Advanced Branch Predictors for Soft Processors

Di Wu, Jorge Albericio and Andreas Moshovos
Electrical and Computer Engineering Department
University of Toronto
peterwudi.wu@utoronto.ca, jorge@eecg.toronto.edu, moshovos@eecg.toronto.edu

***Abstract*—The abstract goes here.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly popular to be used in embedded systems. Such designs often employ one or more embedded microprocessors, and there is a trend to migrate these microprocessors to the FPGA platform. Although these soft processors cannot match the performance of a hard processor, soft processors have the advantage that the designers can implement the exact number of processors to efficiently fit the application requirements.

Current commercial soft processors such as Altera's Nios II [1] and Xilinx's Microblaze [2] are in-order pipelines with five to six pipeline stages. These processors are often used for less computation-intensive applications, for instance, system control tasks. To support more compute-intensive applications, a key technique to improve performance is branch prediction. Branch prediction has been extensively studies, mostly in the context of application specific custom logic (ASIC) implementations. However, naïvely porting ASIC-based branch predictors to FPGAs results in slow and/or resource-inefficient implementations since the tradeoffs are different for reconfigurable logic. Wu et al. [3] have shown that a branch predictor design for soft processors should balance its prediction accuracy as well as the maximum operating frequency. They proposed an FPGA-friendly minimalistic branch prediction implementation *gRselect* for Altera's highest performing soft-processor Nios II-f.

Wu et al. limits the hardware budget of the gRselect predictor to one M9K Block RAM [4] on Altera Stratix IV devices, which is the same hardware budget as Nios II-f. Such a small hardware budget prohibits more elaborated and accurate branch prediction schemes such as perceptron [5] and TAGE [6]. This work looses the hardware budget constraint and investigates FPGA-friendly implementations of perceptron and TAGE predictors. This work also assumes a pipelined processor implementation representative of Altera's Nios II-f for comparison versus gRselect.

Specifically, this work makes the following contributions: (1) It studies the FPGA implementation of the perceptron predictor and TAGE predictor, including many optimizations to improve maximum frequency of these predictors. (2) It shows that comparing the branch direction prediction accuracy over the benchmarks versus gRselect, perceptron is $\tilde{1}$% worse while TAGE is $\tilde{1}$% better, assuming these predictors can be accessed in a single cycle. (3) It discovered that TAGE is too slow for single-cycle access, so in reality TAGE can only provide a prediction in two cycles. This work proposes an overriding predictor that uses a simple base predictor to provide a base prediction in the first cycle, then override that decision should TAGE disagree with the base predictor in the second cycle. It shows that the overriding TAGE predictor achieves 5.2% better instruction throughput over gRselect.

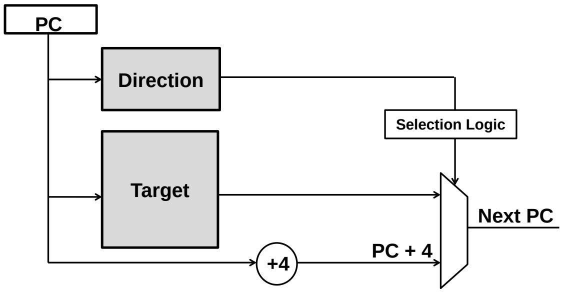

Fig. 1: Canonical Branch Predictor.

II. Background and Goals

Fig. 1 shows the organization of a typical branch predictor comprising a direction predictor and a target predictor. The predictor operates in the fetch stage where it aims to predict the program counter (PC), that is the address in memory, of the instruction to fetch in the next cycle using the current instruction's PC and other dynamically collected information. The direction predictor guesses whether the branch will be taken or not. The target predictor guesses the address for predicted as taken branches and function returns respectively. The multiplexer at the end selects based on the branch type and the direction prediction whether the target is the fall through address (PC+4 in Nios II) or the target predicted by the target predictor. Since, at this point in time, the actual instruction is not available in a typical ASIC implementation, it is not directly possible to determine whether the instruction is a return, a branch, or some other instruction. Accordingly, a Selection Logic block uses either pre-decode information or a PC-based, dynamically populated lookup table to guess which target is best to use. With the latter scheme, when no entry exists in the lookup table, some default action is taken until the first time a branch is encountered. Once the branch executes, its type is stored in the lookup table where it serves to identify the branch type on subsequent encounters. This scheme is not perfectly accurate due to aliasing.

A. Design Goals

This work aims to implement perceptron and TAGE that has high operating frequency as well as accuracy to maximize

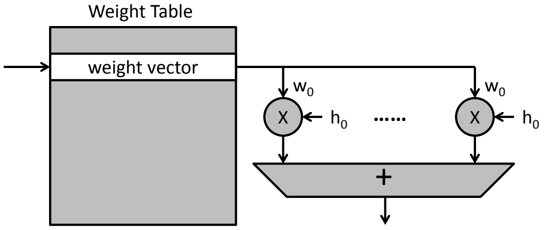

Fig. 2: The preceptron branch predictor.

execution performance. As section **??** will show, a single-cycle TAGE is prohibitively slow. Therefore, this work proposes an overriding TAGE predictor that produces a base prediction in one cycle and overrides that decision with a better prediction if the later prediction differs from the base prediction. Since perceptron and TAGE both requires large storage spaces, on-chip resources is not the main concern of this paper.

III. Perceptron Predictor

The perceptron predictor use vectors of weights (i.e., perceptrons) to represent correlations among branch instructions [5]. Fig. 2 shows the structure of a perceptron predictor.

Perceptron uses a table to store vectors of weights. When making a prediction, a weight vector is loaded from the table, and each weight is multiplied by the corresponding global branch history (i.e. 1 if taken and -1 if not-taken) from the GHR. Finally, the multiplication results are summed up, and perceptron predicts taken if the sum is positive, and not taken otherwise.

Section **??** introduced that perceptron predictor maintains vectors of weights in a table. It produces a prediction through the following steps: (1) a vector of weight (i.e., a perceptron) is loaded from the table. (2) multiply the weights with their corresponding global history (1 for taken and -1 for not-taken). (3) sum up all the products, predict taken if the sum is positive, and not-taken otherwise. Each of these steps poses difficulties to map to the FPGA platform. The rest of this section addresses these problems.

A. Perceptron Table Organization

Each weight in a perceptron is typically 8-bit wide, and perceptron predictors usually use at least 12-bit global history [5]. The depth of the table, on the other hand, tends to be relatively shallower (e.g. 64 entries for 1KB hardware budget). This requires a very wide but shallow memory, which does not map well to BRAMs on FPGAs. For example, the widest configuration that a M9K BRAM on Altera Stratix IV chip is 36-bit wide times 1k entries. If we implement the 1KB perceptron from [5], which uses 96-bit wide perceptrons 12-bit global history

As Fig. **??** shows, 64-entry

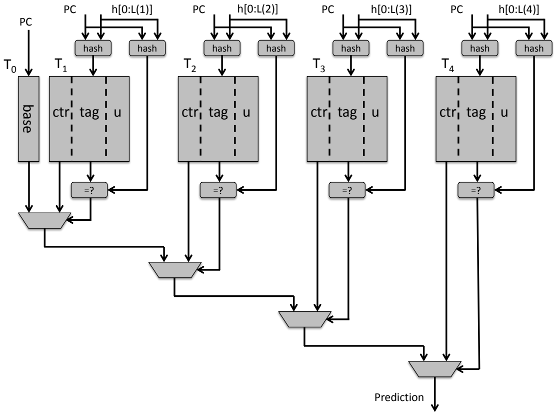

Fig. 3: A 5-component TAGE branch predictor.

B. Multiplication

C. Adder Tree

IV. TAgged GEometric history length Branch Predictor (TAGE)

The TAGE predictor features a bimodal predictor as base predictor T_0 to provide a basic prediction and a set of M tagged predictor components T_i [6]. These tagged predictor components T_i, where $1 \leq i \leq M$, are indexed with hash functions of the branch address and the global branch/path history with various length. The global history lengths used for computing the indexing functions for tables T_i form a geometric series, i.e., $L(i) = (int)(\alpha^{i\text{-}1} \times L(1) + 0.5)$. TAGE achieves its high accuracy by utilizing very long history lengths

Fig. 3 shows a 5-component TAGE predictor. Each table entry has a 3-bit saturating counter *ctr* for prediction result, a *tag*, and a 2-bit useful counter *u*. The indices of the tables are produced by hashing PC and global history with various lengths. A valid prediction result from each table is provided only on a tag match (i.e. a hit). The final prediction of TAGE comes from the hitting tagged predictor component that uses the longest history.

V. Conclusion

The conclusion goes here.

References

[1] Altera Corp., “Nios II Processor Reference Handbook v9.0,” 2009.

[2] *MicroBlaze Processor Reference Guide*, Xilinx Inc., July 2012.

[3] D. Wu, K. Aasaraai, and A. Moshovos, “Low-cost, high-performance branch predictors for soft processors,” in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013.

[4] *TriMatrix Embedded Memory Blocks in Stratix IV Devices*, Altera Corp, Dec. 2011.

[5] D. A. Jimenez and C. Lin, “Dynamic Branch Prediction with Perceptrons,” in *Intl’ Symposium on High-Performance Computer Architecture*, January 2001.

[6] A. Seznec and P. Michaud, “A case for (partially) tagged geometric history length predictors,” in *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2006.

[7] *Nios II Processor Reference*, Altera Corporation, May 2011.

[8] A. Seznec, “The O-GEHL Branch Predictor,” in *The 1st JILP Championship Branch Prediction Competition*, December 2004.