

# Branch Prediction Techniques for Low-Power VLIW Processors

G. Palermo<sup>†‡</sup> M. Sami<sup>†</sup> C. Silvano<sup>†</sup> V. Zaccaria<sup>†</sup> R. Zafalon<sup>‡</sup>

<sup>†</sup>Politecnico di Milano, Dip. di Elettronica e Informazione, Milano, ITALY

<sup>‡</sup>STMicroelectronics, Agrate Brianza, Milano, ITALY

## ABSTRACT

Main goal of the paper is to introduce a branch prediction scheme suitable for energy-efficient VLIW (Very Long Instruction Word) processors aiming at reducing the energy associated with the prediction phase by filtering the accesses to the branch predictor block. To analyze the effectiveness of the proposed low-power branch prediction scheme, we combined it to some well-known dynamic branch prediction techniques suitable for VLIW processors. Experimental results have been carried out on Lx, a 4-issue VLIW architecture with 6-stage pipeline. The proposed solution implies a performance improvement of 7% on average and an average energy reduction of 15%.

## Categories and Subject Descriptors

C.1.0 [Processor Architectures]: General

## General Terms

Design, Performance

## Keywords

VLIW Processors, Branch Prediction, Low-Power Design

## 1. INTRODUCTION

Static as well dynamic branch prediction schemes could be applied to optimize system performance. Traditional prediction methods could be applied also to VLIW architectures [1] but, up to now, they haven't been implemented due to their relevant hardware overhead. In fact, most VLIW processors (such as Philips Trimedia or TI VLIW processor family) prefers to use static branch prediction techniques

associated with other hardware (e.g., *predication*) and software (e.g. *trace scheduling*) mechanisms [2,3]. Among static techniques, *branch delay slots* [2] are also used to overcome branch penalties in VLIW processors. In this scheme, the branch is delayed until the outcome is known and a fixed number of instructions after the branch are committed. The main drawback of these techniques is that they are applied at compile time on the basis of static or *profiled* information. In this paper, we will show that dynamic branch prediction techniques can be used for VLIW processors to obtain low energy consumption and increases performances at the expense of limited area overhead.

The main goal of this paper is to propose a low-power branch prediction architecture for VLIW processors. The basic idea consists of exploiting the pre-fetch decompression buffer of VLIW processors to filter the accesses to the branch predictor block. 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 predictor and consequently the related power dissipation. To analyze the effectiveness of the proposed low-power branch prediction scheme, we combined it to 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.

The paper is organized as follows. Section 2 introduces some previous works on branch prediction schemes as well as design space exploration. The target design space is introduced in Section 3, while the proposed low-power branch detector scheme is presented in Section 4. The experimental results carried out on the Lx VLIW architecture are shown in Section 5, and finally some concluding remarks and future evolutions of the work are reported in Section 6.

## 2. BACKGROUND

Branch prediction can be static or dynamic. Static branch prediction associates, with each branch of a program, a fixed prediction that is used every time the branch instruction is met in the stream. Dynamic branch prediction associates a variable prediction with each branch, that is updated based on a specific policy.

A dynamic branch prediction technique is usually composed of two interacting mechanisms [2]: a *branch outcome*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'03, April 28–29, 2003, Washington, DC, USA.  
Copyright 2003 ACM 1-58113-677-3/03/0004 ...\$5.00.

*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: if the branch is predicted as not taken, the program counter is normally incremented; otherwise, the branch target predictor is accessed to retrieve a prediction of the target of the branch. If the branch outcome predictor is accessed during or after the decode stage, then it is not necessary to access the branch target predictor since the target of the branch may be exactly known at that time. However if a branch prediction is needed during the instruction fetch stage (as is usually the case), a branch target predictor is needed since at that time, the target of the branch has not yet been decoded.

Several structures have been proposed for the implementation of branch outcome predictors [4]. A *Branch Prediction Buffer* (BPB) or *Branch History Table* (BHT) is a simple dynamic branch outcome predictor that is built above a small memory accessed using the lower bits of the PC of the current instruction. The memory can contain the previous outcome of the branch, stored with one bit, or a set of bits representing the state of a branch prediction state machine.

A *correlating branch predictor* [5] is a more complicated form of BPB, in which the prediction is conditioned also by the history of the previously executed branches (stored in a branch history register, BHR). A *Gshare* [2] predictor is a dynamic two bit predictor that is indexed by a XOR between the branch address and the branch history. This is also called *local/global* branch predictor [6]. In this case, the outcome of the XOR is used to access a *pattern history table* that contains the state of a two bit saturating counter. In [7], the branch history is used alone to access the pattern history table, that is represented as a generic state machine. This branch predictor is also called *GAg* (or global/adaptive/global). A *GAs* (or global/adaptive/set) predictor is very similar to the Gshare except for the fact that the branch address is concatenated with the branch history register to access the pattern history table.

A *tournament branch predictor* [2] exploits a saturating counter to choose between two different predictors (say A and B). Generalization of such type of predictor are also called *hybrid* predictors [8]. One of the most used implementation of branch target predictors is the *branch target buffer* (BTB) [2]: a branch prediction cache storing the predicted address for the next instruction after the branch. Tags are used in the cache to select the correct branch prediction. BTB entries can be managed with the same policy of the BPBs, i.e., with a simple 1-bit history (BTB<sub>1</sub>) or an n-bit saturating counter (BTB<sub>n</sub>) [9].

A work concerning branch prediction and power/performance exploration recently appeared in the literature [10]. In this work, several branch prediction organizations for superscalar processors have been analyzed from the power/performance standpoint.

### 3. TARGET DESIGN SPACE

The first goal of this paper is to present a power/performance characterization of the design space composed of several branch prediction mechanisms suitable for a *Very Long Instruction Word* (VLIW) processor. Our target VLIW architecture has been directly derived from the Lx processor, a

scalable and customizable 4-issue (VLIW) pipeline architecture. Lx is a family of VLIW processors for embedded systems jointly designed by STMicroelectronics and Hewlett-Packard Labs [11]. The original Lx static branch predictor (that predicts always *branch not taken*) has been substituted by a configurable dynamic branch predictor. In our exploration, the branch prediction schemes used by the configurable branch predictor are based on *BHT*, *GAs* and *BTB* with two different replacement policies.

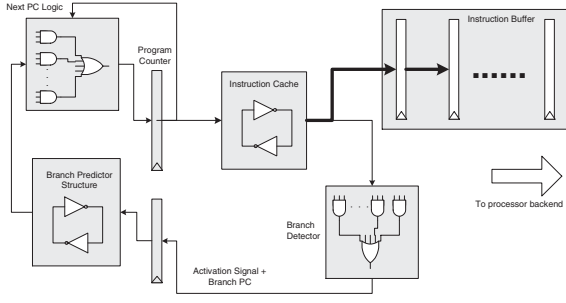
For the *BHT* and *GAs* predictors, the design space is composed of the number of entries in the history table that can assume the following values: 16, 32, 64, 128, 256, 512, 1024. For the two predictors based on Branch Target Buffer (*BTB*<sub>1</sub> and *BTB*<sub>2</sub>), we have chosen a very large design space, since the architecture of the Target Buffer is very similar to a cache. The *BTB* design space is composed of size (16B to 1KB), degree of associativity (Direct Mapped to Fully Associative) and number of bits to address the BTB (4 to 28). The variable size of the address of the BTB could imply some inaccuracies in the branch prediction. More in detail, the address sent to the BTB can be decomposed into two fields: index and tag. The index is used for referencing the correct cache line while the tag is used to avoid aliasing problems associated with branches with the same index. Varying the size of the address, in our case, means varying the size of the tag that is compared when accessing the cache line. The reduction of the size of the comparison implies an energy saving but a loss of accuracy of the branch predictor [12].

### 4. LOW-POWER OPTIMIZATION

The binary code of a 'pure' VLIW processor is usually quite full of NOP operations, since the compiler cannot fill all the available parallel slots with a corresponding useful operation. To solve this problem, several compression mechanisms have been proposed; for example the TI TMS 320C6x uses a flexible instruction encoding that is used to indicate parallel and serial operations within a word; during the instruction fetch phase, only parallel operations are sent simultaneously to the back-end of the pipeline. The Lx processor [11] has a very similar mechanism to decompress long instructions, i.e., a small prefetch buffer that has the size of two long instructions.

For our purposes, the Lx architecture has been modified by substituting the fixed prefetch buffer with a buffer of variable length (see Figure 1). This buffer is used to extract, in the instruction fetch stage, the next bundle to be decoded. This mechanism is similar to an instruction queue, in which each element has a variable size.

To reduce the energy dissipation due to the use of the Branch Predictor, we implemented a selective technique to access this block only when a branch occurs. Figure 1 shows the implementation of this technique. When operations are fetched from the instruction cache and stored in the decompression buffer (Instruction Buffer), a *Branch Detector* (BD) partially pre-decodes the instructions to find a possible branch. The implementation of the BD is dependent on the encoding of the branch instruction. If a branch is detected, its PC is sent to the branch prediction block in order to activate the branch prediction, otherwise the instructions in the decompression buffer are sequentially executed. The implementation of the BD block (see Figure 1) introduces a delay corresponding to two logic levels to the critical path



**Figure 1: The proposed branch detector (BD) for pipelined VLIW processors**

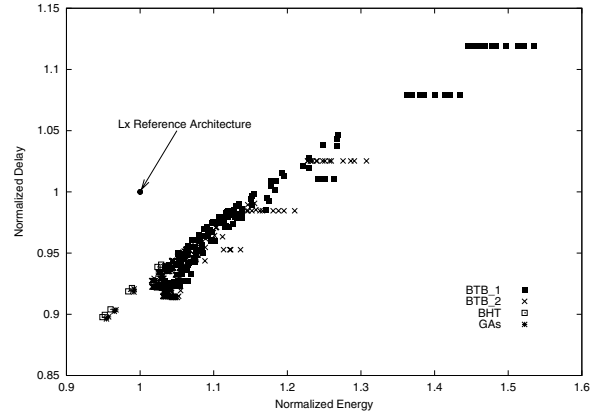
of the fetch stage. When the decompression unit is empty, branches are directly sent to the decode unit and the branch detector cannot activate the branch predictor block. This implies an increased delay due to the de-activation of the branch predictor.

## 5. EXPERIMENTAL RESULTS

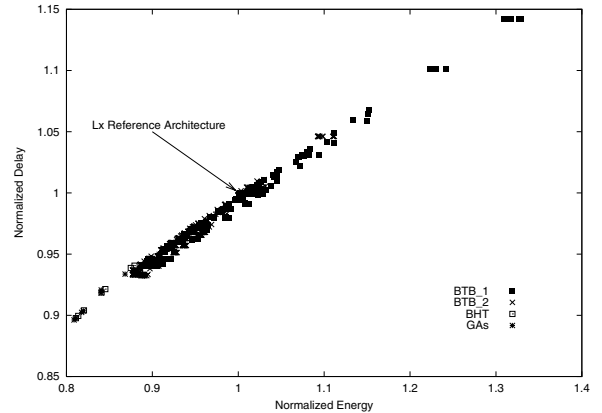
In this section, we present the power saving results obtained by introducing the Branch Detector mechanism proposed in Section 4 for the target design space introduced in Section 3. The target system under consideration is based on the Lx microprocessor. The applications used for the characterization are a set of multimedia benchmarks taken from the Mediabench Suite [13]. The experimental results have been obtained by using an *augmented* Instruction Set Simulator of the Lx architecture in which the power models of the Lx core [14] and of the various branch predictors have been plugged in. In particular, the ISS is composed of the following modules:

- An instruction-accurate timing estimation module containing the power model of the core. According to [14], the ISS power model has an average error of 1.9% with a standard deviation of 5.8% with respect to gate level power estimation of the reference design. The power consumption of the processor core running at 200 MHz ranges between 200 [mW] and 700 [mW].
- An instruction decompressor simulator that simulates the issue of operations to the decode phase and the delay due to the decompression phase. This module contains also the configurable power model of the decompressor.
- A configurable branch predictor simulator and power model derived from the cache model described in [15].

As evaluation metrics, we have chosen the relative energy of the system (measured as the ratio between the actual configurations energy and the reference Lx energy) and the relative delay of the system (measured as the ratio between the execution clock cycles of the actual configurations and those of the reference Lx). In the rest of the paper, percentage values representing speed-up or energy reduction must be intended as referred to the reference configuration of the Lx processor that, as we said before, has only a static branch predictor.



**Figure 2: Energy delay scatter plot**



**Figure 3: Energy delay scatter plot. (BD enabled)**

### 5.1 DSE Results - Without Branch Detector

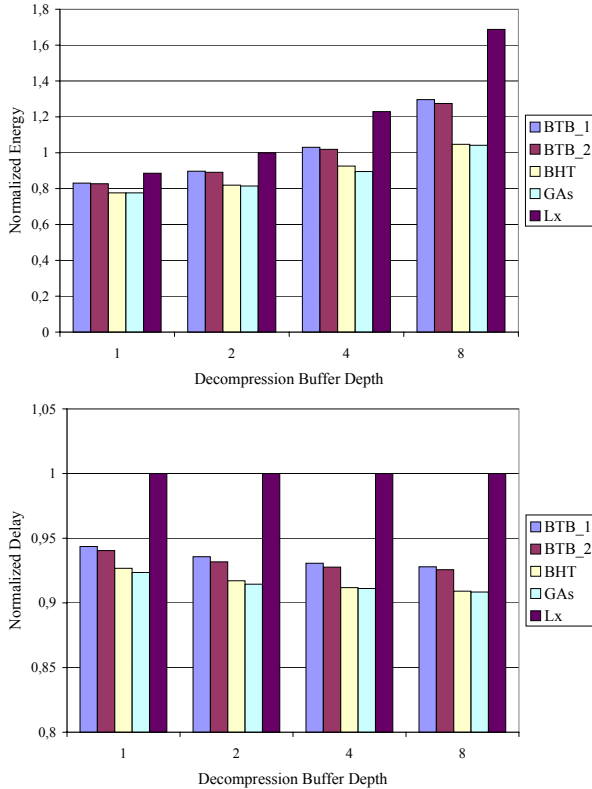
In this section, we show the overall results obtained by experimenting with the exploration method presented in Section 3 (without the proposed BD mechanism). Figure 2 shows the two dimensional scatter plot of the average relative delay and the relative energy of the different branch predictor configurations. As can be seen, for the specific design space, there are only few points representing the optimal solution (*Pareto Curve*). These points are associated with the GAS and BHT branch predictors, that are also characterized by the highest accuracy in terms of prediction and consequently a performance improvement. The reason of this is due to the fact that many of the branch predictors we explored are very small in terms of area and power consumption, thus they imply a negligible impact on the energy consumption. In fact, as can be seen, the energy behavior of the different configurations depends proportionally on the delay of the system. Figure 2 shows also the energy and delay configurations of the original architecture (indicated as *Lx reference architecture*). As can be seen, the same energy consumption of the original architecture can be obtained, by using some of the predictors, with a decreased value of the delay.

### 5.2 DSE Results - With Branch Detector

The same design space exploration has been repeated for the Branch Detector technique. Figure 3 shows the results

of the exploration when the proposed BD is used. We can observe a behavior similar to those shown in Figure 2: in this case, the scatter plot is translated along the energy axis.

Comparing Figure 3 with Figure 2, we can note that the number of predictors showing a better behavior in terms of both energy and delay with respect to the Lx reference architecture is significantly increased. This is justified by the fact that, by using the BD, we can perform a filtered access to the branch predictor, thus resulting in an overall energy saving.



**Figure 4: Behavior of the energy and delay of the system when changing the decompression buffer depth**

Figure 4 shows the behavior of the energy and delay of the system when the decompression buffer depth is varied (the size of the predictors is of 256 entries). As can be seen, the GAs branch prediction scheme is always the best technique both in terms of energy and delay, while the reference configuration is always the worst for each value of the depth.

Regarding the four architectures analyzed, a trade-off exists between energy and delay. The highest energy saving (20%) can be reached with one-entry decompression buffer with GAs with 5% of performance improvement. The highest performance improvement (7%) can be obtained with eight-entry decompression buffer with GAs, without energy savings.

In conclusion, the results have shown an overall energy reduction demonstrating that the area(energy) overhead introduced by the BD block has been traded off by the performance and energy savings of the overall processor.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, a low-power branch prediction technique has been proposed to exploit the pre-fetch decompression buffer in VLIW processors. Experimental results have shown an average energy saving of 15% while ensuring an average performance improvement of 7%. As future evolutions of the present work, we can envision to combine the proposed analysis of branch prediction architectures with static prediction techniques for VLIW architectures. The combined effects of both static and dynamic techniques to predict branch behavior could provide better performance improvements while reducing power dissipation.

## 7. REFERENCES

- [1] Jan Hoogerbrugge. Dynamic branch prediction for a VLIW processor. In *IEEE PACT*, pages 207–216, 2000.
- [2] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, 2nd edition, 1996.
- [3] P. Geoffrey Lowney, Stefan M. Freudenberger, Thomas J. Karzes, W. D. Lichtenstein, Robert P. Nix, John S. O’Donnell, and John C. Ruttenberg. The Multiflow Trace Scheduling compiler. *The Journal of Supercomputing*, 7(1-2):51–142, 1993.
- [4] James E. Smith. A study of branch prediction strategies. In *Conference proceedings of the eighth annual symposium on Computer Architecture*, pages 135–148. IEEE Computer Society Press, 1981.
- [5] Shien-Tai Pan, Kimming So, and Joseph T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 76–84. ACM Press, 1992.
- [6] M.-C. Chang and Y.-W. Chou. Branch prediction using both global and local branch history information. *Computers and Digital Techniques, IEE Proceedings-*, 149(2):33–38, March 2002.
- [7] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th annual international symposium on Microarchitecture*, pages 51–61. ACM Press, 1991.
- [8] Kai Wang and Manoj Franklin. Highly accurate data value prediction using hybrid predictors. In *International Symposium on Microarchitecture*, pages 281–290, 1997.
- [9] C.H. Perleberg and A.J. Smith. Branch target buffer design and optimization. *IEEE Transactions on Computers*, 42(4):396–412, 1993.
- [10] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proc. of the 2002 International Symposium on High-Performance Computer Architecture*. ACM Press, 2002.
- [11] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, and F. Homewood. Lx: a technology platform for customizable vliw embedded processing. In *Proceedings of the International Symposium on Computer Architecture*, pages 203–213, June 2000.
- [12] Barry Fagin and Kathryn Russell. Partial Resolution in Branch Target Buffers. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 193–198, Ann Arbor, Michigan, November 29–December 1, 1995.
- [13] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating multimedia and communication systems. In *Proceedings of Micro 30*, 1997.
- [14] Andrea Bona, Mariagiovanna Sami, Donatella Sciuto, Cristina Silvano, Vittorio Zaccaria, and Roberto Zafalon. Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering. In *Proceedings of the 39th Design Automation Conference DAC’02*, pages 886–891, June 2002.
- [15] S. Wilton and N. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, 1996.