Adaptive VP Decay: Making Value Predictors Leakage-efficient Designs for High Performance Processors

Juan M. Cebrián[1], Juan L. Aragón[1], José M. García[1] and Stefanos Kaxiras[2]

[1]Dept. of Computer Engineering
University of Murcia,
Murcia, 30100, Spain
+34 968 367656

{jcebrian,jlaragon,jmgarcia}@ditec.um.es

[2]Dept. of Electrical and Computer Engineering
University of Patras
Rio, 26500 Patras, Greece
+30 2610 996441

kaxiras@ee.upatras.gr

ABSTRACT

Energy-efficient microprocessor designs are one of the major concerns in both high performance and embedded processor domains. Furthermore, as process technology advances toward deep submicron, static power dissipation becomes a new challenge to address, especially for large on-chip array structures such as caches or prediction tables. Value prediction emerged in the recent past as a very effective way of increasing processor performance by overcoming data dependences. The more accurate the value predictor is the more performance is obtained, at the expense of becoming a source of power consumption and a thermal hot spot, and therefore increasing its leakage. Recent techniques, aimed at reducing the leakage power of array structures such as caches, either switch off (non-state preserving) or reduce the voltage level (state-preserving) of unused array portions.

In this paper we propose the design of leakage-efficient value predictors by applying adaptive decay techniques in order to disable unused entries in the prediction tables. As value predictors are implemented as non-tagged structures an adaptive decay scheme has no way to precisely determine the induced miss-ratio due to prematurely decaying an entry. This paper explores adaptive decay strategies suited for the particularities of value predictors (Stride, DFCM and FCM) studying the tradeoffs for these prediction structures, that exhibit different pattern access behaviour than caches, in order to reduce their leakage energy efficiently compromising neither VP accuracy nor the speedup provided. Results show average leakage energy reductions of 52%, 70% and 80% for the Stride, DFCM and FCM value predictors of 20 KB respectively.

Categories and Subject Descriptors

C.1.1 [**Processor Architectures**]: Single Data Stream Architectures – *RISC/CISC, VLIW architectures*.

CF'07, May 7–9, 2007, Ischia, Italy.

General Terms: Measurement, Performance, Design.

Keywords: Energy efficient architectures, leakage, value prediction, cache decay.

1. INTRODUCTION

Energy consumption and power dissipation are one of the main goals when facing the design of a modern microprocessor in the high performance domain and, more crucially, in the embedded microprocessor domain, especially in the case of battery-operated devices. There are two sources of power dissipation, *dynamic* power 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]. However, as feature size shrinks to allow greater transistor density and higher performance, supply voltage must be lowered in order to restrain dynamic power consumption since it is proportional to the square of supply voltage. But using smaller geometries, 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 [5][14].

Several proposals can be found in the literature for managing leakage power, at both circuit and architecture level. Some proposals have focused on reducing the leakage power by switching off unused portions of large array structures, in particular for caches, since they occupy a significant fraction of total die area, therefore, providing a great opportunity for reducing leakage energy. 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. This *non-state preserving* technique has also been successfully applied to branch predictors and *BTB* structures [8][11].

On the other hand, Value Prediction (VP) has been proposed as a very effective way of improving superscalar processor performance [6][7][10][17] by overcoming data dependences which are one of the major performance limitations in current high performance superscalar processors. More recently, VP has also been successfully proposed to perform early load retirements in high performance processors [15]. However, the use of value prediction structures despite the speedup provided (average 15% as reported in [2]) has not been widely spread, mainly due to complexity-delay

issues. Note however that, unlike other prediction structures such as branch predictors where increasing access time and complexity can significantly reduce their benefits since the next fetched instruction is needed as early as possible, the access time in VPs is not so crucial. First, the predicted value is not needed until the instruction has reached its issue stage, and second, current high performance processors typically implement deeper pipelines (14 stages or more) which effectively *hide* the VP latency due to the increased front-end pipeline length. When an instruction reaches the end of the multi-stage front-end, the predicted value allows a speculative issue of the instruction if any register input is not ready, making traditional VP a very effective way of increasing processor performance.

However, the use of VP structures incurs in 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 in the case of caches and branch predictors. In modern high performance processors, due to high operating temperatures, it is necessary to fight 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. We cannot afford not to attack leakage even at the smallest structures.

In this paper we propose *Adaptive Value Prediction Decay (AVPD)*, a mechanism able to dramatically reduce the leakage energy of traditional Value Predictors with negligible impact on prediction accuracy nor processor performance by dynamically locating VP entries that have not been accessed for a noticeable amount of time. When those entries have been identified, *AVPD* switches them off to prevent them from leaking, which makes Value Predictors complexity-effective structures (due to the minimal extra hardware required) when used in medium and long pipelines as well as a power-performance efficient mechanism suitable for high performance processor designs. It is important to note that VPs show a significant amount of spatial and temporal locality but, unlike caches, a prematurely decayed VP entry does not degrade performance as much as a prematurely decayed cache line since losing the contents of a VP entry might result –or not– in a value misprediction on the next access to that entry, but this is exactly what would happen if we had a real generational change. On the other hand, prematurely decaying a cache line always induces additional accesses to the L2 cache.

Previous proposals that applied *static decay* approaches to both caches and branch predictors needed to carefully choose a decay interval, which could be even tuned per application, in order to minimize the performance impact of leakage power reduction. However, even obtaining the best decay interval per application (by profiling techniques) does not guarantee the best energy savings, since the static decay approach cannot capture variations within an application. This is particularly important in the case of prediction structures since correct and wrong predictions usually appear clustered. An *adaptive decay* approach can dynamically choose decay intervals at run-time to match the generational behaviour of particular entries. In [12], an adaptive decay approach –suited for caches– able to set decay intervals *per cache line* was proposed. Despite the extra flexibility provided, the hardware overhead required to manage decay intervals per cache line resulted in moderate net leakage energy savings. In [20], the authors proposed another adaptive decay approach, again suited for caches, exploiting the fact that caches have tags and deactivating only the data portion of cache lines but not the tag portion. By doing so, the *ideal* miss rate, even when deactivating a cache line prematurely, can be calculated and compared to the *actual* induced miss rate in order to guide the adaptive scheme, at the expense of having the tag array leaking all the time.

The contribution of the present work is a novel adaptive decay scheme suited for the peculiarities of Value Predictors (to the best of our knowledge this is the first such proposal). The new *Adaptive Value Prediction Decay* (*AVPD*) approach is needed for two reasons. First, adapting the decay interval individually for the very small 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 not feasible to track the ideal miss rate *vs.* the induced miss rate. The proposed *AVPD* takes the best attributes of each of the two previous adaptive decay proposals for the purpose of VP decay. It 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 or not the current decay interval causes an inordinate number of entries to be prematurely shutoff. Finally, we have evaluated *AVPD* in terms of dynamic and static power consumption, instead of using indirect metrics such as *active ratio* and *turn-off ratio* metrics.

The rest of the paper is organized as follows. Section 2 analyzes the utilization of the prediction tables and the static decay approach. The proposed *AVPD* scheme is described in Section 3. Section 4 shows the experimental methodology and the leakage energy savings obtained. Section 5 provides some background and reviews some related works. Finally, Section 6 summarizes the main conclusions of the work.

2. Problem Overview

2.1 Generational Behaviour in Value Predictors

Power dissipation of value prediction structures is divided into dynamic and static power, as cited before. The dynamic component strongly depends on the utilization of the VP tables. Values can be predicted at different demanding levels: the most aggressive utilization predicts the output value for all instructions traversing the pipeline. 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 L1 or L2 data cache, instructions that belong to a 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 present, as the VP structure leaks regardless of utilization with increasing leakage loss for finer process technologies. For this reason, this work is focused on reducing the VP structure's static power component.

The authors in [12] showed that, very frequently, cache lines have an initial active period (known as *live time*) followed by a period of no utilization (known as *dead time*) before they are eventually evicted. They proposed to break the stream of references to a particular cache line into generations. Each generation lasts until the cache line is evicted and replaced by a new one. This generational

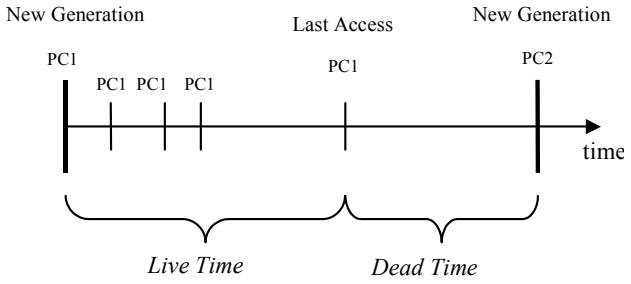

Figure 1. Different generations for a value predictor entry.

behaviour also appears in the VP structure, although with some particularities: as value predictors are implemented as direct-mapped tables with no tags and allowing destructive interferences, in our proposal, a generation ends when the VP entry is accessed by an instruction with a different PC, as it can be seen in Figure 1. Its *live time* will be the period of accesses with the same PC and its *dead time* will be the period between the last access with an specific PC until an access with a different one.

To better understand the generational behaviour in value predictors, Figure 2 shows the utilization of the VP entries by measuring the fraction of time each entry remains in a *dead* state[1] for the whole SPECint2000 benchmark suite as a function of VP size. 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 VP 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 VP entries. Long periods of inactive *live time* could be also 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 next sections.

2.2 Static Decay Scheme for Value Predictors

In this section we perform a detailed analysis on the leakage-efficiency of the *static decay* approach when applied to traditional Value Predictors as well as an introduction to the potential benefits that an *adaptive decay* scheme could achieve.

The *static decay* scheme suited for value predictors needs to detect those VP entries that have been unused for a significant period of time in order to switch them off [3]. But in order to successfully apply decay techniques, it is necessary to carefully choose the number of cycles we should wait before shutting an entry off in order to match generational changes. 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

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

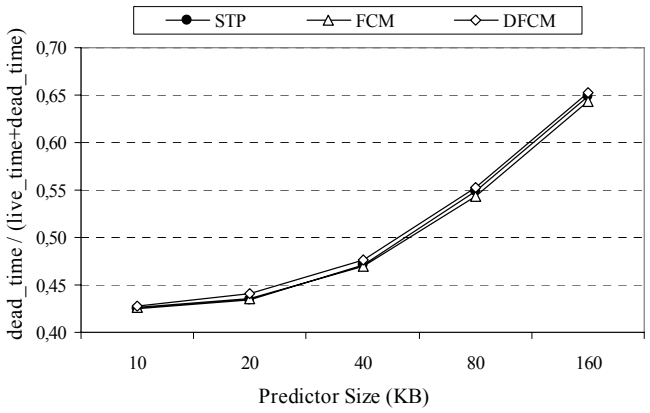

Figure 2. Fraction of time VP entries are in dead state (SpecInt2000).

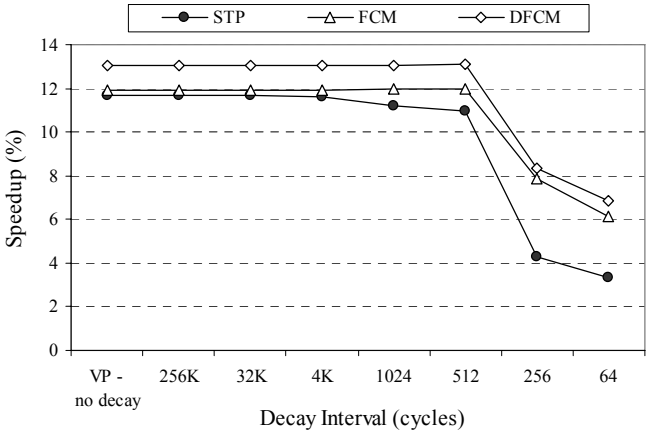

Figure 3. Average speedup for the *static decay* scheme for 10KB value predictors (SPECint2000).

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.

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 they are 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.

In order to better understand the effects of prematurely deactivating a VP entry, Figure 3 shows the speedup provided by Value Prediction with no decay as well as the speedup reduction when applying the static decay approach for decay intervals from 256 Kcycles to just 64 cycles. It is important to note that, as cited previously, traditional Value Prediction (with no decay) can provide significant average speedups (13% for a 10 KB DFCM as shown in Figure 3). Looking into the performance degradation caused by *static decaying*, we can notice that for FCM and DFCM there is no IPC degradation until 256-cycle decay intervals. For STP, there is a

slight but negligible IPC degradation (less than 1%) for 1024- and 512-cycle decay intervals. As before, for 256-cycle (and smaller) intervals the performance degradation is not tolerable.

Next analysis performs an evaluation on the energy-efficiency of the static decay scheme for VPs. Unlike previous cache decay proposals [12][20], we are reporting leakage energy measurements using a modified version of *HotLeakage* simulator [19] that includes the static power model for the evaluated VPs as well as the static and dynamic power overhead of the static decay approach (see section 4.1 for details about simulation methodology and processor configuration). In order to precisely evaluate the net leakage energy savings provided by the static VP decay approach, it is necessary to consider the following overheads associated with the mechanism. 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[2] per VP entry [3]). The second component overhead is derived 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 dynamic and static processor power and can easily cancel whatever VP leakage energy savings provided by the static decay scheme.

Figure 4 shows the average leakage energy savings provided by the static VP decay scheme for the DFCM value predictor when considering different decay intervals and VP sizes (sizes are not power-of-two numbers because of the extra 2-bit counters per entry). As expected from the IPC degradation showed in Figure 3, the best decay interval corresponds to a window of 512 cycles for all VP sizes. For a predictor size of about 10 KB, static VP decay obtains average leakage energy savings of 55% and, for a 20 KB DFCM, the average leakage energy savings are 65%. As expected, greater leakage energy savings can be obtained for greater VP sizes. The greatest energy savings for VPs are obtained for decay intervals within the 512-cycle range, unlike data caches where the best decay intervals are within the 8-Kcycle range [12]. This due to the fact that the average *live time* is around 400 cycles for VPs (note also that the decay mechanism needs some additional cycles to determine that an entry has entered in a *dead state*).

Figure 5 is similar to Figure 4 but for the FCM value predictor. As it can be observed, the two VPs behave very similarly to each other. Again, the best static decay interval corresponds to a window of 512 cycles. However, the average leakage energy savings are greater for the FCM predictor. For a size of about 10 KB, static VP decay obtains average leakage energy savings of 64% and, for a size around 20 KB, the average leakage energy savings are 75%. In these two figures it can be noticed that, for decay intervals smaller than 512 cycles, the IPC degradation is not tolerable since the overhead due to the induced extra cycles completely cancels all the leakage power savings provided from static decaying, resulting in negative net leakage energy savings.

Finally, Figure 6 shows the leakage energy savings, per benchmark, for a 10 KB DFCM predictor. This Figure reveals one of the weaknesses of the static decay scheme when compared to the adaptive scheme: the static decay interval must be carefully chosen

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

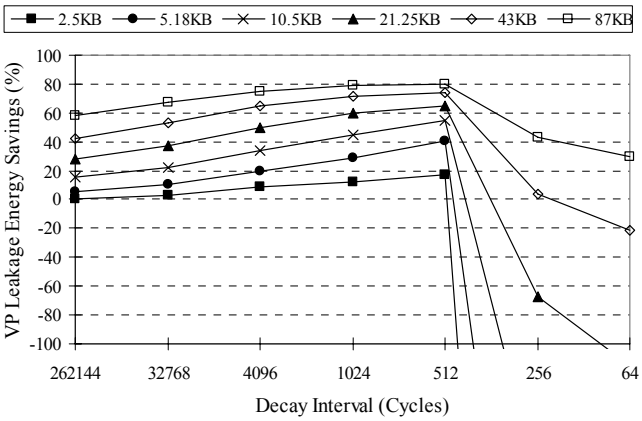

Figure 4. Static decay scheme for the DFCM value predictor (SPECint2000).

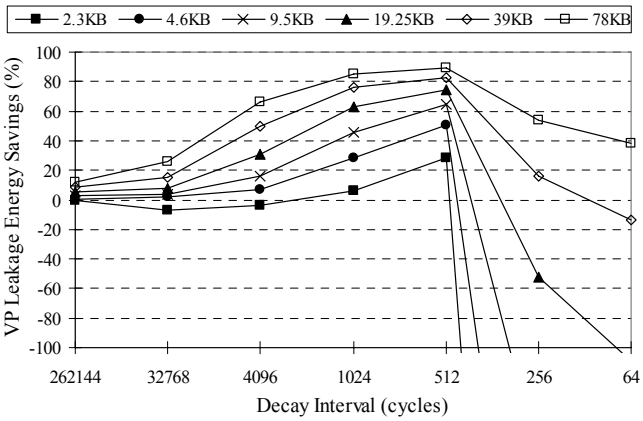

Figure 5. Static decay scheme for the FCM value predictor (SPECint2000).

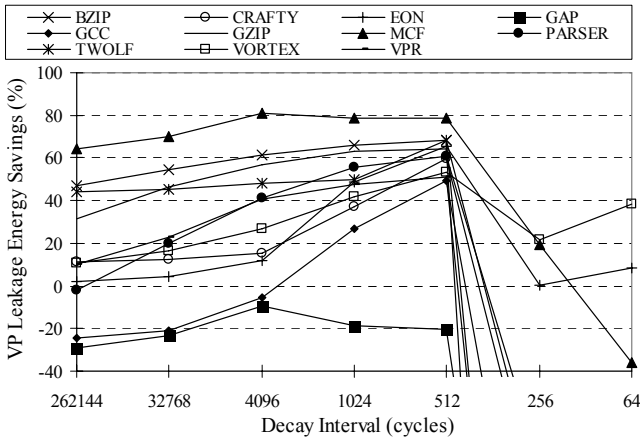

Figure 6. Static decay scheme for a 10 KB DFCM value predictor.

in order to maximize the leakage energy savings. In some cases, the best static decay interval may differ between applications (as it can be seen in Figure 6 for *mcf* and *gap* benchmarks where the best static decay interval is 4 Kcycles). However, even when using profiling techniques in order to determine the best decay interval per

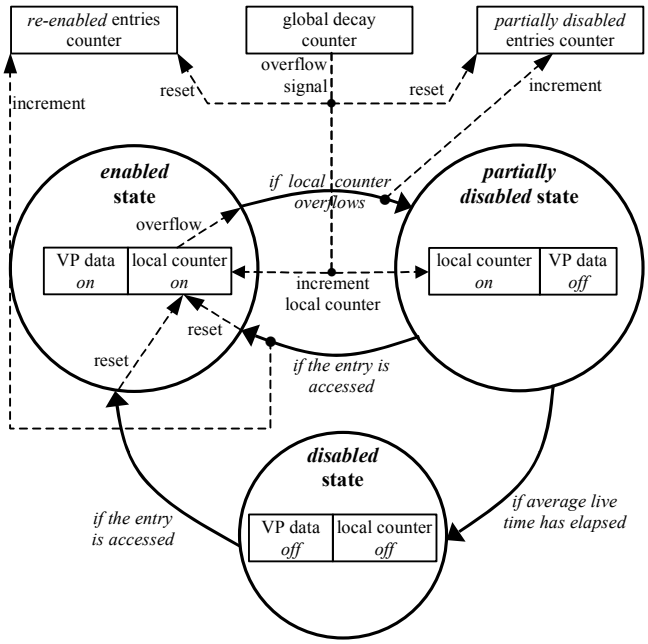

Figure 7. *AVPD* mechanism.

application, there is no guarantee that the best leakage energy savings are obtained, since the *static decay* approach cannot capture variations within an application. This second effect is particularly important in the case of VP 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 behaviour in prediction table entries.

3. Adaptive Value Prediction Decay (*AVPD*)

Dynamically applying decay techniques to Value Predictors is not a trivial fact as we need to detect those VP entries that have been unused for a significant amount of time and switch them off to prevent them from leaking. *Adaptive Value Prediction Decay* (*AVPD*) is a time-based mechanism that analyzes each VP entry individually to detect how often that entry is accessed. 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.

The problem is to dynamically determine how long a decay interval must be. If we choose to turn VP entries off using too long decay intervals, the potential leakage energy savings will be reduced. Conversely, if the time-based policy chooses too short decay intervals, the VP accuracy might be reduced and, therefore, inducing a performance degradation. A positive effect of *AVPD* compared to the original cache decay mechanism is that prematurely disabling a VP entry is not so 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 information incurring in extra execution cycles; however, losing the contents of a VP entry might result –or not– in a value misprediction on the next access to that entry but this is exactly what would happen if we had a real generational 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).

As shown in the previous section, the decay interval is dependant on the application running in the processor or even on the section of the code 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). In other program sections the number of VP entries being accessed is low, or we can 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 it.

The implementation of the decay interval is done by means of a hierarchical counter composed of a *global counter* and a two-bit saturated gray-code counter for each individual value predictor entry[3] (*local counters*). 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_{DD} transistors [16]. These "sleep" transistors are inserted between the ground (or supply) and the cells of each VP entry, which reduces the leakage in several orders of magnitude and it can be considered negligible. An alternative to using *gated-*V_{DD} transistors consists of using quasi-static 4T transistors, although similar leakage savings would be expected [11].

The *AVPD* mechanism considers that each VP entry can be in one of the following three states, as shown in Figure 7: *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 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 *re-activation ratio* is calculated as the number *re-enabled* entries over the number of *partially disabled* entries.

In addition, *AVPD* uses two pre-defined threshold values (*increasing threshold* and *decreasing threshold*) in order to determine whether the length of the current decay interval is correct, that is, if the current decay interval makes VP entries to decay during their *live time* (prematurely) or during their *dead time*. Therefore, if the *re-activation ratio* is higher than the *increasing threshold*, the current decay window is too short and it is doubled since the are many entries being disabled prematurely. On the other hand, if the *re-activation 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, loosing opportunities to reduce the VP leakage.

The *AVPD* mechanism works as follows (see Figure 7): each cycle the global decay counter is incremented by one and, when it

[3] 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 associated power overhead.

[4] As cited in section 2.2, the static decay experiments showed that the *average live time* is around 400 cycles for the three evaluated VPs.

overflows, the local counters of all VP entries in either the *enabled* or *partially disabled* state are incremented. However, an access to any VP entry will result on 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[4], 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): an access to the entry will change it to the *enabled* state.

Regarding the pre-defined 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 *re-activations* before raising the decay interval. In Section 4.2 we evaluate the leakage-efficiency of the *AVPD* mechanism for different *increasing* and *decreasing thresholds*.

Finally, the power overhead associated to the *AVPD* mechanism can be divided into three main components. The first component is associated to the dynamic and static power derived from the two-bit local counters inserted into every entry of the predictor (same overhead as for the static decay scheme). The second component comes from the three global counters: one is part of the two-level decay interval counter (also appears in the static decay scheme) and the other two counters are particular of the adaptive decay scheme. The third component overhead, as explained in Section 2.2, 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. Note that this third component (also appears in the static decay scheme) is highly destructive since each extra cycle accounts for the overall processor dynamic and static power and can easily cancel whatever leakage energy savings provided by *AVPD*.

Therefore, we must try to make the *AVPD* mechanism accurate enough to not increase the execution cycles. It is important to note that *AVPD* is virtually not introducing additional power overhead nor complexity (just the additional two global counters whose power overhead that has been conveniently modelled into the *AVPD* power model) when compared to the *static* decay scheme providing, however, significant additional leakage energy savings as we will show in next section.

Table 1. SPECint2000 benchmark characteristics.

Benchmark	Input set	Total # simulated instr. (Mill.)	# skipped instr (Mill.)
bzip2	input source 1	500	500
crafty	test (modified)	437	-
eon	kajiya image	454	-
gap	test (modified)	500	50
gcc	test (modified)	500	50
gzip	input.log 1	500	50
mcf	test	259	-
parser	test (modified)	500	200
twolf	test	258	-
vortex	test (modified)	500	50
vpr	test	500	100

Table 2. Configuration of the simulated processor.

Processor Core	
Process Technology:	70 nanometers
Frequency:	5600 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
	2 Memports
Pipeline:	22 stages
Memory Hierarchy	
L1 Icache:	64KB, 2-way
L1 Dcache:	64KB, 2-way
L2 cache:	2MB, 4-way, unified

4. Experimental Results

4.1 Simulation Methodology

To evaluate the energy-efficiency of the *AVPD*, 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 [19] that includes the dynamic and static power model for the evaluated Value Predictors (Stride, FCM and DFCM) as well as the power overhead associated to *AVPD*. The VP access latency is 5 cycles.

Table 1 shows, for each particular benchmark, the input set, the total number of simulated instructions and the number of forwarded instructions. Due to the large number of dynamic instructions in some benchmarks, we reduced the input data set while keeping a *complete* execution. Table 2 shows the configuration of the simulated architecture. Leakage related parameters have been taken from the Alpha 21264 processor, provided with the HotLeakage simulator suite, and using a process technology of 70 nanometers.

4.2 Leakage-efficiency of *AVPD* Mechanism

This section presents the leakage-efficiency evaluation of the proposed *AVPD* mechanism for the Stride, FCM and DFCM

predictors. Each figure shows the VP *leakage energy* savings[5] respect to not applying a decay scheme for some representative configurations of the adaptive mechanism as well as the best static decay configuration (512-cycle decay interval according to section 2.2) for comparison purposes.

For the evaluation of *AVPD*, we carried out a comprehensive set of experiments for many configurations defined by using different *decreasing* and *increasing threshold* values. In this work 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 try to decrease the decay interval only if none of the entries are re-activated; and it will only try to increase the decay interval when all the entries are re-activated. It works pretty well for all 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 keeps changing the decay interval continuously, increasing or decreasing the decay interval according to the *re-activation ratio*. This configuration is so aggressive that the constant changes on the decay interval neutralize, for many benchmarks, the VP energy savings with the overhead of the extra execution cycles.
- Configurations 40/60 and 70/100: they are the best ones we have found for the different predictors. The 40/60 is quite aggressive but works well with the Stride predictor, as it balances long decay intervals with short ones. The 70/100 configuration has the trend to shorten the decay interval whenever is possible, only raising it when all decayed entries are re-activated.

Figure 8 shows the average leakage energy savings for the DFCM predictor and the cited adaptive configurations as well as for the best static decay interval (512 cycles). For this predictor, the best adaptive configuration is 70/100 that surpasses the best static decay scheme for all evaluated predictor sizes. For an average size of 10.5 KB, *AVPD* obtains 64% leakage energy savings versus the 55% of the static scheme. For the smaller size of 5 KB, the difference between the adaptive and static schemes is even more evident: *AVPD* provides additional leakage energy savings of 14% respect to the static scheme (*AVPD* obtains 55% and the static scheme just 41% of leakage energy savings). It can be observed that, as size grows, the differences between the adaptive and static schemes disappear, both obtaining 80% leakage energy savings for a size of 87 KB. In such big size predictors, there is no need for an adaptive scheme as 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 its trend is to reduce the decay interval towards its lower limit of 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, like in the 50/50 configuration, will result in a loose of net leakage energy savings.

Figure 9 shows the average leakage energy savings for the STP predictor. As cited in section 3, 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 accuracy enough (as shown in Figure 3 of Section 2.2) to

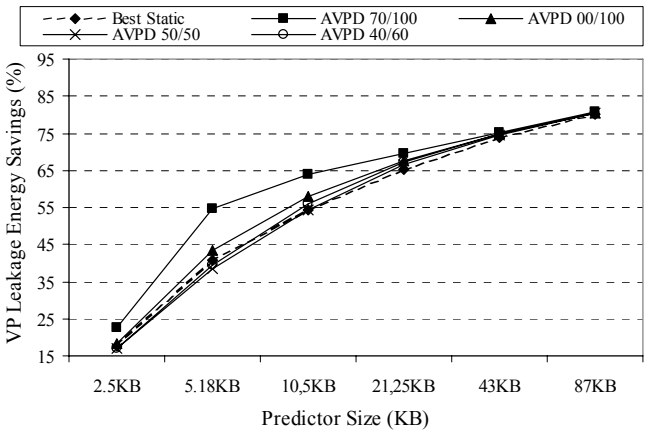

Figure 8. DFCM value predictor leakage energy savings (SPECint2000).

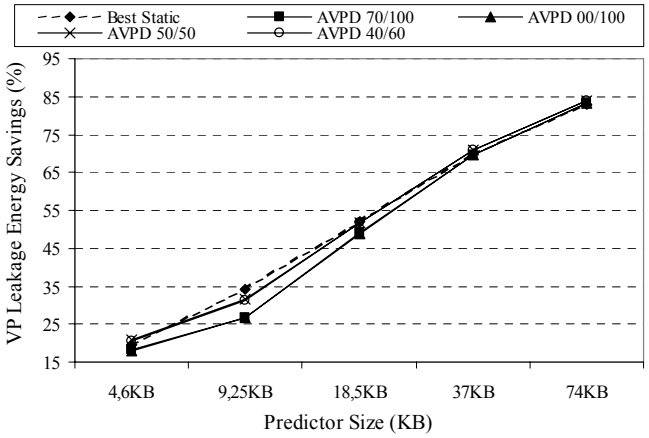

Figure 9. STP value predictor leakage energy savings (SPECint2000).

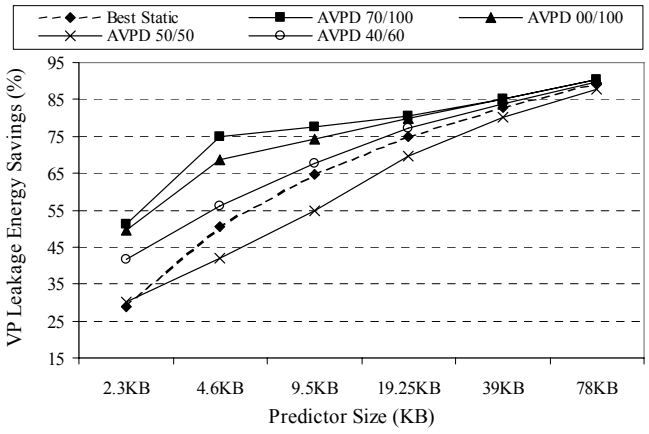

Figure 10. FCM value predictor leakage energy savings (SPECint2000).

[5] Total processor leakage-energy results are not presented due to HotLeakage limitations that only provides static-power models for regular array structures (caches, predictors and register file).

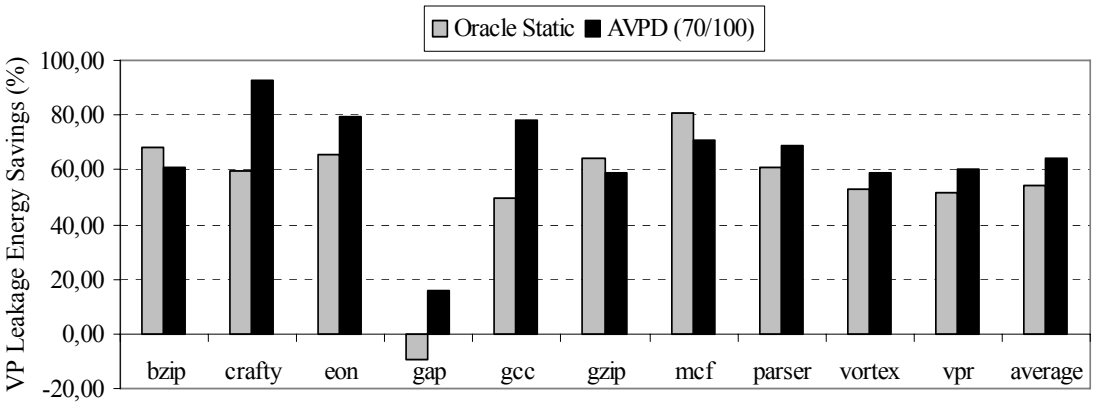

Figure 11. Oracle Static decay *vs* AVPD for a 10 KB DFCM predictor.

make the power overhead due to the induced extra cycles equal to the power savings provided by *AVPD*. This results in 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 with a trend to shorten the decay interval (like 70/100) decrease the predictor's accuracy too much, making the overhead even greater than the provided energy savings.

Figure 10 shows the average leakage energy savings for the FCM predictor. This predictor behaves very similarly to DFCM, with the same best configuration (70/100), but obtaining even greater leakage energy savings. In addition, the differences compared to the best static decay scheme are also higher. For a predictor size of 4.6 KB, the static approach obtains 50% leakage energy savings whereas the adaptive scheme obtains 74% (an additional 24%). For greater sizes, the differences between the static and adaptive schemes keep lowering until they converge to the same leakage energy savings for very big predictor sizes (close to 90% leakage energy savings for a size of 78 KB). If we focus on moderated FCM sizes (around 10 KB), the best static scheme gets 64% leakage energy savings whereas *AVPD* obtains 77% (13% 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.

Finally, Figure 11 compares the proposed *AVPD* along with an *oracle static* decay scheme for a DFCM predictor of about 10 KB. The *oracle static* decay is the one able to choose the best static decay interval per application, as it can be extracted from Figure 6. According to Figure 11, we can notice that *AVDP* can surpass, for most applications, the leakage energy savings provided by the oracle static approach (up to additional 33% energy savings for *crafty* and 29% for *gcc*) which makes *AVPD* an energy-effective mechanism to more precisely match the dynamic generational behaviour in Value Prediction structures.

5. Related Work

In order to reduce leakage power in processors, several techniques have been proposed at both circuit level and architectural level. At the architectural level, many proposals have focused on reducing the leakage power by switching off unused portions of large array structures such as caches. These techniques have been categorized into *state-preserving* and *non-state preserving* [1][9][18].

Studies by Powell *et al.* [16] proposed *gated-*V_{DD} as a technique to limit static leakage power by banking and providing "sleep" transistors which dramatically reduce leakage current by gating off the supply voltage. This technique, known as *decay*, reduces the leakage power drastically at the expense of losing the cell's contents, being necessary to apply it very carefully since the loose of information can result in an increase of the dynamic power to retrieve it again. 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 [8][11].

On the other hand, *drowsy* techniques try to reduce leakage without losing the cell's information. Drowsy caches [4] use different supply voltages according to the state of each cache line. The lines in drowsy mode use a low-voltage level, retaining the data, while 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 additional penalty cycles (about 7 cycles). Of course, the leakage savings of this mechanism are lower than the decay ones, but the increase of dynamic power consumption due to the loose of information is also lower. Flautner *et al.* [4] showed that a drowsy cache putting to sleep all cache blocks periodically achieves 54% leakage power savings with a negligible performance degradation (about 1%).

Li *et al.* [9] evaluated the use of state and non-state preserving techniques in caches. The authors showed that for a fast L2 cache (5-8 cycles latency) decay techniques are superior in terms of both performance loss and energy savings to drowsy ones.

An alternative to traditional decay is to use quasi-static, four-transistor (4T) memory cells. 4T cells are approximately as fast as 6T SRAM cells, but do not have connections to the supply voltage (V_{SS}). Rather, the 4T cell is charged upon each access, whether read or write, and it slowly leaks the charge over time until, eventually, the value stored is lost. In [11], it was proposed to apply decay techniques to branch predictors by using 4T cells. By doing this, some of the drawbacks of using gated-V_{DD} transistors are eliminated, since an access to a 4T cell automatically reactivates the cell, whereas reactivating a 6T cell from the "sleep" mode is somewhat more complex, requiring extra hardware involved in gating the supply voltage.

Studies by Zhou *et al.* [20] show an adaptive time based mechanism suited for caches to dynamically disable cache lines in order to reduce leakage power dissipation. This mechanism uses the cache tag array, which is never switched off, in order to track if there are many induced cache misses and adapt the decay interval accordingly.

Regarding the work on Value Prediction, in the literature it can be found a plethora of studies and proposals. The *last value predictor* was introduced by Lipasti *et al.* [10]. 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.* [6], it uses the last value produced by an instruction plus a stride pattern. The *finite context method* value predictor (FCM), introduced by Sazeides *et al.* [17], 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 context of the recent history of the instruction. The second level stores, for each possible context, the value which is most likely to follow it. The *differential finite context method* value predictor (DFCM), introduced by Goeman *et al.* [7], joins the two previous predictors into one structure. DFCM works like FCM (two-level prediction tables), but it stores the *differences* between the values instead of the values themselves, plus the last value of the instruction. This allows DFCM to capture stride patterns. For non-stride patterns, DFCM works just like the FCM predictor.

6. Conclusions

This paper proposes *Adaptive Value Prediction Decay* (*AVPD*), a mechanism able to dramatically reduce the leakage energy of traditional Value Predictors with negligible impact on prediction accuracy nor processor performance by dynamically locating VP entries that have not been accessed for a noticeable amount of time. Once those unused entries have been located, *AVPD* switches them off to prevent them from leaking and, therefore, making Value Prediction an energy-efficient approach for low-power processor designs. The proposed *AVPD* extends the static decay approach in order to better exploit the program behaviour as well as the differences between sections of code where the VP can be under-utilized.

The *AVPD* mechanism requires just slight modifications, with virtually no extra hardware overhead compared to the static decay scheme (just two additional global counters). *AVPD* combines the best attributes, for the purpose of VP decay, of two previous adaptive decay schemes. From [12] it inherits the time-based scheme of detecting early shutoffs but shuns adaptation of the decay interval per entry. Conversely, it inherits adaptation of a single global decay interval from [20] but discards miss ratio measurement since that would require tags. In addition, in our scheme, the aggressiveness of the adaptation is easily controlled by two parameters (*increasing* and *decreasing thresholds*).

Initially, we present in this work an analysis on the leakage-efficiency of the *static decay* approach when applied to traditional Value Predictors (Stride, DFCM and FCM). After that, we evaluate the leakage-efficiency of the proposed *AVPD* showing that it can indeed surpass the leakage energy savings provided by the static decay approach (and even an oracle static decay). The average leakage energy savings for the best known configuration of the adaptive mechanism for a moderated predictor size of around 10 KB are 32%, 64% and 77% for the three evaluated predictors, Stride, DFCM and FCM, respectively. Compared to the best static decay scheme, *AVPD* provides significant *additional* average leakage energy savings (e.g., 14% for a 5 KB DFCM and 24% for a 5 KB FCM).

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

7. ACKNOWLEDGMENTS

This work has been supported by the Ministry of Education and Science of Spain under grants TIN2006-15516-C04-03 and CSD2006-00046. This work is also supported in part by the EU FP6 IP-FET SARC contract No. 27648, and the EU FP6 NoE HiPEAC IST-004408.

8. REFERENCES

[1] J.A. Butts and G. Sohi. “A static power model for architects”. In *Proc. of the 33rd Int. Symp. on Microarchitecture*, 2000.

[2] B. Calder, G. Reinman and D.M. Tullsen. “Selective Value Prediction”. In *Proc. of the 26th Int. Symp. on Computer Architecture*, May 1999.

[3] J.M. Cebrián, J.L. Aragón and J.M. García. “Leakage Energy Reduction in Value Predictors through Static Decay”. In *Proc. of the Int. Workshop on High-Performance, Power-Aware Computing HP-PAC’07 (in conjunction with IPDPS'07)*, March 2007.

[4] K. Flautner et al. “Drowsy Caches: Simple Techniques for Reducing Leakage Power”. In *Proc. of the 29th Int. Symp. on Computer Architecture*, 2002.

[5] M.J. Flynn and P. Hung. “Microprocessor Design Issues: Thoughts on the Road Ahead”. *IEEE Micro*, vol. 25, no. 3, pp. 16-31, May/Jun, 2005.

[6] F. Gabbay and A. Mendelson. “Speculative execution based on value prediction”. Technical Report 1080, Technion – Israel Institute of Technology, 1997.

[7] B. Goeman, H. Vandierendonck and K. de Bosschere. “Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency”. In *Proc. of the 7th Int. Symp. on High-Performance Comp. Architecture*, 2001.

[8] Z. Hu et al. “Applying Decay Strategies to Branch Predictors for Leakage Energy Savings”. In *Proc. of the Int. Conf. on Computer Design*, Sep. 2002.

[9] Y. Li et al. “State-Preserving vs. Non-State-Preserving Leakage Control in Caches,” In *Proc. of the DATE Conference*, Feb. 2004.

[10] M. Lipasti, C. Wilkerson and J. Shen. “Value locality and load value prediction”. In *Proc. of the 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[11] P. Juang et al. “Implementing Branch-Predictor Decay Using Quasi-Static Memory Cells”. *ACM Transactions on Architecture and Code Optimization*, Vol. 1, June 2004.

[12] S. Kaxiras, Z. Hu and M. Martonosi. “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power”. In *Proc. of the 28th Int. Symp. on Computer Architecture*, 2001.

[13] A. Kesharvarzi. “Intrinsic iddq: Origins, reduction, and applications in deep sub-micron low-power CMOS IC’s”. In *Proc. of the IEEE International Test Conference*, 1997.

[14] N.S. Kim, T. Austin *et al*. “Leakage Current: Moore’s Law Meets Static Power”. *IEEE Computer*, 2003.

[15] N. Kırman, M. Kırman, M. Chaudhuri and J.F. Martínez. “Checkpointed early load retirement”. In *Proc. of the Intl. Symp. on High-Performance Comp. Architec.*, Feb. 2005.

[16] M.D. Powell *et al*. “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*, 2000.

[17] Y. Sazeides and J.E. Smith. “The predictibility of data values”. In *Proc. of the 30th Annual International Symposium of Microarchitecture*, Dec 1997.

[18] S. Yang *et al*. “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*, 2001.

[19] Y. Zhang, D. Paritkh, K. Sankaranarayanan, K.Skadron and M. Stan. “HotLeakage: a temperature-aware model of subthreshold and gate leakage for architects”. Technical Report, Dept. Comp. Science, U. Virginia, 2003.

[20] H. Zhou, M. C. Toburen, E. Rotenberg and T. M. Conte. “Adaptive mode-control: A static-power-efficient cache design”. In *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques*, 2001.