Reducing the Number of Bits in the BTB to Attack the Branch Predictor Hot-Spot

N. Tomás, J. Sahuquillo, S. Petit and P. López

Dept. of Computing Engineering (DISCA)
Universidad Politécnica de Valencia, Valencia, Spain
ntomas@gap.upv.es, {jsahuqui, spetit, plopez}@disca.upv.es

Abstract. *Current superscalar processors access the BTB early to anticipate the branch/jump target address. This access is frequent and aggressively performed since the BTB is accessed every cycle for all instructions in the ICache line being fetched. This fact increases the power density, which could create hot spots, thus increasing packaging and cooling costs. Power consumption in the BTB comes mostly from its two main fields: the tag and the target address fields. Reducing the length of either of these fields reduces power consumption, silicon area and access time. This paper analyzes at what extent tag and target address lengths could be reduced to benefit both dynamic and static power consumption, silicon area, and access time, while sustaining performance. Experimental results show that the tag length and the target address could be reduced by about a half and one byte, respectively with no performance losses. BTB peak power savings can reach about 35% when both reductions are combined together, thus effectively attacking the hot-spot.*

1 Introduction

Traditionally, a key issue to improve microprocessor performance has been the clock frequency. The higher the clock frequency, the faster the processor is able to execute instructions. However, such increase in clock frequency yields to an increase in the processor power consumption. As the frequency has kept rising, the power consumption and the corresponding package and cooling costs have become prohibitive, constraining such trend. It has been estimated [1] that heat dissipation above 30-40 watts increases the total cost of thermal packaging and cooling per chip by more than one dollar per watt. The costs can be as high as those derived from a 130W heat dissipation [2]. Consequently, power consumption has become nowadays a major microprocessor design concern. Power consumption can be reduced by either reducing the static power (i.e., the power consumed due to transistor leakage currents) or dynamic power (i.e., the power consumed due to transistor switching).

Focusing on the average power is important when designing power-limited devices constrained by the power supply such as batteries. However, power is not distributed uniformly across the chip area but on specific structures. In other words, the different microprocessor structures present different power densities [3]. Those structures with highest density may trigger thermal emergencies when working at peak power, and are referred to as thermal hot-spots. These points are the main responsible for the costs of

Table 1. Increase of temperature per cycle

Component	Area (mm^2)	Peak power (W)	Temp. increase (°C)
Load/Store Queue	0.5	2.7	0.7e-3
Instruction Window	0.9	10.3	3.1e-3
Register File	0.25	5.5	2.6e-3
Branch Predictor	0.35	5.3	5.2e-3
L1 Data Cache	1.0	11.6	2.6e-3
Integer Units	0.5	4.3	2.5e-3
Floating Point Units	0.5	6.7	2.6e-3

thermal packaging and cooling solutions, since they must be designed to operate at peak power (even if peak power is rarely reached). That means that, to allow high frequencies while keeping the packaging and cooling costs at a reasonable cost, solutions should address hot spots.

On the other hand, in current microprocessors, deep pipelines have become predominant in order to allow high clock speeds. In these microarchitectures, the fetch stage bandwidth must be high enough to efficiently feed the remaining stages. In this context, fast handling of branch instructions is important because they can stall the fetch stage until their direction (i.e., taken or not taken) and target address are known, thus dropping the performance. To deal with control hazards, branch prediction techniques have been extensively studied during the last decade [4–8].

The branch predictor can be accessed multiple times every cycle. Thus, this structure presents a high power density. Because of this reason, it is one of the scorching hot-spots in the processor. This fact can be appreciated in Table 1, which shows the temperature increase per cycle reached by different main processor structures when working at 1.5Ghz (peak performance). These results are derived from the model detailed in [9] for a processor model similar to the Alpha 21264 using a 100°C baseline temperature. Notice how the branch predictor, despite having one of the smallest areas, is the main heat contributor in the list.

A major structure of the branch predictor is the branch target buffer (BTB), and it constitutes an important percentage of its heat. The BTB has been implemented in modern microprocessors in two different ways. In some processors, the BTB performs both predictions (target address and direction). However, most current microprocessors decouple the target address prediction, which is performed by a BTB table, from the branch direction, which is performed by a branch predictor structure. This decoupled implementation has been the model addressed in this paper.

Different works have been proposed to reduce power consumption in the BTB. In this paper, we study at what extent power consumption might be reduced by merely reducing the tag length, the target address or both. By reducing the power consumption of the BTB without adding any extra hardware, we reduce the peak consumption of a major hot-spot of the processor. The benefits of this reduction are twofold. On one hand, packaging and cooling costs can be decreased, while on the other hand, there are more opportunities for increasing the processor frequency. However, as a side effect, both branch direction and target address misprediction rise, which may negatively impact on performance.

This paper analyzes the effects of reducing the tag and the target address bits of the BTB on power, energy, area, access time, and performance. Experimental results show that the tag length and the target address could be reduced by about a half and one byte, respectively with no performance losses. When both techniques are applied together savings can reach about 35%. Notice that, as we propose a hardwired technique, these benefits consistently apply regardless of whether the BTB is working or not at peak power. Thus, the hot-spot is effectively attacked.

The remaining of this paper is organized as follows. Section 2 discusses some related work. Section 3 summarizes the proposed reduction of the BTB tag and target address length, and its pros and cons. Section 4 analyzes the experimental results and finally, some concluding remarks are drawn.

2 Related Work

Important research work has focused on reducing the BTB power consumption. Some of these works attempt to reduce the power consumption in the BTB by reducing the number of accesses [10–12], while other approaches apply at the circuit level [13, 14].

Regarding the first approach, Petrov and Orailoglu [10] propose a mechanism that uses control flow information (e.g., basic block lengths), obtained at compile time. This information is stored in a table called the ACBTB (Application Customizable Branch Target Buffer). Two auxiliar registers are used, one to count the number of instructions until the next branch, and other to index the table. When the counter reaches zero, the ACBTB is accessed to read the target address. Then, the counter is updated with a value according to the stored control information.

Deris and Baniasadi [11] propose the Branchless Cycle Prediction (BLCP) that uses a structure to predict which cycles the ICache line has no branch (branchless cycles). Thus, there is no need to access the BTB. To this end, a small Global History Shift Register (GHR) and a Prediction History Table (PHT) are used. The GHR records the history of the branch and the branchless cycles. This table indexes the PHT, which predicts whether a branch instruction will be fetched in the next cycle. If the prediction is false, the BTB is not accessed. The technique proposed by Palermo *et al.* for VLIW architectures [12] uses a branch detector that partially pre-decodes instructions to find possible branches, in order to access the branch prediction block selectively, and thus to the BTB.

Regarding the second approach, Chaver *et al.* [13] propose an adaptive resizing of the BTB. To this end, some portions of the BTB are selectively disabled using dynamic cache resizing techniques.

Hu *et al.* [14] apply decay strategies which, from time to time, switch-off specific entries in the BTB, so reducing leakage energy at the expense of some performance loss.

Unlike these techniques, this paper neither reduces the number of accesses nor disables BTB entries. Instead, we analyze how power consumption can be saved by reducing the number of bits in the BTB, so saving power consumption for every performed access. In this way, we address not only average power savings but also thermal pack-

aging and cooling costs. Nevertheless, notice that the aforementioned techniques are orthogonal to our approach, so they can be applied all together.

3 Proposed Mechanism

The branch target buffer is implemented like a cache structure that has two main fields: the tag and the target address. The tag field is compared with the corresponding bits of the PC of the instruction being fetched. On a hit, if the branch prediction outcome is taken, the stored target address is written to the PC, otherwise, it is discarded. When the real target address is calculated later in the pipeline, it is compared with the predicted one. If the prediction fails, the BTB must be updated with the correct target address.

In modern microprocessors, the BTB is accessed at the same time as the instruction cache. At this point, it is not known whether the instruction being read is a branch or not. A straightforward solution to force that only branches may hit the BTB is to store in the tag field the whole branch address (i.e., its PC). Of course, if some of these bits are used to index the BTB (i.e., implemented as a set-associative table), there is no need to store all of them. In addition, as memory is usually byte addressable but instructions are word aligned, some least significant bits of the instructions address can be discarded since they are always equal to zero.

Another key issue is to analyze the relationship between the target address of a branch and its PC (i.e., how distant they are). In this sense, if both PCs are close enough, they will share some of the most significant bits. In such a case, if the target address field is reduced, the BTB would provide only a fraction of such address, but the final address could be obtained by using the most significant bits of the PC accessing the BTB.

The key issue behind the proposal is to reduce the number of stored bits in the BTB in order to remove static and dynamic power consumed by these bits. As a side effect, area and access time are also reduced. The proposal attacks the two main parts of the BTB (i.e., the tag and the target address).

3.1 Reducing the number of Tag Bits

Reducing the number of tag bits might affect the hit ratio, because phantom branches and branch aliasing could arise. This section discusses the impact of reducing the tag field on these negative effects.

The problem raises because there could be more than one instruction matching the same tag (of course, only one of them is the correct one), regardless of whether they are branch instructions or not, therefore resulting in BTB misspeculations. This kind of misspeculation is named in two different ways depending on whether the instruction causing it is a branch or not.

A phantom branch is a non-branch instruction that hits the BTB, thus, it is executed as a normal branch. Consequently, if it is predicted taken, the PC is updated with the target address read from the BTB and a wrong inflow of instructions is inserted in the processor pipeline until the misprediction is detected. This is just like a common branch misspeculation, but in this case, it can be detected much earlier, (i.e., when decoding

the instruction at the decode stage). Thus, it is not necessary to wait for the branch to be resolved later in the pipeline to recover the machine to a precise state. Notice that only the fetched and not yet decoded instructions are affected. Thus, the only action to be done is to squash those instructions, without affecting instructions from the ROB neither the mapping table.

Branch aliasing refers to different branches that match the same tag in the BTB. The problem is that any of these branches might be predicted with the target address of any other, what would incur a BTB misprediction. This misprediction is detected when the target address is calculated later in the pipeline, and would cause branch misspeculation if the branch is predicted taken. Notice that this case can be handled as a normal branch misspeculation. Thus, to recover from this misspeculation, there is no need to add extra logic.

Reducing the tag length, not only reduces the number of memory cells but also the comparator size (i.e., the number of XOR gates). This fact will positively impact on both static and dynamic power. In addition, the area, the circuit delay and, the BTB access time will be also reduced.

3.2 Reducing the number of Target Address Bits

The field storing the target address in the BTB may be reduced by removing either of its bits. However, as instructions are stored in continuous memory addresses, they share their most significant bits. Thus, only a subset of the bits might be stored in the BTB and the effective target address could be computed by concatenating the most significant bits of the PC of the instruction accessing the BTB with the fraction of the target address stored in the BTB. In such a case, the problem is that the effective target address might be wrong, thus introducing BTB mispredictions that would result in branch misspeculation if the branch was taken. This kind of misspeculation is handled as a normal branch misspeculation, thus there is no need to add extra logic to detect and recover from it. On the other hand, the benefits of reducing the target address length are similar to those of reducing the tag length, since a reduction in the number of memory cells is applied.

Again, there is a tradeoff between power, area and access time, and performance loss. Thus, a fair analysis should take into account the advantages and shortcomings of reducing the tag and target address lengths. The key issue is to look for how many bits could be removed without hurting the performance.

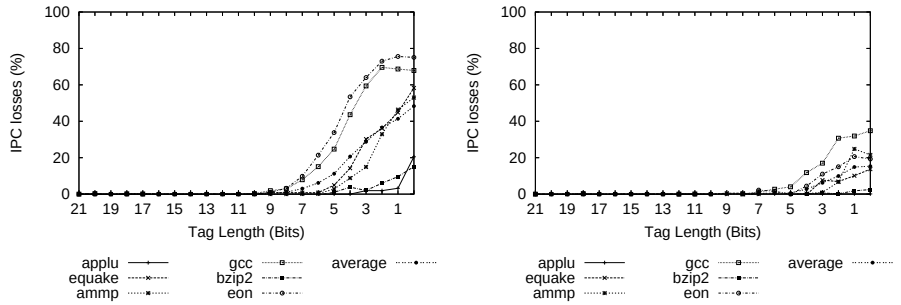
4 Experimental Results

Experiments were run by using the Multi2Sim simulation framework [15], which was extended to support different tag and target address lengths, detecting phantom branches and branch aliasing, and the SPEC2000 benchmark suite [16] as workload. To measure the energy, area and cycle time of the BTB, the CACTI 4.0 tool [17], a widely used cache timing and power model, has been used.

Experimental results have assumed a 32-bit address microprocessor configuration as shown in Table 2, with a baseline 4-way 512-set BTB implemented using 65 nm silicon process.

Table 2. Machine Parameters.

Microprocessor core	
Branch predictor	Hybrid gShare/bimodal
gShare	4KB 2-bit counters
Bimodal	4KB 2-bit counters
Choice Predictor	4KB 2-bit counters
BTB	512 sets, 4 ways, 13 cycles misprediction, 3 cycles phantom branch penalty
Decode/Issue/Retire bandwidth	4 instructions/cycle
# of Int ALU's, mult/div	4,1
# of FP ALU's, mult/div	2,1
Memory Hierarchy	
L1 data cache	64KB, 2 way, 64 byte-line
L1 data cache hit latency	1 cycles
L2 data cache	512KB, 4 ways, 64byte-line
L2 data cache hit latency	10 cycles
Memory access latency	200 cycles



(a) due to both branch aliasing and phantom branches

(b) due to *only* branch aliasing

Fig. 1. IPC losses when reducing the tag length

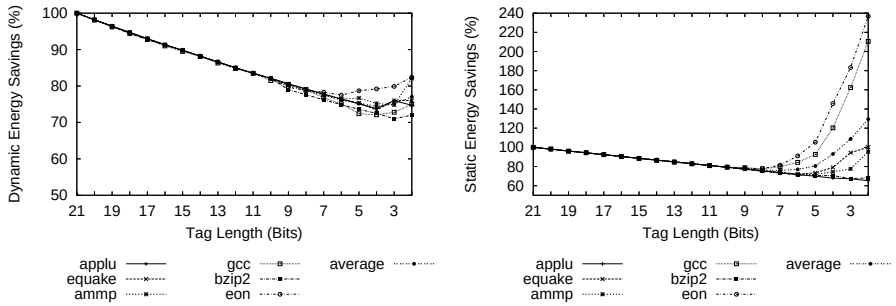
4.1 Reducing Tag Bits

First, the effect of reducing the tag length has been evaluated alone (i.e., without reducing the target address length). Note that the length of the tag stored in the BTB, without applying any reduction in baseline processor, must be 21 bits[1]. Thus, the analysis starts assuming a 21-bit tag BTB, and this length is progressively reduced on 1-bit steps, down to zero bits.

Impact on Performance. As far as the tag length is reduced, performance (i.e., IPC) drops as phantom branches and branch aliasing rise. Figure 1(a) shows how phantom branches and branch aliasing impact on the IPC. As observed, if we use less than 9-10 bits, there is a strong impact on performance, but using just a 10-bit tag (about three times smaller) the performance loss is almost negligible. Figure 1(b) shows the IPC losses due only to branch aliasing (by using branch predecoding). In this case, the tag could be reduced to 6-7 bits without negatively impacting performance. Therefore, the highest fraction of performance losses comes from phantom branches.

Impact on Energy, Area and Access Time. Reducing the tag length also brings hardware benefits, since it reduces the energy consumption, the silicon area and the access time of the BTB. These benefits have been measured ranging the tag size from 21

[1] 21 = 32 – 2 (4-byte words) – 9 (512 sets)



(a) Dynamic (b) Static

Fig. 2. Energy Savings when Reducing the Tag Length

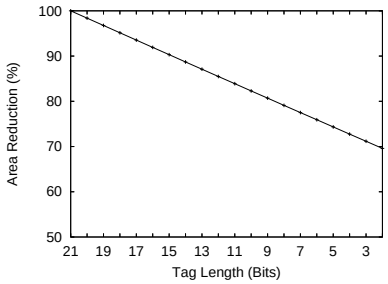


Fig. 3. Area Reduction when Reducing the Tag Length

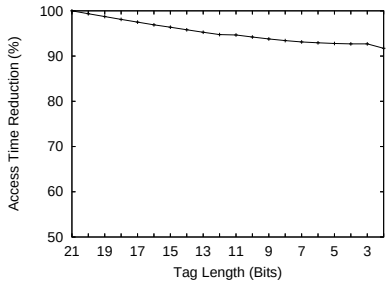


Fig. 4. Access Time Reduction when Reducing the Tag Length

down to 2 bits. Power savings have been calculated taking into account the whole execution time. For the dynamic power, first CACTI has been used to measure the energy of a single access to the BTB with every tag length. Then, this energy has been multiplied by the number of accesses during the execution time. For the static power, the transistor has been taken as consumption unit and the static energy has been calculated for every tag length. Then, this energy has been multiplied by the number of execution cycles. Notice that under performance loss conditions (higher execution time), this analysis is optimistic because the extra energy consumed by processor structures other than the BTB had not been accounted. Nevertheless, in the absence of performance losses (e.g., a 10-bit tag length) this analysis remains valid.

Figure 2(a) and Figure 2(b) show the dynamic and static power consumption respectively. As expected, as the tag length is reduced, the BTB power is also decreased. However, notice that, for some applications, a very low tag length gives no further improvements due to increased accesses to the BTB in case of mispredictions. A 9-bit tag length BTB, which has been shown to have no impact on the processor performance, has about 20% less dynamic power consumption and about 20 % less static power consumption.

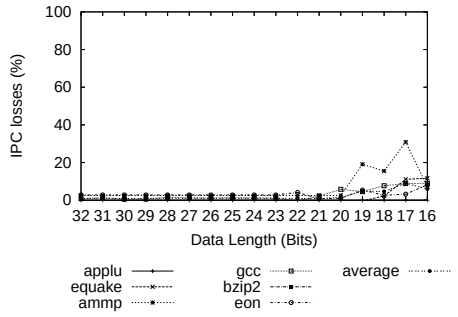


Fig. 5. Combined IPC Losses

Finally, Figure 3 and Figure 4 show the results for area and access time. Both of them are reduced as the number of tag bits decrease. Again, a 9-bit tag length gives a good tradeoff, requiring about 20% less silicon area and reducing the access time about 6%.

4.2 Reducing Target Address Bits

This analysis assumes an initial 32-bit target address length, which is progressively reduced down to zero bits while keeping fixed the baseline target address length. Due to space restrictions, results are not plotted but they are discussed.

Impact on Performance. As the target address length is reduced, performance drops due to BTB address mispredictions that, in case of branch predicted as taken, results in a branch misspeculation. Results show that IPC losses rise as the target address length is reduced down to 21 bits for all applications other than applu and bzip2. The explanation is that, in these cases, some BTB mispredictions are hidden due to branches predicted as not taken, thus no branch mispeculation appears. Consequently, there is no negative impact on the IPC.

Impact on Energy, Area and Access Time. Regarding power savings, results show that a good tradeoff value could be a 24-bit target address field, which achieves across the different applications by about 15% and 20% of dynamic and static power savings, respectively. More aggressive target address reduction dramatically impact on performance, and as a consequence, increase the energy budget. Concerning silicon area and access time improvements, results show that a 24-bit target address length achieves about 7% area reduction, decreasing the access time by about 6%.

4.3 Combining Tag and Target Address Reduction

This section explores the impact of reducing simultaneously the number of bits in the tag and the target address fields. To this end, we assume a fixed 10-bit tag since, as shown above, it is the smallest size with no performance losses. Figures 5, 6, 7, 8, and 9 plot the results. As expected, when combining both reductions, IPC losses begin to

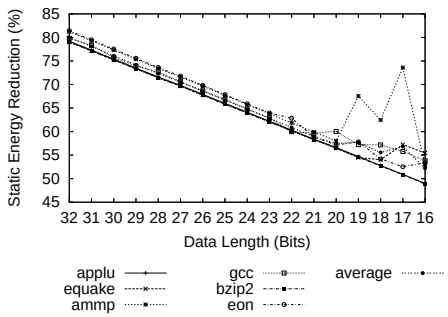


Fig. 6. Combined Static Energy Reduction

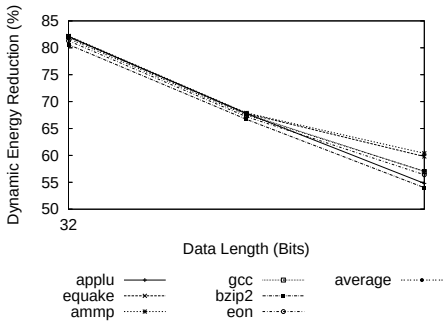


Fig. 7. Combined Dynamic Energy Reduction

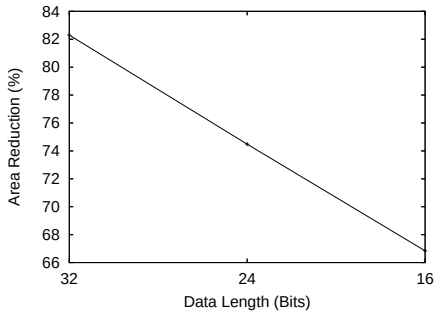


Fig. 8. Combined Area Reduction

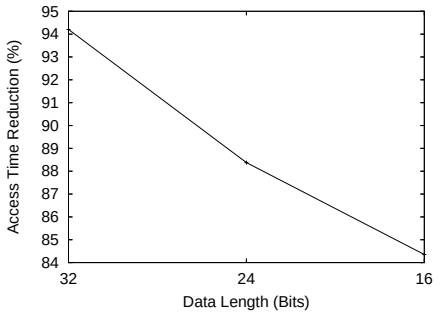


Fig. 9. Combined Access Time Reduction

arise for target address lengths shorter than 24 bits. Thus, at this point is when we reach the major benefits without hurting performance, achieving savings of 32%, 35%, 26%, and 12% in dynamic energy, static energy, area, and access time, respectively.

5 Conclusions

The BTB is a key structure of the branch prediction unit, which is aggressively accessed every cycle, and acts as a major hot-spot in current microprocessors. This work has analyzed how both BTB dynamic and static power consumption can be saved by reducing the tag length, the target address length or both. Consequently, silicon area and access time are also reduced. Side effects, such as possible adverse impacts on performance or extra energy due to performance dropping, have been also analyzed.

Results show that a 10-bit tag has no adverse impact on performance while providing important benefits (e.g., by about 17% of power savings). These results can be improved if target address is simultaneously reduced. For instance, using a 10-bit tag length with a 24-bit target address length, power savings grow up 35% with no performance losses.

Acknowledgements

This work was supported by Spanish CICYT under Grant TIN2006-15516-C04-01, CONSOLIDER-INGENIO 2010 under Grant CSD2006-00046 and Universidad Politécnica de Valencia under Grant PAID-06-07-20080029.

References

1. Borkar, S.: Design challenges of technology scaling. IEEE Micro **19**(4) (1999) 23–29
2. : SIA. International Technology Roadmap for Semiconductors, 1999
3. Hamann, H.F., Weger, A., Lacey, J.A., Hu, Z., Bose, P., Cohen, E., Wakil, J.: Hotspot-limited microprocessors: Direct temperature and power distribution measurements. Solid-State Circuits, IEEE Journal of **42**(1) (Jan. 2007) 56–65
4. Evers, M., Yeh, T.Y.: Understanding branches and designing branch predictors for high-performance microprocessors. Proceedings of the IEEE **89**(11) (2001) 1610–1620
5. Lee, C.C., Chen, I.C., Mudge, T.: The bi-mode branch predictor. micro **0** (1997) 4
6. Sprangle, E., Chappell, R., Alsup, M., Patt, Y.: The agree predictor: A mechanism for reducing negative branch history interference. Computer Architecture, 1997. Conference Proceedings. The 24th Annual International Symposium on (2-4 Jun 1997) 284–291
7. Jimenez, D.A., Lin, C.: Dynamic branch prediction with perceptrons. hpca **00** (2001) 0197
8. Eden, A., Mudge, T.: The yags branch prediction scheme. micro **0** (1998) 69
9. Skadron, K., Abdelzaher, T., Stan, M.R.: Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In: HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, Washington, DC, USA, IEEE Computer Society (2002) 17
10. Petrov, P., Orailoglu, A.: Low power branch target buffer for application-specific embedded processors. in Proc. of Euromicro Symposium on Digital System Design (2003) 158–165
11. Deris, K.J., Baniasadi, A.: Branchless cycle prediction for embedded processors. Proceedings of the 2006 ACM symposium on applied computing (2006)
12. Palermo, G., Sam, M., Silvan, C., Zaccari, V., Zafalo, R.: Branch prediction techniques for low-power vliw processors. Proceedings of the 13th ACM Great Lakes symposium on VLSI (2003)
13. Chaver, D., Pinuel, L., Prieto, M., Tirado, F., Huang, M.C.: Branch prediction on demand: an energy efficient solution. in ISPLED'03 (2003)
14. Hu, Z., Juang, P., Skadron, K., Clark, D., Martonosi, M.: Applying decay strategies to branch predictors for leakage energy saving. Proceedings of IEEE International Conferences on Computers and Processors (2002)
15. Ubal, R., Sahuquillo, J., Petit, S., López, P.: Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. Proc. of the 19th International Symposium on Computer Architecture and High Performance Computing (Oct. 2007)
16. : Standard Performance Evaluation Corporation. http://www.spec.org/cpu2000
17. Tarjan, D., Thoziyoor, S., Jouppi, N.P.: Cacti 4.0. Hewlett-Packard Development Company, L.P., Technical Report (June 2006)