Power-Aware Branch Prediction Techniques: A Compiler-Hints Based Approach for VLIW Processors

M. Monchiero† G. Palermo† M. Sami† C. Silvano† V. Zaccaria‡ R. Zafalon‡

†Politecnico di Milano, Dip. di Elettronica e Informazione, Milano, ITALY
‡STMicroelectronics, Agrate Brianza, Milano, ITALY

ABSTRACT

Main goal of the paper is introducing a dynamic branch prediction scheme suitable for energy-aware VLIW (Very Long Instruction Word) processors. The proposed technique is based on a *compiler hint* mechanism to filter the accesses to the branch predictor blocks. Experimental results have been carried out on Lx/ST200, an industrial 4-issue VLIW architecture. We gathered two sets of results: First, by introducing the proposed low-power branch prediction technique in the Lx processor, which features fully static branch prediction, a significant improvement of the energy-delay metric has been observed. Second, we evaluated filtering efficacy of the proposed method and we found that it gets an access reduction to the branch prediction unit of 93% with respect to a processor directly derived from Lx, featuring cycle-by-cycle prediction, corresponding to an average 9% energy reduction of the whole processor power budget.

Categories and Subject Descriptors

C.1.0 [**Processor Architectures**]: General

General Terms

Design, Performance

Keywords

VLIW Processors, Branch Prediction, Low-Power Design

1. INTRODUCTION

Control dependencies between instructions prevent pipelined processors from meeting maximum performance. Specifically for pipelined *ILP (Instruction Level Parallelism)* processors, as the amount of parallelism grows, control dependencies become one of the limiting factors to increase the system performance. Branch prediction techniques can reduce performance degradation due to branch instructions.

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 main goal of the present paper is to propose a low-power branch prediction architecture suitable for VLIW processors. In the paper, we propose 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. 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.

The paper is organized as follows. Section 2 introduces some previous works. The proposed low-power branch prediction technique based on Hint Instructions is presented in Section 3. The target architecture and the related set of experimental results are shown in Section 4. Finally some concluding remarks and future evolutions of the work are reported in Section 5.

2. BACKGROUND

Many works have been targeted at reducing the performance degradation due to control dependencies, proposing several branch prediction techniques [1] [2]. Branch prediction can be static or dynamic. Static branch prediction associates with each branch a fixed prediction that is used once 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 com-

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

posed of two interacting hardware modules [2]: 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 (usually implemeted as a *Branch Target Buffer* (*BTB*) [2]), computes the *taken* address of the branch. Hoogerbrugge [3] compares the typical dynamic branch prediction techniques for superscalar processors against *ad hoc* techniques for VLIW architectures showing that branch prediction techniques designed *ad hoc* for VLIW processors can reduce the misprediction rate significantly.

Microarchitectural power reduction techniques are usually combined with technology-level transformations to obtain circuits with reduced energy dissipation and leading-edge performance [4]. Research on branch prediction and power/performance exploration appeared only recently in the literature. In [5], the authors analyze several branch prediction organizations for superscalar processors from the power/performance perspective. 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. A similar work has been presented in [6], where the authors introduce a branch prediction scheme for a low-power VLIW processor. 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.

The present paper represents a further step on the way of designing low-power dynamic branch prediction techniques in a VLIW architecture, exploiting a compiler assisted predictor.

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 [6].

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 [6] [5]. 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 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 and manages `target` and `direction` fields, if they are encoded in the *HI*.

Peculiar VLIW code features can be exploited to make the *HI* insertion really efficient. Usually a VLIW compiler cannot fill every long instruction (namely, *bundle*) with useful operations [2]. 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 (on average 20M instructions executed for each benchmark) taken from the Mediabench Suite [7].

Our target architecture has been directly derived from the Lx processor, a scalable and customizable 4-issue (VLIW) pipeline architecture, whose implementations are the CPUs of the ST210/ST220 cores family for embedded systems (by STMicroelectronics and Helwett-Packard Labs) [8]. Lx architecture is a 'pure' VLIW with a very short pipeline, only six stages. The original Lx architecture has only a static branch prediction policy, *always not taken*. Branch instructions are early solved in the *decode* stage (second stage), so the misprediction penalty is only *one cycle stall*.

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

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

Benchmark	Branches [%]	Misprediction Rate [%]			Predictor Saved Accesses [%]			Hinted Branches [%]	Miss Rate without hints [%]	Miss Rate with hints [%]
		HWBD	PPD	HI-based	HWBD	PPD	HI-based			
adpcmenc	14.172	12.063	11.73	0.7147	87.354	86.823	93.797	85.051	1.20	1.33
adpcmdec	13.774	1.0378	0.39091	1.0872	87.86	87.049	99.723	42.593	1.82	1.91
gsmenc	10.051	8.7913	6.7143	5.5038	91.299	90.804	95.206	84.683	9.54	10.00
gsmdec	9.7331	7.9546	6.7955	3.479	91.191	90.927	94.271	92.332	5.99	6.39
jpegenc	23.339	19.867	6.9001	28.082	84.002	79.82	87.398	70.586	4.23	4.78
jpegdec	16.889	17.091	4.6609	19.524	87.776	85.239	93.476	58.479	18.40	18.99
pegwitenc	14.748	8.7217	6.1443	11.033	86.713	86.927	91.838	84.713	1.66	1.85
pegwitdec	14.405	8.3684	5.9297	10.242	87.303	86.52	90.312	84.55	2.46	2.73
Average	14.639	10.487	6.1582	9.958	87.937	86.764	93.253	75.37	5.66	6.00

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. The branch outcome predictors chosen for the validation of the methodology, with the relative configuration of the parameters, are: Bimodal predictor (64B), GShare (256B, 1KB, 4KB, 16KB), PAs (192B, 640B, 2.2KB, 4.1KB), Hybrid1 Bimodal/PAs (1.3KB), Hybrid2 GShare/PAs (1.8KB).

The experimental results are obtained by using an in-house custom trace-driven tool called BPSIM (Branch Prediction SIMulator). The power models for the configurable branch predictor and detector schemes derived from Cacti [9] and Wattch models [10], configured for a 0.25μm implementation running at 250MHz, 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 [11].

We show two different sets of results: First, 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, 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.

The overall results, obtained by applying the branch selection method that we propose (HI-based) to the Lx processor, are shown in Figure 1, where there is 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 really 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 speedup with respect to the original Lx architecture, since some branches can't 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. It can be observed that a significant performance speedup of the Lx architecture enhanced by HI-based dynamic branch prediction with respect to the original Lx architecture is achieved. We measured, on the selected set of simulated benchmarks, a speedup of 4.4% on average and 7.2% at maximum.

Furthermore, we want to evaluate the efficacy of the HI-based method; that is, how many accesses to the branch predictor modules can be saved 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 of 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

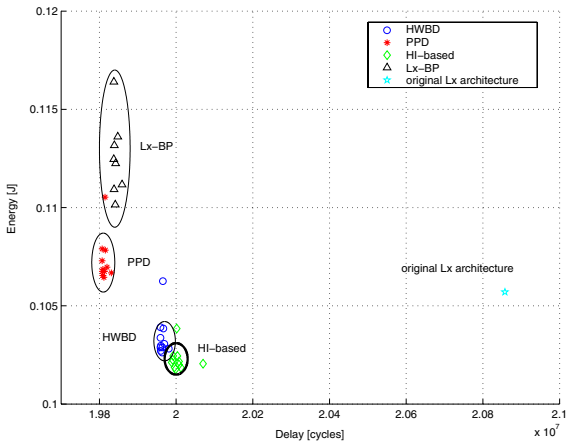

Figure 1: Energy-Delay Scatter Plot for the simulated architectural configurations by varying branch predictor parameters and low-power optimizations (PPD, HWBD and HI-based)

branches out of total executed branches. It is important to note that in the mispredicted branches are 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 (similar to NOP instructions) due to the *HI* introduction are the motivations for the high reduction in terms of energy shown in Figure 1.

Table 1 shows also the percentage of *hinted* branches out of the total number of executed branches and the Instruction Cache (IC) miss rate behavior when *hints* are inserted, for each benchmark. The first set of these data (Hinted Branches) indicates how many branches are effectively detected by using HI-based approach. The number of the executed branches is split into two 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).

Since the *HI-based* technique does not require the insertion of extra bundles, the total number of NOPs is reduced and that implies a less compressed VLIW code and a higher Instruction Cache miss rate. However, the last two columns of Table 1 shows that, for all the benchmarks, we can note only a slight increment of the miss rate, due to *hints* insertion.

Figure 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 speedup, 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 28.08% and 19.52% for JPEGDEC and JPEGENC respectively. 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%.

5. CONCLUSIONS AND FUTURE WORK

In this paper, a low-power branch prediction methodology for VLIW processors has been proposed. The proposed

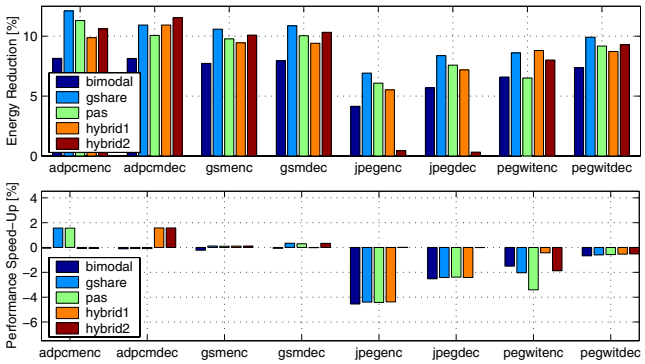

Figure 2: Energy reduction and performance speed-up of *HI-based* technique for each benchmark, with respect to the Lx-BP architecture, for a set of branch predictor schemes (Bimodal, GShare, PAs, Hybrid1 and Hybrid2)

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.

As future evolutions of the present work we are evaluating other architectures and related simulation environments, as well as compiler-based techniques to extensively support low-power dynamic branch prediction techniques.

6. REFERENCES

[1] M. Evers and T.-Y. Yeh. Understanding Branches and Designing Branch Predictors for High Performance Microprocessors. *Proc. of the IEEE*, November 2001.

[2] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3rd edition, 2003.

[3] Jan Hoogerbrugge. Dynamic branch prediction for a VLIW processor. In *IEEE PACT*, pages 207–216, 2000.

[4] A. Chandrakasan and R. Brodersen. Minimizing power consumption in digital cmos circuits. *Proc. of the IEEE*, 1995.

[5] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proc. of HPCA-8*. ACM Press, 2002.

[6] G. Palermo, M. Sami, C. Silvano, V. Zaccaria, and R. Zafalon. Branch Prediction Techniques for Low-Power VLIW Processor. In *Proc. of GLSVLSI'03*, Washington D.C, April 2003.

[7] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating multimedia and communication systems. In *Proceedings of Micro 30*, 1997.

[8] P. Faraboschi, G. Brown, J. Fisher, G. Desoli, and F. Homewood. Lx: a technology platform for customizable vliw embedded processing. In *Proc. of ISCA'00*, June 2000.

[9] S. Wilton and N. Jouppi. CACTI:An Enhanced Cache Access and Cycle Time Model. *IEEE JSSC*, 1996.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proc. ISCA'00*, pages 83–94, 2000.

[11] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon. Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering. In *DAC'02*, June 2002.