the *decreasing threshold* to small values will make *AVPD* sure that there are few *re-enabled entries* before lowering the decay interval, resulting in a more conservative policy. On the other hand, setting the *decreasing threshold* to high values will make *AVPD* to decrease the decay interval more frequently, resulting in a more aggressive policy. Analogously, setting the *increasing threshold* to small values means that *AVPD* will increase the decay interval even if there are few *re-enabled entries*; whereas setting the *increasing threshold* to high values will make *AVPD* to wait until having a great fraction of *reactivations* before increasing the decay interval. In Sect. 5.2.3 we evaluate the leakage-efficiency of the *AVPD* mechanism for different *increasing* and *decreasing thresholds*.

We can split the power overhead of the *AVPD* mechanism into three main components. The first component is associated with the dynamic and static power derived

from the two-bit local counters inserted into every predictor entry (same overhead as for the static decay scheme). The second component comes from the three global counters: one is part of the hierarchical decay interval counter (also appears in the static decay scheme) and the other two counters are specific of the adaptive decay scheme. The third component overhead is derived from the induced VP misses (when a VP entry is prematurely disabled) that increase program execution time. These extra cycles that the program is running will also lead to additional static and dynamic power dissipation.

It is important to note that *AVPD* is virtually not introducing additional power overhead nor complexity when compared to the *static* decay scheme (just the additional two global counters whose power overhead that has been conveniently modeled into the *AVPD* power model).

### 4.3 State preserving scheme: drowsy

The drowsy technique aims to reduce leakage power consumption while preserving the contents of a cell by switching between two working modes, low-power and high-power. While the cell is in low-power mode, the information is preserved, but it cannot be accessed. In order to access the cell again it must be reinstated into the high-power mode. A drowsy scheme can be configured according to the following parameters [6]:

- *Update window size*: specifies if the amount of cycles to wait before turning entries into drowsy mode can be varied.
- *Simple or No-access policy*: "simple" means that *all* entries are turned into drowsy mode after a number of cycles. "No-access" puts to drowsy mode only the entries that have not been accessed in a number of cycles.
- *Awake or drowsy tags*: put tags into drowsy mode or not (affects latency).

According to Flautner et al. [6], the *simple policy* with a window size of 8000 cycles comes very close to the behavior of the *no-access policy* with a window size of 2000. They choose a *simple policy* with a window size of 4000 cycles for their tests, as it reaches a reasonable compromise between simplicity of implementation, power savings and performance. In our case, it is not fair to compare the decay and drowsy approaches using the *simple policy* since the decay approach will delete all the information of the structure every decay interval (data loss will mean an access to L2 or memory in decay, but only a few cycles for drowsy to re-enable the entry). Instead, we will then use an *update window–no-access policy.*[5] In our case, drowsy will be used exactly as decay, at an entry level. The leakage power consumption of a transistor in drowsy mode is measured to be 15% of the original leakage (as estimated in [6]), something significant compared with the almost 0% of the decay scheme.

There are several ways of implementing these two power-level techniques (ABC-MTCMOS, DVS, etc.). The Dynamic Voltage Scaling (DVS) [6] mechanism allows structures to dynamically change their speed and voltage while operating, increasing their energy efficiency. If we refer to memory structures, it is possible to reduce

[5] The *awake/drowsy tags policy* cannot be evaluated because the evaluated VPs do not have tags.

their supply voltage while retaining the data. Scaling the voltage of the cell to 0.5 times the supply voltage can maintain the cell's state. This reduction of the operating voltage allows us to reduce the leakage currents, and thus the leakage power of the memory cell. DVS must be controlled by a *voltage scheduler* to dynamically adjust the processor speed and voltage during execution. Voltage schedulers analyze the state and context of the system in order to predict future workload of the processor, increasing the complexity of scheduling.

Auto-Backgate-Controlled MT-CMOS is a mechanism that dynamically increases the transistors' threshold voltages when going to "sleep" mode by raising the transistors' body voltage. This higher $V_T$ reduces the leakage current without losing the cells contents when going to "sleep" mode but offsets the total leakage power savings. This technique also requires high energy to switch between states increasing the time needed to transition. DVS is faster, easier to implement, and obtains more power reduction than ABC but depends on the process technology and is more sensible to SEU noise.

## 5 Experimental results

### 5.1 Simulation methodology

To evaluate the energy-efficiency of *SVPD*, *AVPD* and *drowsy* techniques we have used the SpecInt2000 benchmark suite. All benchmarks were compiled with maximum optimizations (-O4-fast) by the Compaq Alpha compiler and they were run using a modified version of *HotLeakage* power-performance simulator [21] that includes the dynamic and static power models for the evaluated Value Predictors (Stride, FCM and DFCM), as well as the power overhead associated with *SVPD*, *AVPD* and *drowsy* techniques. All benchmarks were run to completion using the reduced input data set (test). The VP access latency has been set to 5 cycles for the three evaluated VPs.

Table 1 shows the configuration of the simulated architecture. The STP value predictor has 76 bits per entry: value (64) + stride (8) + confidence (2) + decay counter (2). FCM has a variable number of bits per entry on the first level table: first level decay counter (2) + history bits for second level table (variable) and 68 bits per entry on the second level table: value (64) + confidence (2) + second level decay counter (2). DFCM has 68 bits per entry + a variable amount of bits on the first level table: value (64) + first level decay counter (2) + confidence bits (2) + history bits for second level table (variable) and 12 bits on the second level table: confidence (2) + decay counter (2) + stride (8). The leakage-related parameters have been taken from the Alpha 21264 processor provided with the *HotLeakage* simulator suite, using a process technology of 70 nanometers.

### 5.2 Leakage-efficiency of state and non-state preserving mechanisms

As discussed previously, predicting the output values for *all* instructions traversing the pipeline does not benefit any of the proposed mechanisms, either static or adaptive, since they are based on locating unused predictor entries. The more demanding use of the VP structures, the less opportunities to detect unused VP entries.

**Table 1** Processor configuration

| | |
|---|---|
| Processor core | |
| Process Technology: | 70 nanometers |
| Frequency: | 3200 MHz |
| Instruction Window | 128 RUU, 64 LSQ |
| Decode Width: | 8 inst/cycle |
| Issue Width: | 8 inst/cycle |
| Functional Units: | 8 Int Alu; 2 Int Mult |
| | 8 FP Alu; 2 FP Mult |
| Memory hierarchy | |
| L1 I-cache: | 64 KB, 2-way |
| L1 D-cache: | 64 KB, 2-way |
| L2 cache: | 2 MB, 4-way, unified |
| Memory: | 2 memports |
| Other information | |
| Branch Predictor: | 16 bit Gshare |
| Branch & Value Pred.: | 2 ports |

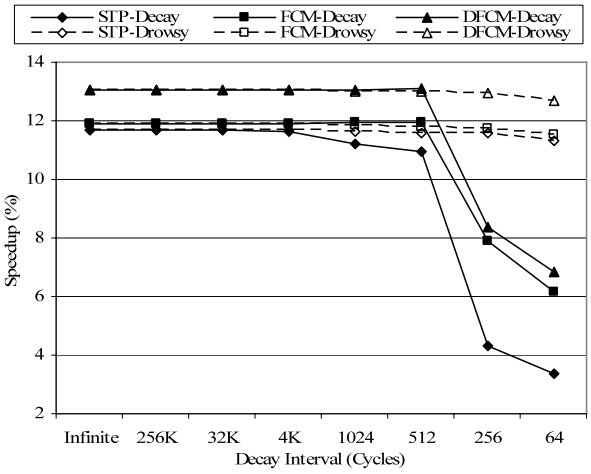


**Fig. 4** Average speedup for the *static decay* and *drowsy* schemes for 10 KB VPs (SpecInt2000)

In order to better understand the effects of prematurely deactivating a VP entry, Fig. 4 shows the differences in speedup for both static decay and drowsy with decay intervals from 256 K cycles to just 64 cycles. We must never forget that traditional value prediction (with no power-saving technique) can provide significant speedups (13% for a 10 KB DFCM).

Looking into the performance degradation caused by *static decaying*, we can notice that for FCM and DFCM predictors there is no IPC degradation until 256-cycle decay intervals. For the STP predictor, there is a slight but negligible IPC degradation (less than 1%) for 1024- and 512-cycle decay intervals. As happened before, for 256-cycle (and smaller) intervals the performance degradation is not tolerable.

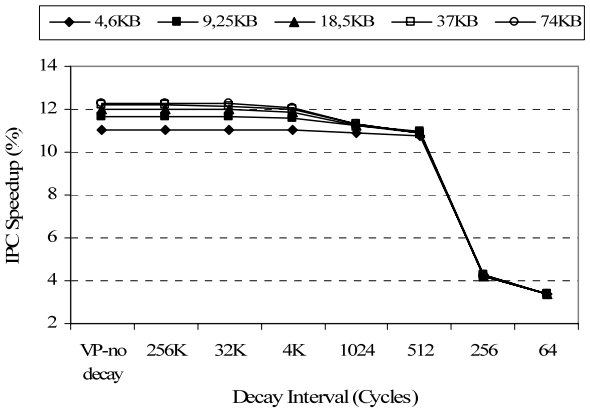


**Fig. 5** STP value predictor performance degradation (SpecInt2000)

We measured the VP entries' average live time to be about 400 cycles, so, when the decay interval exceeds that critical point, the speedup provided falls apart.

On the other hand, the drowsy technique "normalizes" the optimal decay interval for all predictors to 256 cycles, behind the average live time, but close enough to it to maintain the speedups provided by the value predictor. Note that drowsy does not lose the entries' contents, which makes drowsy not to fall in performance as quickly as decay does. But this does not make drowsy more energy-efficient, as performance is only an element of the energy metric, and the power reduction provided by drowsy is lower than decay's.

### *5.2.1 Static decay for VPs*

In this section we perform an evaluation on the energy-efficiency of the static decay scheme for value predictors for different predictor sizes (sizes are not power-of-two numbers because of the extra 2-bit counters per entry) and for several decay interval windows: 64, 256, 512, 1024, 4 K, 32 K and 256 K cycles.

For each evaluated VP we report the IPC degradation as we reduce the decay interval and the corresponding leakage energy[6] savings for the VP structure. Overall leakage energy savings are not presented due to *HotLeakage* limitations that only provide static power models for regular array structures such as caches, predictors, and the register file.

Figure 5 shows the performance degradation of the STP value predictor for different sizes (SpecInt2000 average). Looking at the performance degradation caused by *static decay* we can notice that it is degraded as expected, due to the data loss of prematurely deactivating VP entries that still are in a *live state*. In particular, there is a slight IPC degradation (around 1%) for 1024- and 512-cycle decay intervals. However, due to early deactivation of entries and the induced extra execution cycles for 256-cycle and smaller decay intervals, the performance loss is not tolerable.

[6]Recall that the performance degradation is also included in the energy metrics (leakage energy = leakage power * delay).

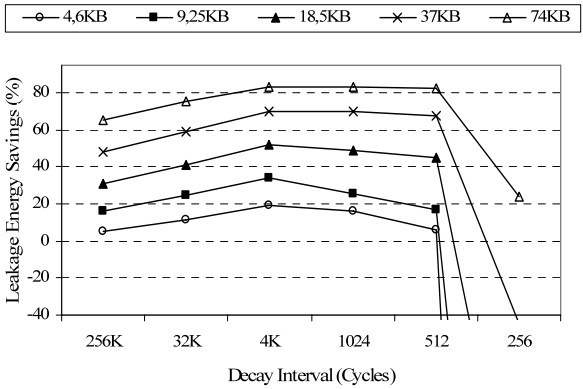


**Fig. 6** STP value predictor leakage energy savings (SpecInt2000)

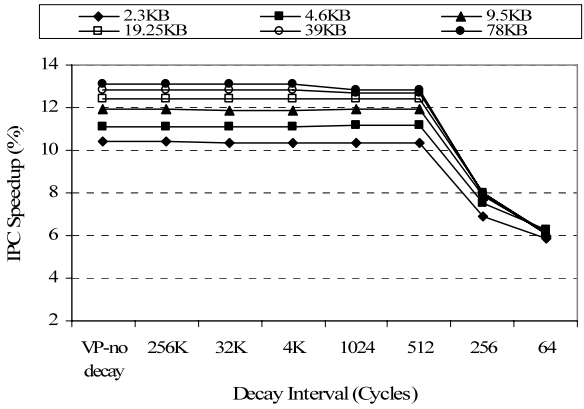


**Fig. 7** FCM value predictor performance degradation (SpecInt2000)

Figure 6 shows the average leakage energy savings of the STP predictor. As expected from the IPC degradation, for very small decay intervals (64 and 256 cycles), the early deactivation of entries results in no leakage energy savings at all due to the induced extra execution cycles that completely cancel whatever leakage power savings provided by the decay mechanism. However, for 1024 cycles and, particularly, for a 4 K-cycle decay interval, the proposed VP decay approach obtains 52% average leakage energy savings when considering a medium size (20 KB) predictor.

Figures 7 and 8 show the performance degradation and the average leakage energy savings of the FCM value predictor. Since FCM is a two-level predictor (with the relevant data stored in the second level table), we will disable both levels independently. We can notice a behavior similar to the STP predictor for the very small decay intervals (64 and 256 cycles), again with negative leakage energy savings due to the early deactivation of entries and the induced extra execution cycles. On the other hand, for very big decay intervals (32 K and 256 K cycles), the overhead is almost zero, but we obtain very small leakage energy savings (Fig. 8) since there are almost no deactivations of VP entries. However, as we reduce the decay interval length, there is an increase in leakage savings with a maximum in the 512-cycle interval. For this

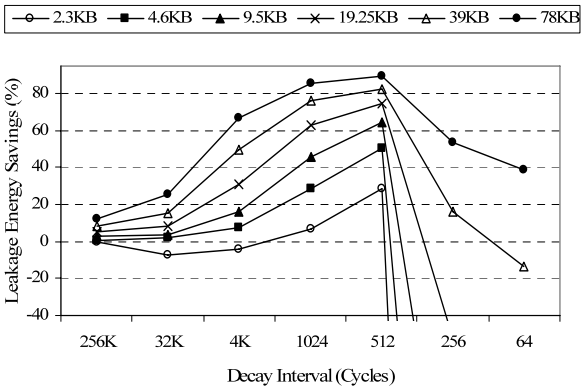


**Fig. 8** FCM value predictor leakage energy savings (SpecInt2000)

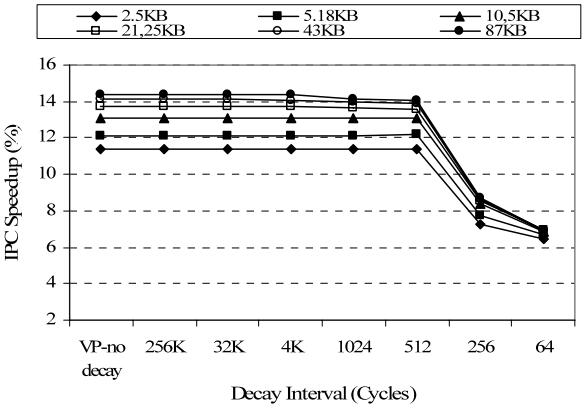


**Fig. 9** DFCM value predictor performance degradation (SpecInt2000)

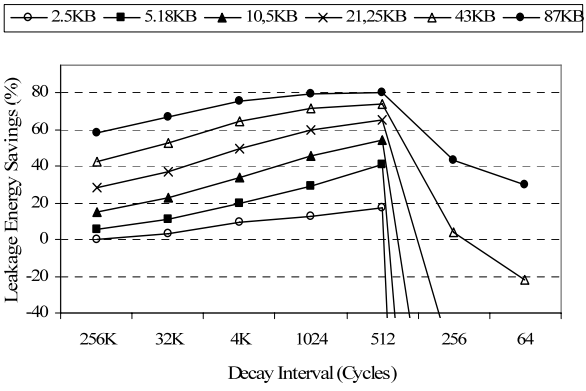


**Fig. 10** DFCM value predictor leakage energy savings (SpecInt2000)

decay interval, a 20 KB FCM predictor obtains average leakage energy savings of 75%, showing the benefits of *Value Prediction Decay*.

Figures 9 and 10 show the performance degradation and the average leakage energy savings of the DFCM value predictor. We can notice that the DFCM predictor

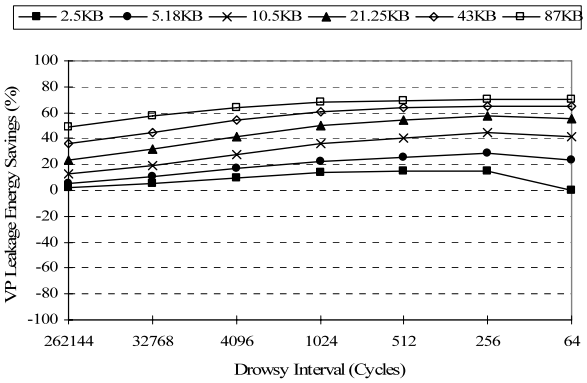


**Fig. 11** Static drowsy scheme for the DFCM value predictor (SpecInt2000)

behaves very close to FCM. However, for big decay intervals, DFCM obtains better energy savings (Fig. 10) due to a positive side effect when shutting entries off, which results in a reduction of destructive interferences. Imagine that we have entries in the first level table with different predictions that point to different second level table entries. If the instruction that these entries are predicting changes, there is "trash" on the tables from the previous prediction that will interfere during some cycles with the new information until the predictor gets stable, whereas if we decay, all entries will reset, so they will always behave the same. Again, as we reduce the decay interval length, there is an increase in leakage energy savings with a maximum in the 512-cycle interval. In this case, for a predictor size of around 20 KB we obtain average leakage energy savings of 65%. For both FCM and DFCM predictors, the best energy savings are obtained for a decay interval within the 512-cycle range, unlike data caches where the best decay intervals are within the 8 K-cycle range [12].

### *5.2.2 Drowsy for VPs*

This section shows the results of the drowsy technique applied to DFCM, STP and FCM value predictors per entry. In order to access an entry in drowsy state, we add 5 additional latency cycles to the original 5 latency cycles of the VP structure.

Figure 11 shows the average energy savings for a DFCM predictor using the drowsy technique applied statically. Compared to Fig. 10, the best drowsy configuration obtains less leakage energy savings than the best static decay. Despite the fact that drowsy shortens the best decay interval from 512 to 256 cycles, the performance degradation and the additional overhead (15% extra consumption) makes drowsy to lose against static decay for DFCM. For a predictor size of around 10 KB, drowsy obtains average leakage energy savings of 44% while for a size around 20 KB, the average leakage energy savings are 57%.

In the STP predictor case (Figs. 12 and 6), drowsy behaves better than decay. As we can see in Fig. 4, the speedup degradation in the static decay for STP predictor begins in 4 K cycles but, for drowsy, it starts at 256 cycles. This reduction of the decay interval and the low IPC degradation makes energy savings greater in drowsy than in decay.

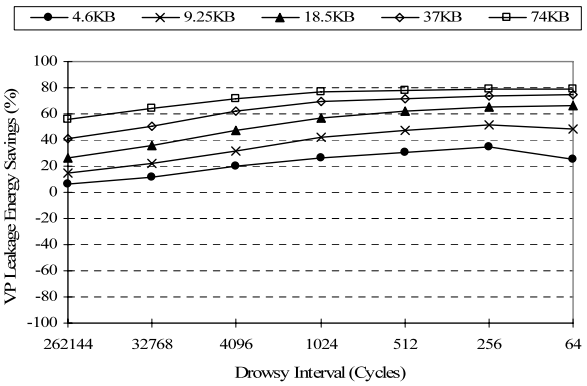


**Fig. 12** Static drowsy scheme for the STP value predictor (SpecInt2000)

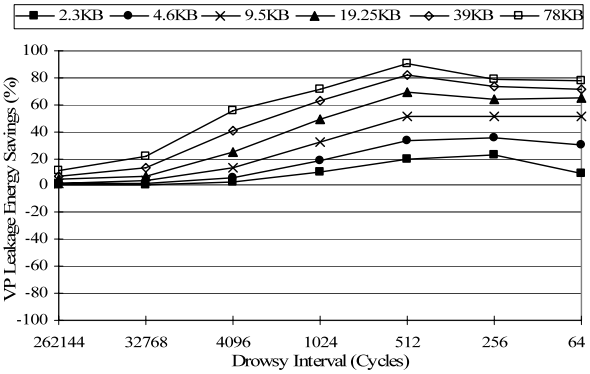


**Fig. 13** Static drowsy scheme for the FCM value predictor (SpecInt2000)

As happened previously with the DFCM predictor, the FCM predictor behaves better for the static decay scheme than for drowsy (Figs. 8 and 13). As we can see in Fig. 13, there best decay interval depends on the predictor's size. For small predictor sizes, the best decay interval is 256 cycles, and, for anything greater than 10 KB, the best decay interval is 512 cycles. This is due to extra penalty cycles in the FCM predictor for sizes bigger than 10 KB. For a predictor size of around 10 KB, the drowsy scheme obtains average leakage energy savings of 51%, while for a size about 20 KB it obtains 64% energy savings.

### *5.2.3 Adaptive decay for VPs*

Using a static decay interval means that we must choose that decay interval carefully in order to maximize the leakage savings. Figure 14 shows the leakage energy savings, per benchmark, for a 10 KB DFCM value predictor using the *static decay* technique. In some cases, the best static decay interval may differ between applications (as it can be seen in Fig. 14, where the best static decay interval is 4 K cycles for some benchmarks—mcf and gap—and 512 cycles for the rest). However, even when using profiling techniques in order to determine the best decay interval per application, there is no guarantee that the best leakage savings are obtained, since the *static decay* approach cannot capture variations within an application. This second

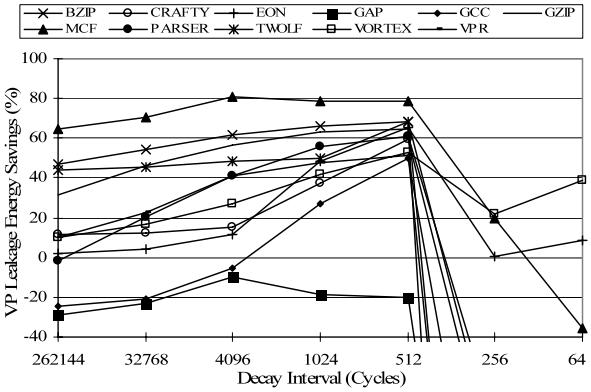


**Fig. 14** Static decay scheme for a 10 KB DFCM value predictor

effect is important in the case of value prediction structures since correct and wrong predictions appear clustered depending on the program phase. Therefore, an *adaptive decay* scheme can dynamically choose decay intervals at run-time to more precisely match the generational behavior of prediction tables' entries.

This section presents the leakage-efficiency evaluation of the proposed *AVPD* mechanism for the Stride, FCM and DFCM predictors, compared to the best configuration of both static decay and drowsy schemes. Each figure shows the average VP *leakage energy savings* for some representative configurations of the adaptive mechanism, as well as the best static decay (512-cycle decay interval according to Sect. 5.2.1) and the best static drowsy configuration (256-cycle interval according to Sect. 5.2.2) for comparison purposes.

For the evaluation of *AVPD* we carried out a comprehensive set of experiments for many configurations using different *decreasing threshold* and *increasing threshold* values. In this paper we only present the most representative configurations:

- Configuration 00/100 (*decreasing threshold* set to 0% / *increasing threshold* set to 100%): this is the most conservative policy since *AVPD* will only try to decrease the decay interval if none of the entries are reactivated; and it will only try to increase the decay interval when all the entries are reactivated. It works pretty well for all the studied predictors as it does not take any risks when changing the decay interval.
- Configuration 50/50: this is the most aggressive configuration as it continuously keeps changing the decay interval, increasing or decreasing the decay interval according to the *reactivation ratio*.
- Configurations 40/60 and 70/100: these are the best ones we have found for the different predictors. The 40/60 is also aggressive but works well with the Stride predictor, as it balances long decay intervals with short ones. The 70/100 configuration tends to shorten the decay interval whenever possible, only raising it when all decayed entries are reactivated.

Figure 15 shows the average leakage energy savings for the DFCM predictor and the cited adaptive configurations as well as the best static and drowsy decay intervals (512 and 256 cycles respectively). For this predictor, the best adaptive configuration is 70/100 and surpasses the best static decay and drowsy schemes for all evaluated

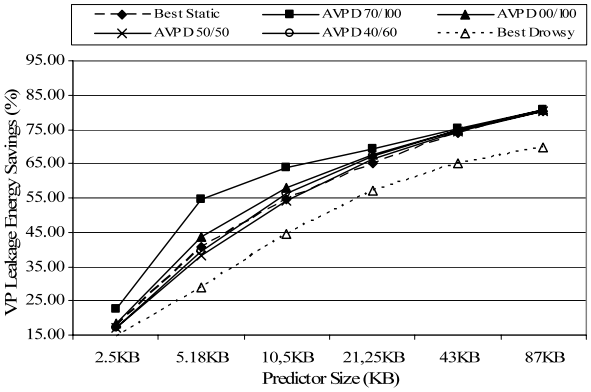


**Fig. 15** DFCM value predictor leakage energy savings (SpecInt2000)

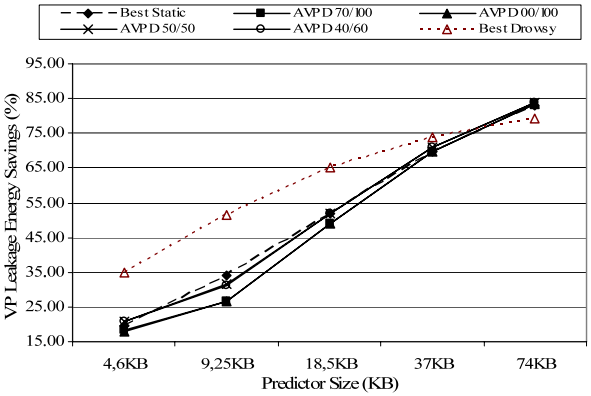


**Fig. 16** STP value predictor leakage energy savings (SpecInt2000)

predictor sizes. For an average size of 10.5 KB, *AVPD* obtains 64% average leakage energy savings *versus* 55% of the static scheme and 44% of drowsy. For the smaller size of 5 KB, the difference between the adaptive and static schemes is even more evident: *AVPD* provides additional average leakage energy savings of 14% with respect to the static scheme (*AVPD* obtains 55% and the static scheme just 41%) and 26% with respect to the drowsy scheme (55% versus 29%). It can be observed that, as size grows, the differences between the adaptive and static schemes disappear, both obtaining 80% average leakage energy savings for a size of 87 KB. In such large predictors there is no need for an adaptive scheme, because there are very low generational changes, and they can be easily identified by the static scheme. The 70/100 configuration is the best one we have found since it tends to reduce the decay interval toward its lower limit (256 cycles). In general, we have seen that whatever configuration that tends to shorten the decay interval will perform well with DFCM, but constant changes of the decay interval will result in a loss of net leakage energy savings.

Figure 16 shows the average leakage energy savings for the STP predictor. As cited in Sect. 4.2, the *AVPD* mechanism tries to decrease the decay interval in order to reduce the leakage energy. The STP predictor is especially susceptible to these trials of reducing the decay interval since a big interval reduction degrades the STP

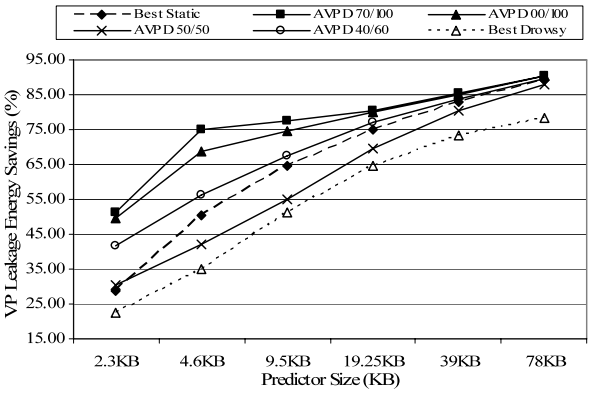


**Fig. 17** FCM value predictor leakage energy savings (SpecInt2000)

accuracy (as shown in Fig. 4) enough to make the power overhead due to the induced extra cycles equal to the power savings provided by *AVPD*. This makes the adaptive scheme to behave similarly to the static scheme. The STP predictor works better with configurations that change the decay interval quickly, like 50/50 or 40/60, because configurations that tend to shorten the decay interval (like 70/100) decrease the predictor's accuracy too much, making the overhead much greater than the provided energy savings. On the other hand, as said in Sect. 5.2.2, drowsy dominates over decay approaches for the STP due to the reduction of the decay interval from 4 K cycles to 256. This reduction increases power savings more than the drowsy costs, without any major effect on performance, making the energy savings larger for drowsy than decay schemes.

Finally, Fig. 17 shows the average leakage energy savings for the FCM predictor. This value predictor behaves very similarly to DFCM, with the same best configuration of 70/100, but obtaining even greater leakage energy savings. In addition, the differences compared to the best static decay and the best drowsy schemes are also higher. For a predictor size of 4.6 KB, the static decay approach obtains 50% leakage energy savings whereas the adaptive scheme obtains 74% (an additional 24%). Drowsy only obtains 35% average leakage energy savings for that size. For bigger sizes, the difference between the static and adaptive schemes keeps lowering until it converges to the same leakage energy savings for big predictor sizes (close to 90% savings for a size of 78 KB), but still perform better than the static drowsy. If we focus on moderated FCM sizes (around 10 KB), the best static decay scheme obtains 64% average leakage energy savings, while the best drowsy obtains 51%, and the *AVPD* obtains 77% (13% and 28% of additional savings). Note that FCM, like DFCM, performs well with any configuration that tends to decrease the decay interval, due to the negligible impact on its accuracy.

## 6 Conclusions

This paper analyzes several mechanisms able to dramatically reduce the average leakage energy of traditional value predictors with negligible impact on neither prediction accuracy nor processor performance making *Value Prediction* a low-power approach

to increase performance in energy-efficient microprocessor designs. We must reduce leakage in any possible structure, despite its size, as smaller and hotter structures can leak more than larger but cooler ones. In this paper we evaluate both state and non-state preserving techniques for reducing leakage power in value predictors including *Static Value Prediction Decay (SVPD)* [4], *Adaptive Value Prediction Decay (AVPD)* [5] and a novel power-performance *drowsy* approach for *Value Predictors*. *SVPD* and *AVPD* are mechanisms able to dramatically reduce the leakage energy of traditional value predictors with negligible impact on neither prediction accuracy nor processor performance, even in the proposed pessimistic scenario where all instructions are predicted. These techniques dynamically locate VP entries that have not been accessed for a noticeable amount of time and switch them off to prevent leakage. On the other hand, the drowsy approach will locate these entries and lower the supply voltage, retaining the data but requiring five additional cycles to access it again.

*AVPD* extends the static decay approach in order to better exploit the variable program behavior as well as the different code sections where the value predictor can be infra-utilized. On the other hand, the drowsy approach shows how important is to retain the data in some cases (STP predictor) at the expense of a reactivation penalty. Also note that if an entry has been unused for a long time, even if drowsy can restore its previous state, the prediction is likely to be incorrect either because the instruction inside the predictor is different (destructive interference) or because it has entered a new program phase and the predictor must warm up before being able to predict accurately.

The *AVPD* mechanism requires just slight modifications, with virtually no extra hardware overhead compared to the static decay scheme, and it is able to beat both static decay and drowsy for the most precise value predictors (FCM and DFCM) while the STP predictor behaves better with the drowsy approach. The reason is that the STP predictor is more sensitive to losing the entries' contents than FCM and DFCM, since the STP has all the information stored in only one table while the other are two level predictors. Experimental results show that, for the DFCM value predictor and considering an average predictor size of 10 KB, the leakage energy savings obtained by *AVPD* surpass the static approach by 14% and the drowsy approach by 24%, on average.

Finally, the present work shows that the use of low-power value prediction structures could make Value Prediction still a power-performance efficient mechanism suitable for both low-power and high-performance processor designs.

**Acknowledgements** This work has been jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 05831/PI/07, and also by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2006-15516-C04-03".

## References

1. Borkar S (1999) Design challenges of technology scaling. IEEE Micro 19(4)
2. Butts JA, Sohi G (2000) A static power model for architects. In: Proc of the 33rd int symp on microarchitecture

3. Calder B, Reinman G, Tullsen DM (1999) Selective value prediction. In: Proc of the 26th int symp on computer architecture
4. Cebrián JM, Aragón JL, García JM (2007) Leakage energy reduction in value prediction through static decay. In: Proc of the int workshop on high-performance, power-aware computing HP-PAC'07 (in conjunction with IPDPS'07)
5. Cebrián JM, Aragón JL, García JM, Kaxiras S (2007) Adaptive VP decay: making value predictors leakage-efficient designs for high performance processors. In: Proc of the ACM int conference on computing frontiers (CF)
6. Flautner K et al (2002) Drowsy caches: simple techniques for reducing leakage power. In: Proc of the 29th int symp on computer architecture
7. Flynn MJ, Hung P (2005) Microprocessor design issues: thoughts on the road ahead. IEEE Micro 25(3):16–31
8. Gabbay F, Mendelson A (1997) Speculative execution based on value prediction. Technical Report 1080, Technion – Israel Institute of Technology
9. Goeman B, Vandierendonck H, de Bosschere K (2001) Differential FCM: increasing value prediction accuracy by improving table usage efficiency. In: Proc of the 7th int symp on high-performance computer architecture
10. Hu Z et al (2002) Applying decay strategies to branch predictors for leakage energy savings. In: Int conf on computer design, Sep
11. Juang P et al. (2004) Implementing branch-predictor decay using quasi-static memory cells. ACM Trans Archit Code Optim, 1
12. Kaxiras S, Hu Z, Martonosi M (2001) Cache decay: exploiting generational behavior to reduce cache leakage power. In: Proc of the 28th int symp on computer architecture
13. Kesharvarzi A (1997) Intrinsic iddq: Origins, reduction, and applications in deep submicron low-power CMOS ICs. In: Proc of the IEEE international test conference
14. Kim NS et al (2003) Leakage current: Moore's law meets static power. In: IEEE Computer
15. Kırman N, Kırman M, Chaudhuri M, Martínez JF (2005) Checkpointed early load retirement. In: Proc of the intl symp on high-performance comp architec
16. Li Y et al (2004) State-preserving vs non-state-preserving leakage control in caches. In: Proc of the DATE conference
17. Lipasti M, Wilkerson C, Shen J (1996) Value locality and load value prediction. In: Proc of the 7th int conf on architectural support for programming languages and operating systems
18. Powell MD et al (2000) Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In: Proc of the int symp on low power electronics and design
19. Sazeides Y, Smith JE (1997) The predictibility of data values. In: Proc of the 30th annual international symposium of microarchitecture
20. Yang S et al (2001) An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-Caches. In: Proc of the 7th int symp on high-performance computer architecture
21. Zhang Y, Paritkh D, Sankaranarayanan K, Skadron K, Stan M (2003) HotLeakage: a temperature-aware model of subthreshold and gate leakage for architects
22. Zhou H, Toburen MC, Rotenberg E, Conte TM (2001) Adaptive mode-control: A static-power-efficient cache design. In Proc of the int conf on parallel architectures and compilation techniques
23. Bhargava R, Kurian L Latency and energy aware value prediction for high-frequency processors. In: Proc of the int conference on supercomputing