ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

INTEGRATION, the VLSI journal 38 (2005) 515–524

INTEGRATION the VLSI journal

www.elsevier.com/locate/vlsi

Low-power branch prediction techniques for VLIW architectures: a compiler-hints based approach

M. Monchiero[a], G. Palermo[a], M. Sami[a], C. Silvano[a,*], V. Zaccaria[b], R. Zafalon[b]

[a]Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34, 20133 Milano, Italy

[b]STMicroelectronics, AST–Advanced System Technology, Agrate Brianza, Milano, Italy

Received 11 June 2004; received in revised form 16 July 2004; accepted 21 July 2004

Abstract

The paper introduces a dynamic branch prediction scheme suitable for energy-aware Very Long Instruction Word (VLIW) processors. The proposed technique is based on a *compiler hint* mechanism to filter the accesses to the branch predictor blocks. We define a configurable *hint* instruction which anticipates some static information about the upcoming branch to reduce the hardware involved in the prediction, thus, the energy consumption. To analyze the effectiveness of the proposed low-power branch prediction scheme, we combined it with some well-known dynamic branch prediction techniques suitable for VLIW processors. The analyzed branch predictors are characterized by simple hardware implementations, matching the low-power characteristics of the target VLIW processors. Experimental results have been carried out on Lx, an industrial 4-issue VLIW architecture.

© 2004 Elsevier B.V. All rights reserved.

Keywords: VLIW processors; Branch prediction; Low-power design

1. Introduction

Control dependencies between instructions prevent pipelined processors from meeting maximum performance, branch prediction techniques can reduce performance degradation due to branch instructions. Modern CPUs widely adopt different dynamic branch prediction techniques: simple processors use simple predictors to soften branch penalty effects [1], modern superscalar machines

*Corresponding author. Tel.: +39-022-399-3692; fax: +39-022-399-3411.

E-mail address: silvano@elet.polimi.it (C. Silvano).

0167-9260/$ - see front matter © 2004 Elsevier B.V. All rights reserved.

doi:10.1016/j.vlsi.2004.07.012

exploit large and complex predictors to sustain high ILP [2]. Although a branch predictor is a power consuming device, it is a critical component to minimize branch penalty stalls.

Dynamic branch prediction techniques are usually associated with dynamically scheduled superscalar processors and not directly with statically scheduled Very Long Instruction Word (VLIW) processors. Most VLIW processors use static techniques to cover the branch penalty such as delayed branches [3], where the compiler is responsible for filling *branch delay slots* to overcome branch penalties.

Although dynamic branch prediction is suitable also for VLIW processors, up to now, it has not yet been implemented due to the hardware overhead. In fact, most VLIW processors (such as Philips TriMedia or TI VLIW processor family) prefer to use static branch prediction techniques associated with other hardware (e.g., *predication*) and software (e.g. *trace scheduling*) mechanisms [3]. The first attempt to explore the application of branch prediction techniques to VLIW architectures is reported in [4], where the authors consider Philips Trimedia as target processor.

In general, power dissipation can be considered as important as performance in the design of pipelined processors. When dealing with ILP pipelined processors, statically scheduled VLIW processors require simple hardware implementations, better matching the low-power requirements with respect to superscalar processors, characterized by complex hardware to implement dynamic scheduling. Therefore, the problem of branch prediction in pipelined processors must be afforded from the power/performance combined perspective, particularly in the case of low-power embedded VLIW processors.

The present paper is an extension of the work in [5], where we propose a low-power branch prediction architecture suitable for VLIW processors. In the paper, we will show a compiler-assisted *dynamic* branch prediction technique that can be efficiently adopted by VLIW processors to obtain low-energy consumption, while preserving performance, at the expense of a quite limited area overhead. Such a prediction technique is based on hardware predictor modules, which are managed to achieve a limited power dissipation. The basic idea consists of the introduction of *Hint Instructions—HIs* in the compiled code to statically move up some information, related to the branch, to the control unit. The HIs inform the processor that a branch is coming and, therefore, the branch prediction unit will be activated only when a branch occurs, thus reducing the number of accesses to the branch predictor and consequently the related power dissipation. Additional static information is contained in the HIs to save predictor accesses due to no-branch instructions. The anticipation of the target will save Target Predictor lookups, while, if also direction is available, the predictor will not be accessed at all, and branch instructions are still predicted correctly.

The paper is organized as follows. Section 2 introduces some previous works on branch prediction schemes and low-power optimizations. The problem statement and the proposed low-power branch detector techniques based on HIs are presented in Section 3. The target architecture based on an industrial VLIW and the related set of experimental results are shown in Section 4. Finally some concluding remarks are reported in Section 5.

2. Background

Microarchitectural power reduction techniques are usually combined with technology-level transformations to obtain circuits with reduced energy dissipation and leading-edge performance.

A common approach is first to increase the performance of the design and then to reduce the voltage as much as possible. Proposed methods to do this include increasing algorithm concurrency and pipelining by using faster units, and increasing the number of hardware units [6].

Many works have been targeted at reducing the performance degradation to solve control dependencies. These methods try to improve the branch prediction accuracy by using several structures [7,8]. A dynamic branch prediction technique is usually composed of two interacting mechanisms [3]: a *Branch Outcome Predictor* and a *Branch Target Predictor*. The branch outcome predictor is used to forecast the direction of the branch, i.e., *taken* or *not taken*, while the branch target predictor computes the *taken* address of the branch. These two modules are used by the instruction fetch unit to predict the next instruction to be fetched from the memory.

Several structures have been proposed for the implementation of branch outcome predictors [7]. A *Branch History Table* (BHT) is a simple dynamic branch outcome predictor that is built above a small memory accessed by using the lower bits of the PC of the current instruction. *Correlating branch predictors* [9] and *GShare* predictors [3,7] are more complicated forms of BHT, where the prediction is conditioned also by the outcome history of the previously executed branches (stored in a branch history register, BHR). These techniques are also called *local/global* branch predictor and can be generalized to *hybrid* predictors [8] where a set of predictors are selected by dynamic information. These structures are composed of a set of predictors selected by dynamic information.

For what concerns the branch target predictors, the most common implementation is the *Branch Target Buffer* (BTB) [10,11], that is a branch prediction cache storing the predicted addresses for the branch target.

Hoogerbrugge [4] compares the typical branch prediction techniques for superscalar processors against ad hoc techniques for VLIW architecture and shows that branch prediction techniques designed ad hoc for VLIW processors can reduce the misprediction rate significantly.

Research on branch prediction and power/performance exploration appeared only recently in the literature. In [12], the authors analyze several branch prediction organizations for superscalar processors from the power/performance perspective. The authors found that, to reduce the energy consumption of the processor, it is worthwhile to spend more power on the branch predictor module. The authors propose also a simple module, namely *Prediction Probe Detector—PPD*, that uses some pre-decoded bits to eliminate unnecessary accesses to predictor and BTB. The PPD is a small and fast memory which is accessed every cycle to inform the processor control logic if a branch is going to be fetched. In this way, the predictor is activated only when a branch is fetched. The authors claim an overall processor energy reduction of 6%, measured by using the Wattch [13] simulator.

In [14], the authors introduce a branch prediction scheme for a low-power VLIW processor. The aim of this technique is filtering the accesses to the branch prediction block for the no-branch instructions. In order to reduce the number of accesses, the authors implement a module, called *Hardware Branch Detector* (HWBD), to predecode and to detect branch instructions. More in detail, the HWBD module waits for the branch to be fetched by the instruction decompression buffer, then a detection logic predecodes opcode bits by activating the branch prediction unit selectively. Only if a branch is detected in the decompression buffer, the branch prediction logic is activated, thus reducing the number of accesses to the branch predictors and consequently the power dissipation.

Other authors have analyzed the problem of branch prediction by using both static and dynamic information. In [15], Chang et al. proposed a *branch classification* to alternate static and dynamic predictions. According to this technique, branch instructions are partitioned into classes and for each class the best predictor is assigned. This approach achieves better accuracy with respect to a pure dynamic predictor. Finally, in [16], the authors propose to replace the hardware selector of a hybrid predictor with a static information encoded in the branch instruction.

3. Low-power optimization

Branch prediction can improve processor performance at the cost of added hardware, and related power necessary for the prediction. Branch predictors are like small caches and they are accessed every processor cycle, so their power consumption could become significant, up to 10% of the whole processor power consumption [14].

An effective technique to obtain good power-performance trade-offs consists of reducing predictor lookups by avoiding no-branch instructions to access the branch prediction unit [12,14]. Usually, branch instructions are about 20% of the total executed instructions [3,17]. The idea of accessing the predictor modules only on branch instructions can reduce the total predictor power up to 80% [12]. The crucial point is that, in order to avoid a predictor access for no-branch instructions, it is necessary to know whether the fetched instruction will be a branch, while, at this time, the undecoded instruction is still in cache, so either we rely on a predecoded bits stored in cache, or we wait for the branch is fetched and then we decode it.

The problem of branch prediction in pipelined VLIW processors must be afforded from the power/performance combined perspective. Although dynamic branch prediction techniques are applicable not only to superscalar processors but to VLIW processors as well, these techniques require a consistent hardware and power dissipation overhead that can limit their application to low-power embedded VLIWs. From these considerations, our basic idea consists of a compiler-assisted *dynamic* branch prediction technique that can be efficiently adopted by VLIW processors to obtain low-energy consumption, while preserving performance, at the cost of a quite limited area overhead.

The solution we are proposing in this paper exploits the capability of an optimizing compiler to insert a *Hint Instruction—HI* anticipating to the processor that a branch instruction follows, and adding static prediction information. The HIs move up to the processor some information related to the next branch instruction, basically enabling the processor control logic to activate the branch prediction unit. Moreover, predictor modules accesses are saved by using static information stored in the HI. In fact, the following information can also be included in the HIs, at compile time: *Branch Target Address*, if the branch is of immediate type, since its target address is known by the compiler; *Direction Bit*, in case the branch is a static one (unconditional branch), since its outcome does not depend on dynamic behavior. HIs inform the processor that a branch is coming and, therefore, the branch prediction unit will be activated only when a branch occurs. Other static information help to save additional predictor accesses. The moving up of the target will save Target Predictor lookups, while, if direction is available as well, no predictor will be accessed at all, and yet branch instructions will be predicted correctly. In this way, depending on the type of the branch, only the necessary predictor modules are activated. We propose to extend the instruction set with a *hint* instruction

with the following format: `hint offset[, target][, direction]`; where the `offset` field indicates how many cycles to wait for the branch, and the optional `target` field and `direction` field contain the additional static information.

The only additional hardware needed by this technique is decode logic for the HI. This logic performs HI predecoding, as soon as the HI is fetched. It triggers the branch predictor modules managing `target` and `direction` fields, if they are encoded in the HI.

Peculiar VLIW code features can be exploited to make the HI insertion more efficient. Usually a VLIW compiler cannot fill every long instruction (namely, *bundle*) with useful operations [3]. Thus, the maximum number of parallel operations is hardly ever sent to the processor. The bundle is filled up with NOPs requiring often code compression techniques. Our proposal is to enable the compiler to insert a HI in an existing bundle by substituting a NOP instruction. To do this, every basic block is analyzed: if there is a bundle before the bundle containing the branch including NOPs, the HI is substituted in one of these NOPs. Otherwise, instead of generating a new bundle only to insert the HI, we prefer not to insert the HI at all to avoid increasing bundles count and, therefore, execution time, actually destroying the advantage of a correct prediction. In this way, it is not always possible to add a compiler hint for each branch.

4. Experimental results

In this section, we present the results obtained by introducing the proposed branch filtering mechanism into an industrial VLIW processor executing a set of multimedia benchmarks taken from the Mediabench Suite [18]. Our target architecture has been directly derived from the Lx processor, a scalable and customizable 4-issue (VLIW) pipeline architecture. Lx architecture implementations are the CPUs of the ST210/ST220 cores family for embedded systems, jointly designed by STMicroelectronics and Helwett–Packard Labs [19]. Lx architecture is a 'pure' VLIW with a very short pipeline, only six stages. At each cycle, a word composed of four operations is fetched from instruction cache (IC), and a flexible instruction encoding indicates parallel and serial operations. The fetched word is sent to a small instruction decompression buffer. From the buffer, the parallel operations are issued simultaneously to the back-end of the pipeline. At most, one branch instruction per cycle can be issued, although more branches (up to four, one for each fetched bundle) can be simultaneously fetched. The original Lx architecture has only a static branch prediction policy, *always not taken*, in fact, when a branch occurs, the processor keeps on fetching from the fall-through path. Branch instructions are early solved in the *decode* stage (second stage), so the misprediction penalty is only *one cycle stall*. Predication is only partially supported. Lx provides a *select* operation, which allows predicate execution of mutually exclusive assignment operations.

In the present paper, we consider several extensions to the original Lx architecture: The Lx-BP *architecture*, obtained by substituting the original Lx static branch predictor with a configurable dynamic branch prediction unit accessed in parallel with Instruction Cache; Lx with PPD or HWBD or HI-based, obtained by enhancing Lx-BP with the PPD or HWBD or HI-based low-power technique, respectively. Since every analyzed filtering method misses some branches, we assume that, when a branch is not detected, it is predicted as *not-taken*, accordingly to the static prediction policy of Lx.

Several branch outcome predictors have been explored in terms of their related parameters. With respect to the branch target predictor, during the simulation we used a 256 entries 2-way Branch Target Buffer (BTB), with 14 tag-bits [11]. The branch outcome predictors chosen for the validation of the methodology, with the relative configuration of the parameters, are: Bimodal predictor (64 B), GShare (256 B, 1 kB, 4 kB, 16 kB), PAs (192 B, 640 B, 2.2 kB, 4.1 kB), Hybrid1 Bimodal/PAs (1.3 kB), Hybrid2 GShare/PAs (1.8 kB).

The experimental results are obtained by using an in-house custom tool called BPSIM (Branch Prediction SIMulator). BPSIM is a modular and parameterized trace driven simulator with a driver-interface for typical ISSs (*Instruction Set Simulators*). The power models for the configurable branch predictor and detector schemes derived from Cacti [20] and Wattch models [13], configured for a 0.25 μm implementation running at 250 MHz, have been plugged in BPSIM. In this framework, we plugged in an Lx Instruction Set Simulator, with an instruction-accurate timing estimation module containing the power models of the core [21].

The reported experimental results are strongly dependent on the target architecture. In fact, the misprediction penalty depends on the pipeline stage in which the branches are solved. Another factor impacting the evaluation of branch prediction techniques on processor performance is the percentage of branch instructions in the target application and the probability that a branch is taken. Globally the reported results can be considered a "worst case" analysis since the percentage of branch instruction is only 14.6% on average for the selected benchmarks and the considered Lx processor has only a one-cycle branch penalty. We show two different sets of results: First, an analysis of the impact of the proposed low-power optimized branch prediction technique (HI-based) on performance and energy consumption of the Lx processor. Second, an analysis of the efficacy of HI-based with respect to cycle-by-cycle dynamic prediction. Through our analysis we compare our proposal with PPD and HWBD techniques.

First, we show the overall results obtained by applying the branch selection method that we propose (HI-based) to the Lx processor. Fig. 1 shows the two-dimensional scatter plot of the average energy and average delay for the different simulated configurations. Dynamic branch prediction causes significant performance improvements with respect to the original Lx architecture. This is because dynamic prediction improves significantly the prediction accuracy with respect to the original Lx static policy. From the point of view of energy dissipation, the results strongly depend on the adopted low-power method. As it can be seen, data relative to Lx with branch prediction can be easily grouped into four different clusters. The first cluster represents the Lx-BP architecture (that is, cycle-by-cycle access to the predictor, without filtering accesses to the branch predictor). The other three clusters are characterized by different techniques used to filter the access to the branch predictor: PPD, HWBD and HI-based. Lx-BP and PPD approaches result to be performance-effective, but, since these methods require cycle-by-cycle access to memory structures (respectively the predictor itself and the PPD), they feature large energy consumption. HI-based and HWBD obtain low-energy dissipation, because the only hardware added to the original architecture is decode logic. On the other hand, these approaches get limited delay speed-up with respect to the original Lx architecture, since some branches cannot be detected. The HI-based method is the technique which achieves the lowest energy consumption, since the static prediction component makes many predictions be correctly accomplished without predictor modules activation.

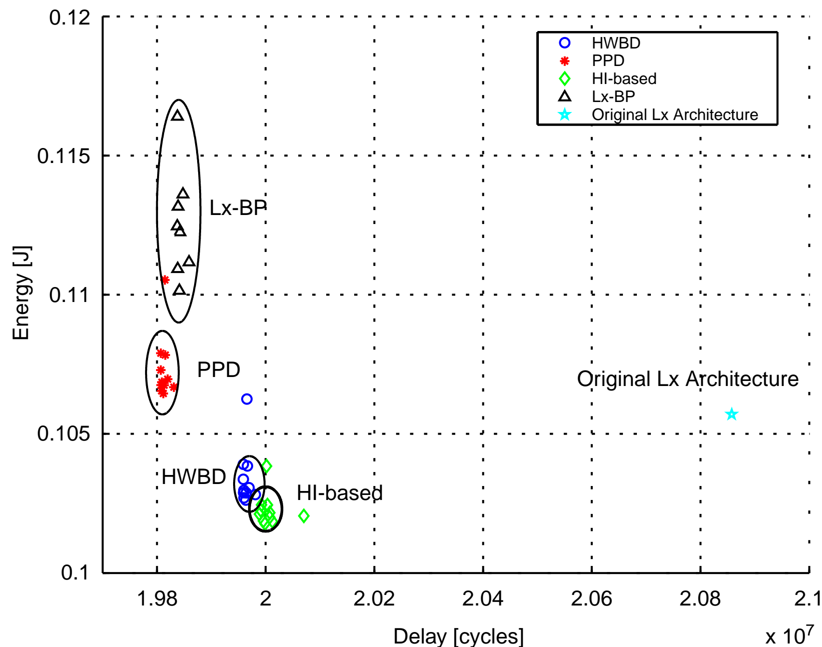

Fig. 1. Energy–Delay Scatter Plot for the simulated architectural configurations by varying branch predictor parameters and low-power optimizations.

Furthermore, we want to evaluate the efficacy of the HI-based method; that is, how many accesses to the branch predictor modules HI-based can save with respect to the accesses performed by traditional cycle-by-cycle dynamic prediction. So, we compare the architecture enhanced with dynamic prediction and HI-based, with the Lx-BP architecture. Table 1 shows the results about the branch predictor filter-access techniques in terms of *Misprediction Rate* and *Predictor Saved Accesses* for the simulated benchmarks. The *Misprediction Rate* is an index of the accuracy of the branch unit. Its value is determined as the fraction of mispredicted branches out of total executed branches. It is important to note that in the mispredicted branches, taken into account the not-detected taken-branch instructions too, which are statically predicted *not-taken*. The column *Predictor Saved Accesses* represents the saved accesses to predictor by the filtering techniques with respect to cycle-by-cycle accesses performed by the Lx-BP architecture.

We can observe in Table 1 that the most accurate method in terms of misprediction rate is the PPD. In fact, accuracy loss is only due to the effectively mispredicted branches and not to the misdetected branches. Table 1 shows that also the higher value for the fraction of *Predictor Saved Accesses* is related to the proposed *HI-based* (on average 93%). This characteristic and the limited energy cost overhead due to the HI introduction are the motivations for the high reduction in terms of energy shown in Fig. 1.

Fig. 2 shows the energy reduction and the performance speed-up for the proposed technique, HI-based, with respect to the Lx-BP architecture for each selected benchmark. The figure shows approximately the same trend in energy reduction for all the benchmarks we analyzed. The energy reduction ranges from 4%, in JPEGENC benchmark by using the Bimodal predictor, to 12%, in ADPCMENC benchmark by using GShare predictor. Concerning the performance speed-up,

Table 1
Performance of branch prediction filtering methods for the selected benchmarks

Benchmark	Branches	Misprediction rate (%)			Predictor saved accesses (%)		
	(%)	HWBD	PPD	HI-based	HWBD	PPD	HI-based
adpcmenc	14.2	12.0	11.7	0.7	87.3	86.8	93.8
adpcmdec	13.8	1.0	0.4	1.1	87.9	87.0	99.7
gsmenc	10.1	8.8	6.7	5.5	91.3	90.8	95.2
gsmdec	9.8	7.9	6.8	3.4	91.2	90.9	94.3
jpegenc	23.3	19.9	6.9	28.1	84.0	79.8	87.4
jpegdec	16.9	17.1	4.6	19.5	87.8	85.2	93.5
pegwitenc	14.7	8.8	6.1	11.0	86.7	86.9	91.8
pegwitdec	14.4	8.4	5.9	10.2	87.3	86.5	90.3
Average	14.6	10.5	6.2	10.0	87.9	86.8	93.3

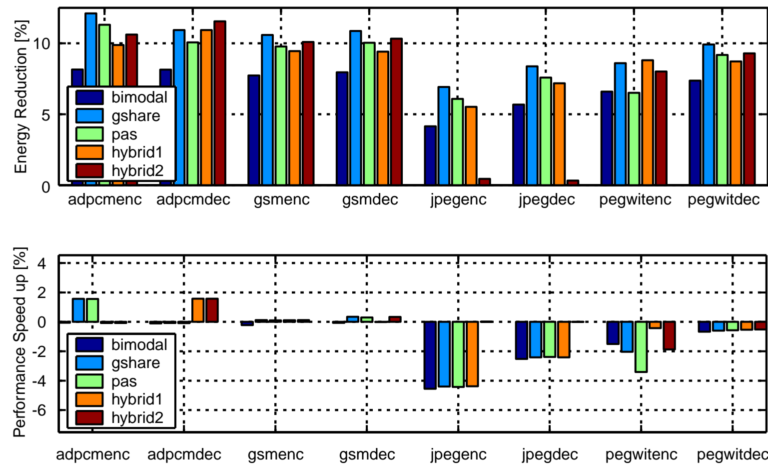

Fig. 2. Energy reduction and performance speed-up of HI-based technique, with respect to the Lx-BP architecture, for selected predictor schemes.

JPEGDEC and JPEGENC benchmarks show the worst behavior (−4.2% and −2.4% respectively). This behavior can be correlated with the misprediction rate shown in Table 1, that is 28.1% and 19.5% for JPEGDEC and JPEGENC, respectively. It can be explained if the structure of the simulated benchmark is considered. In particular, JPEGENC benchmark features a larger fraction of indirect branches (10% out of the total executed branches), which are harder to be dynamically predicted. The fraction of indirect branches in all the other benchmarks is less than 3%.

Table 2 (left column) shows the percentage of *hinted* branches out of the total number of executed branches for each benchmark. These data indicate how many branches are effectively detected by using HI-based approach. The number of the executed branches is split into two

Table 2
Percentages of *hinted* branches and Instruction Cache miss rates without/with *hints*

Benchmark	Hinted branches (%)	IC miss rate without hints (%)	IC miss rate with hints (%)
adpcmenc	85.05	1.20	1.33
adpcmdec	42.59	1.82	1.91
gsmenc	84.68	9.54	10.01
gsmdec	92.34	5.99	6.39
jpegenc	70.59	4.23	4.78
jpegdec	58.48	18.40	18.99
pegwitenc	84.71	1.66	1.85
pegwitdec	84.55	2.46	2.73

groups: the *hinted* branches, which are predicted through *hint* activation, and the others, statically predicted *not-taken* by the original processor policy. The percentage of *hinted* branches depends on the specific application, ranging from 43% to 92%, making a smaller or larger fraction of branches be predicted statically *not-taken*. How the *not-taken* prediction component impacts prediction accuracy depends on the specific benchmark structure. It can lead to low misprediction rate as for ADPCMDEC (featuring only 42% of *hinted* branches and 1% misprediction rate) or to a lot of mispredicted branches as for JPEGENC and JPEGDEC (see Table 1).

As stated before, the *HI-based* technique does not require the insertion of extra bundles, since either the HIs are inserted in the bundles in place of NOPs or the HIs are not inserted to avoid the increment of execution time. Nevertheless, the number of NOPs is reduced and that implies a less compressed VLIW code and a higher Instruction Cache miss rate. Table 2 (right column) shows the miss rate behavior for the selected set of benchmarks without and with the HIs insertion. For all the benchmarks we can note only a slight increment of the miss rate (less than 0.2%) due to the HIs insertion.

5. Conclusions

In this paper, a low-power branch prediction methodology for VLIW processors has been proposed. The proposed technique uses *hint* instructions to filter the accesses to the branch predictor. The combined effects of both static and dynamic information to predict a branch provide a significant energy reduction. Experimental results have shown that this technique can achieve an average access saving to the branch predictor of 93% with respect to a cycle-by-cycle access, which corresponds to 9% total processor energy reduction at the cost of 1% average performance loss. When HI-based technique is adopted in the Lx processor to enhance the original architecture with dynamic branch prediction, it gets a significant improvement of the energy-delay metric.

References

[1] N. Richardson, L.B. Huang, R. Hossain, J. Lewis, T. Zounes, N. Soni, The iCore 520-MHz syntheesizable CPU core, Micro (2003).
[2] R. Kessler, E. McLellan, D. Webb, The Alpha 21264 microprocessor architecture, in: Proceedings of ICCD, 1998.
[3] D.A. Patterson, J.L. Hennessy, Computer Architecture: A Quantitative Approach, third ed., Morgan Kaufmann, Los Altas, CA, 2003.
[4] J. Hoogerbrugge, Dynamic branch prediction for a VLIW processor, in: Proceedings of PACT, 2000, pp. 207–216.
[5] M. Monchiero, G. Palermo, M. Sami, C. Silvano, V. Zaccaria, R. Zafalon, Power-aware branch prediction techniques: a compiler-hints based approach for VLIW processors, in: Proceedings of GLSVLSI, Boston, MA, 2004.
[6] A. Chandrakasan, R. Brodersen, Minimizing power consumption in digital CMOS circuits, Proc. IEEE 83 (4) (1995) 498–523.
[7] M. Evers, T.-Y. Yeh, Understanding branches and designing branch predictors for high performance microprocessors, Proc. IEEE 89 (11) (2001) 1610–1620.
[8] S. McFarling, Combining branch predictors, in: WRL Technical Note TN-36, Digital Equipment Corporation, 1993.
[9] T.-Y. Yeh, Y.N. Patt, Two-level adaptive training branch prediction, in: Proceedings of MICRO-24, ACM Press, New York, 1991, pp. 51–61.
[10] C.H. Perleberg, A.J. Smith, Branch target buffer design and optimization, IEEE Trans. Comput. 42 (4) (1993) 396–412.
[11] B. Fagin, K. Russell, Partial resolution in branch target buffers, in: Proceedings of MICRO-28, November 1995, pp. 193–198.
[12] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, M. Stan, Power issues related to branch prediction, in: Proceedings of HPCA-8, ACM Press, New York, 2002.
[13] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: Proceedings of ISCA, 2000, pp. 83–94.
[14] G. Palermo, M. Sami, C. Silvano, V. Zaccaria, R. Zafalon, Branch prediction techniques for low-power VLIW processors, in: Proceedings of GLSVLSI, Washington DC, April 2003.
[15] P.-Y. Chang, E. Hao, T.-Y. Yeh, Y. Patt, Branch classification: a new mechanism for improving branch predictor performance, in: Proceedings of MICRO-27, San Jose, CA, USA, ACM Press, New York, 1994, pp. 22–31.
[16] D. Grunwald, D. Lindsay, B. Zorn, Static methods in hybrid branch prediction, in: Proceedings of PACT, Paris, France, October 1998.
[17] D.J. Lilja, Reducing the branch penalty in pipelined processors, IEEE Comput. 21 (7) (1988) 47–55.
[18] C. Lee, M. Potkonjak, W.H. Mangione-Smith, Mediabench: A tool for evaluating multimedia and communication systems, in: Proceedings of MICRO-30, 1997.
[19] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, F. Homewood, Lx: a technology platform for customizable VLIW embedded processing, in: Proceedings of ISCA, June 2000, pp. 203–213.
[20] S. Wilton, N. Jouppi, CACTI: an enhanced cache access and cycle time model, IEEE J. Solid-State Circuits 31 (5) (1996) 677–688.
[21] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, R. Zafalon, Energy estimation and optimization of embedded VLIW processors based on instruction clustering, in: Proceedings of DAC, June 2002, pp. 886–891.