Evaluating Branch Predictor Configurations for a MIPS-like Pipeline

Evaluating Branch Predictor Configurations for a MIPS-like Pipeline

Fredrik Brosser, Karthik Manchanahalli Rajendra Prasad, Alen Bardizbanyan, and Per Larsson-Edefors
Chalmers University of Technology, Gothenburg, Sweden
frebro@student.chalmers.se, prasadk@student.chalmers.se, alenb@chalmers.se, perla@chalmers.se

***Abstract*—In this report, we investigate the implementation and efficiency of different types of branch predictors. A configurable VHDL model of a branch predictor unit, composed of a branch direction predictor and a branch target buffer, has been implemented. In order to make informed hardware decisions, different branch predictor configurations are simulated using the open source SimpleScalar simulator and the MiBench benchmark suite. The target architecture is a 7-stage 32-bit MIPS-based pipeline with two instruction fetch stages.**

I. Introduction

Conditional branch instructions in a pipelined processor give rise to performance penalties. Since they are evaluated in the execution stage of the pipeline, the processor potentially has to discard instructions in the previous stages of the pipeline or stall the fetching of new instructions [1]. However, in real-life applications, branches often display patterns or strong tendencies, such as loops, which can be detected and used by a branch predictor. Furthermore, branches often branch to the same target repeatedly. By storing branch targets in a Branch Target Buffer (BTB), it is possible to predict the target address. Branch direction predictors attempt to predict which way a branch will take, i.e., taken or not taken, before the branch condition is evaluated, potentially saving cycles. In doing so, predictors can consider the global branch pattern history, the local history for separate branch instructions, or a combination of these. The BTB is essentially organized in the same manner as a cache memory, storing past branch target address in a table. This table can be organized in a set-associative manner, using some (re)placement policy to decide where a particular entry is to be placed.

By using branch direction prediction and branch target buffering, we can improve the performance of the pipeline. Due to the earlier mentioned typical branch behavior, predictors can be highly accurate (90 %) when executing real-life applications. When designing a BTB, the table size, the associativity, and the replacement policy are the main points of interest [2].

In this report, we investigate the accuracy and impact in terms of pipeline performance of different branch predictor configurations using SimpleScalar simulations [3] and a proven set of benchmarks (MiBench [4] and SPEC [5]). Two main types of direction predictors, the *two-level adaptive predictor* and the simpler *bimodal predictor*, are in focus in this work. We examine the trade-offs involved in designing the branch direction predictor and the BTB, specifically between power and area, and performance. In evaluating branch predictor configurations we consider performance in terms of execution time, time taken to execute a benchmark, expressed in clock cycles. Execution time is then normalized with respect to the execution time of a prefect predictor and this normalized metric is referred to as the execution time ratio. Lower the execution time ratio, better the performance of the branch predictor. We also consider prediction accuracy, area, and power of the synthesized branch predictor design.

The remainder of this report is organized as follows: Section II briefly reviews work related to our paper. Section III introduces the simulation environment and benchmarks used to evaluate branch predictor configurations. Section IV describes the implementation of the branch direction predictor and the BTB. Section V details the verification process of the branch predictor. Section VI describes the synthesis results and important design trade-offs. Finally, in Section VII we give our conclusions and briefly mention future work.

II. Related Work

The design of the branch predictors is a well explored area in the processor community, with notable research being done in the early 1990's. Trade-offs related to the BTB design and implementations were explored early on [6]. Yeh et al. [7], [8] focused on describing and investigating two-level adaptive branch predictors and their implementation, discussing the possible variations of two-level branch predictors and defining the different types according to the manner in which the table content is kept (Global-Private-Set). This classification yielded variations of the two-level adaptive predictor, e.g., GAg (Global-And-global) or PAg (Private-And-global). In the remainder of this paper, we shall use these notations [7]. McFarling looked at more complex predictors and the possibility of combining global and local predictors for improved accuracy [9]. Later, Jimenez concluded that the hardware cost of many complex predictors do not show a proportionate performance increase [10]. Instead Jimenez presented a simple branch predictor design approach. For our BTB implementation, we make use of our previous work [11] to implement a BTB model with configurable size, associativity and replacement policy.

III. Simulations

The SimpleScalar simulator [3] is an open source pipeline simulator written in C. SimpleScalar is able to simulate

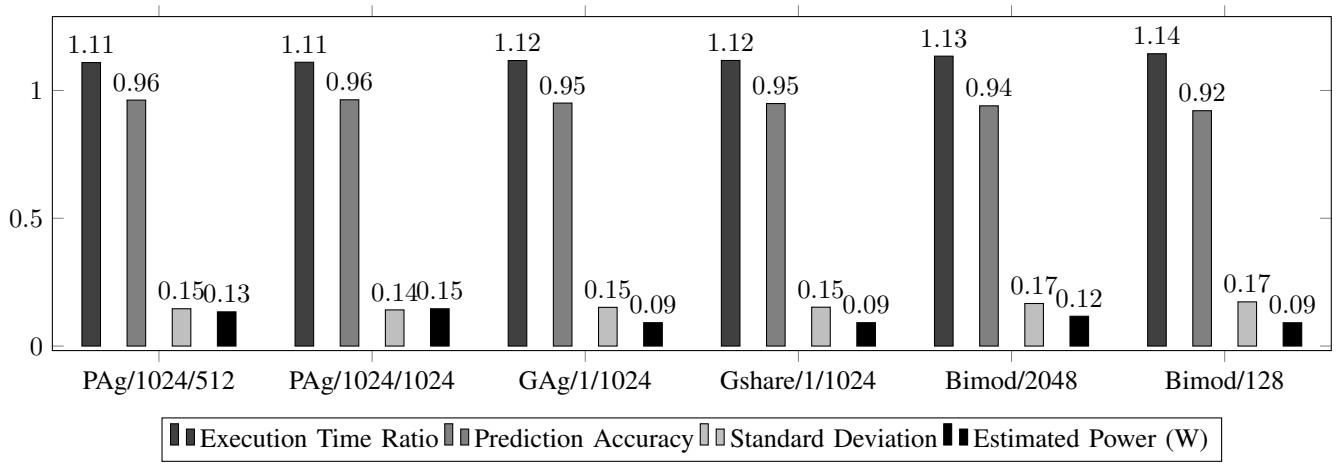

Fig. 1. Performance values of different branch predictors obtained using MiBench benchmark suite.

the PISA and Alpha instruction set architectures and produces simulation statistics and results used to evaluate the pipeline configuration. Here, we use SimpleScalar with the PISA instruction set, in combination with the MiBench [4] and SPEC [5] benchmarks. The MiBench benchmark suite comprises 35 benchmarks written in C, representing typical embedded applications.

By varying the branch direction predictor type and table size as well as the BTB values for table size and associativity, we can study the effects on performance and obtain estimates for area and power consumption. In total, 74 branch predictor configurations were simulated and evaluated. The results were compared to a theoretically perfect predictor with 100 % accuracy. To extract and compile the results, perl scripts were used for automation. In these simulations, we have chosen to limit our exploration to predictor table sizes in the range 64-4,096 branch entries. It is a tedious process to simulate all the branch predictor configurations using SPEC benchmarks, because this suite comprises bigger applications than MiBench. Hence, a few of the best performing branch predictor configurations are selected by evaluating them using the MiBench benchmark suite; then the selected configurations are reevaluated using the SPEC benchmark suite.

Fig. 1 shows the performance values of the selected branch predictor configurations, while Fig. 2 shows the performance of these selected configurations for the SPEC benchmark suite. The figures we present use the following notation style, {Type}/{Table 1 Size}/{Table 2 Size}. Note that the bimodal predictors use only one table level. BTB configurations use the notation {Table Size}/{Associativity}. Note that in Fig. 1 and Fig. 2, the BTB configuration is kept constant, using a direct-mapped table containing 16 entries.

Fig. 1 and Fig. 2 show that the selected branch predictors perform similarly, irrespective of the size of the application. The standard deviation is included as a measure on how well a branch predictor performs over different benchmarks. A high standard deviation could be due to the aliasing effects that arise in some of the benchmarks, as we observe higher standard

Fig. 2. Execution time ratio of different branch predictors obtained using the SPEC benchmark suite.

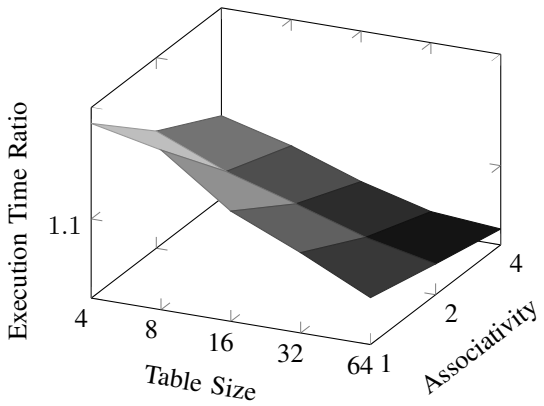

Fig. 3. Performance for different BTB table sizes and associativities.

deviation values for smaller table sizes and BTB configurations with a lower level of associativity. It should be noted that these power estimates serve only as a hint, and are not completely accurate.

Fig. 3 shows the performance of a branch predictor with

respect to different BTB table size and associativity. LRU (Least Recently Used) is used as replacement policy, while the direction predictor configuration is kept constant. It should be noted that while a large BTB table size and high associativity give the best performance in terms of cycles (Fig. 3), there will also be a significant increase in circuit complexity, area and power due to increased table size and associativity. The predictor group that performed best in terms of accuracy and execution time in our simulations were the large PAp-type predictors. However, these come at a significant hardware cost compared to simpler predictors, and the performance increase is not proportional to the increase in hardware cost. Bimodal, GAg and Gshare predictors perform well in relation to the estimated hardware cost and power values. In our simulations, large bimodal predictors (table size 2,048-4,096 entries) saturate at around 94 % direction prediction accuracy.

IV. Implementation

A configurable branch predictor module is implemented in VHDL. The module is designed with modularity and configurability in mind, and comprises three main blocks: the direction predictor, the Branch Target Buffer (BTB) and the control logic. Configuration is done by setting generics at design time. Fig. 4 shows the block diagram of the branch predictor module.

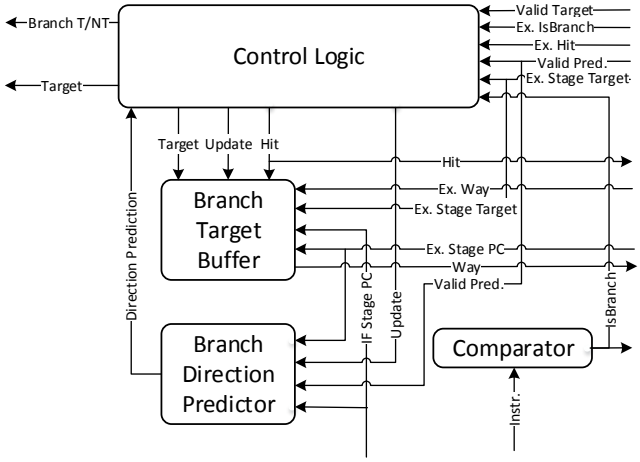

Fig. 4. Branch predictor module block diagram.

The branch predictor module uses information from the pipeline instruction fetch stage to make a BTB lookup in every cycle. Depending on whether the instruction in the instruction fetch stage is a branch or not, the control logic block makes a prediction. If the instruction is a branch, then the direction prediction information for that branch instruction along with the target address read from the BTB are used to determine the next instruction, else the pipeline continues to execute as normal. The branch direction and the target address for the branch instruction are evaluated in the EX stage. This information is fed back to the branch predictor, which updates the direction predictor and the BTB if necessary. In the case of a mispredicted branch direction or address, the branch predictor updates the BTB entry, the direction predictor and passes the correct branch target to the PC. Branch misprediction results in flushing the instructions in the IF and ID stages.

Fig. 5 shows the structure of a two-level predictor. The first table, the Branch History Table (BHT), is indexed using the PC from the IF2 stage. Each BHT entry is a shift register containing the branch history for that index. The second table, the Pattern History Table (PHT), is indexed using the content of the BHT entry.

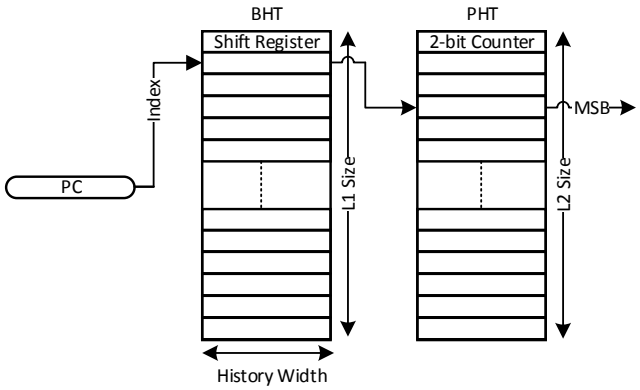

Fig. 5. Generic two-level adaptive predictor.

Each PHT entry is a two bit saturating counter, similar to how a bimodal predictor works. Our direction predictor can be configured for type (two-level or bimodal predictors are supported in the VHDL model) and table sizes at design time. Using a global branch history (as in GAg, GAp and Gshare configurations) is a variation of the two-level predictor where the BHT consists of a single shift register. The direction predictor implementation uses register-based tables.

Fig. 6 gives an overview of the BTB structure. The BTB is SRAM-based and can be configured at design time. The configuration parameters are the number of sets and the set associativity. There are a number of fixed size SRAM blocks available, in sizes 32x32b, 128x32b, 512x32b, 1024x32b and 2048x32b. When synthesized, the BTB implementation will select the smallest SRAM block possible with the given configuration. This means that, e.g., two BTBs of size 16 and size 32 will both use the 32x32b SRAM. There is also a configuration variable for changing the replacement policy when using a set-associative BTB; selecting between LRU and Random replacement.

V. Verification

The branch predictor module has been integrated into a existing 5-stage pipeline and verified using the EEMBC benchmark. Since the branch predictor is designed for two-cycle access, the existing 5-stage pipeline has been modified to accommodate this feature. Fig. 7 shows the branch predictor module integrated into the 5-stage pipeline. The Instruction Fetch stage (IF) is subdivided into two stages, IF1 and IF2. The branch instructions are evaluated in Instruction Decode

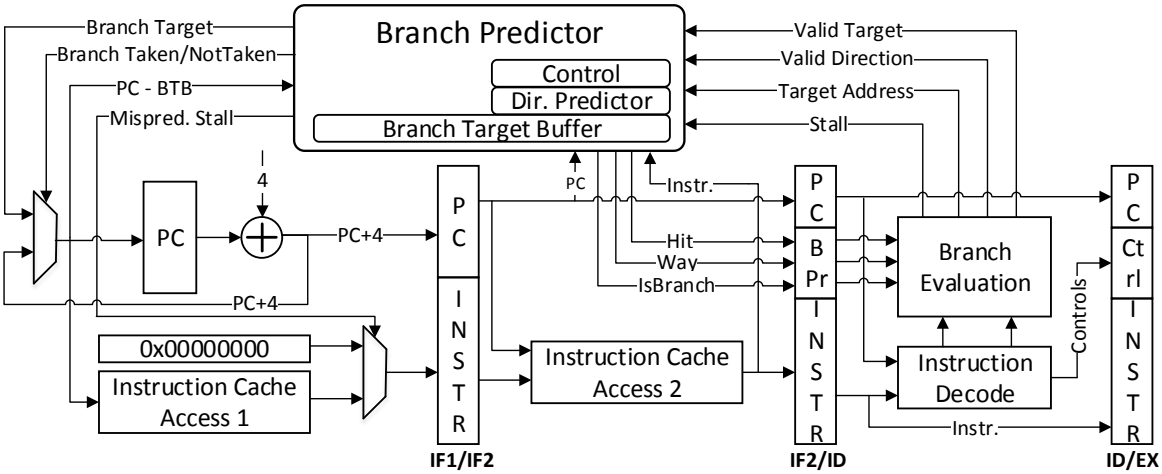

Fig. 7. Branch predictor module integrated into a 5-stage pipeline, which has been modified to accommodate two-cycle access to the predictor.

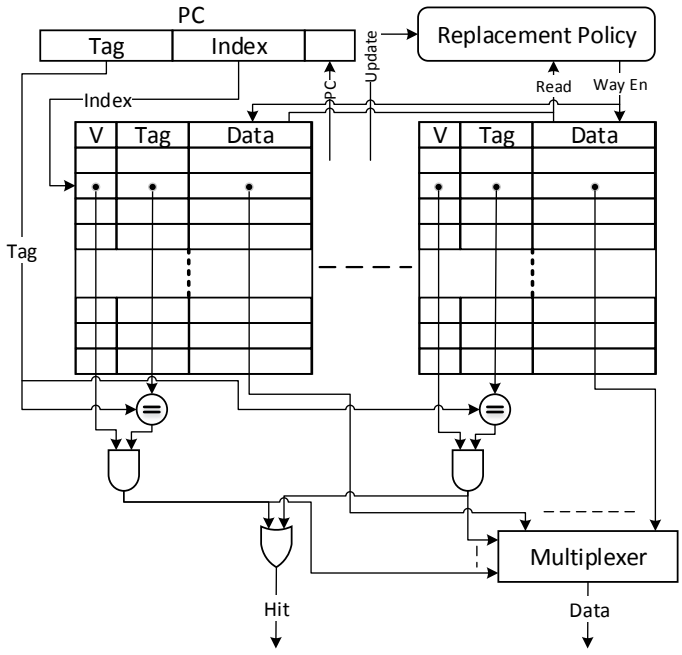

Fig. 6. BTB block diagram.

stage (ID). The result from branch predictor is obtained during the IF2 stage. The BTB is indexed with the instruction address (PC) in IF1 stage, followed by the tag comparison in IF2 stage. The direction predictor and the control logic is accessed in IF2 stage. The branch misprediction penalty is 1 clock cycle, considering that a branch instruction is followed by the branch delay slot instruction as per MIPS ISA.

VI. Results

The VHDL implementation was synthesized using Cadence Encounter RTL Compiler with the settings as given in Table I. Fig. 8 and Fig. 9 show area and power evaluations for four selected branch direction predictor configurations. The BTB configuration is kept constant at 32:1. The figures also show the amount of BTB area (and power) as part of total area (and power).

TABLE I
Synthesis Settings

Clock period	1400 ps
Toggle rate (on primary inputs)	$\alpha = 0.1$
Cell library	65 nm LP, 1.2 V, SVT Library
Optimization Effort	Medium

The results show that the bimodal predictor is much smaller (88 %) than the GAg and Gshare predictors (when looking at the direction predictor only). However, the decrease in execution time is only 1.5 %, and the power is smaller by a factor of 4. A large PAp predictor is included for comparison. By looking at the amount of the area and power made up by BTB, we see that the predictor units utilizing smaller direction predictors are dominated by the BTB, whereas the branch predictor unit with a large PAp predictor is dominated by the direction predictor.

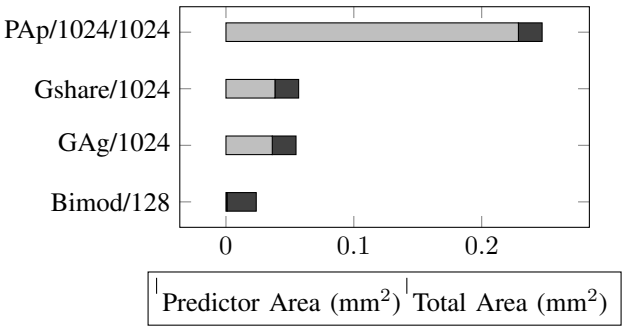

Fig. 8. Post-synthesis results with respect to area for different direction predictors.

The GAg and Gshare predictors give a higher prediction accuracy than the bimodal predictor, both around 3 % better. While these results have to be put into context and are relative to the rest of the pipeline, the bimodal predictors

seem to offer a better performance-area and performance-power trade-off. For the branch predictor unit in isolation, using the synthesis power estimates, a good trade-off between performance and hardware complexity could be combining a GAg:1:1024 direction predictor and a 32:2 BTB. Fig. 10 and Fig. 11 show area and power evaluations for different BTB configurations. The direction predictor is fixed as Bimod 128.

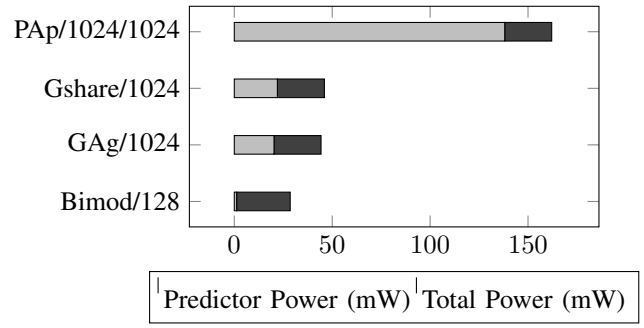

Fig. 9. Post-synthesis results with respect to power for different direction predictors.

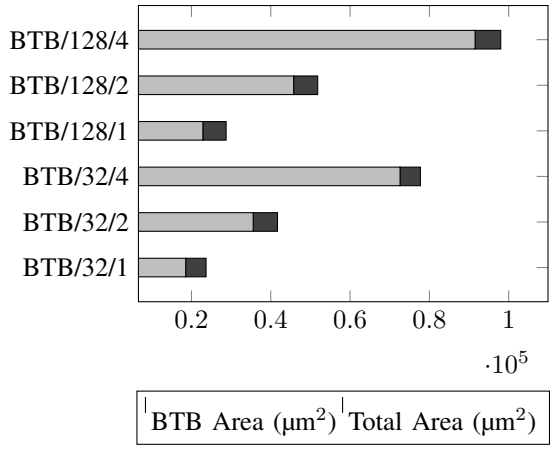

Fig. 10. Post-synthesis results with respect to area for different BTB configurations

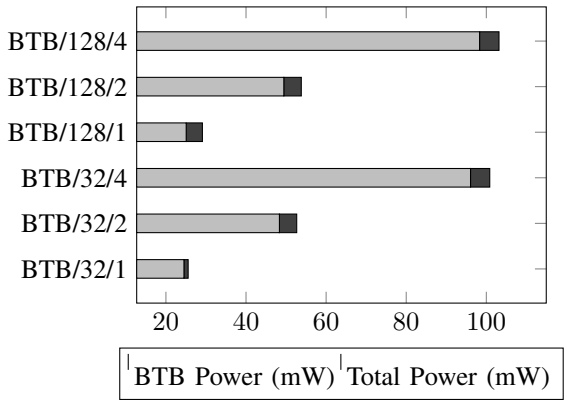

Fig. 11. Post-synthesis results with respect to power for different BTB configurations.

We see that increasing associativity greatly increases the area and power. The BTB area and power numbers include the LRU unit. We note that the difference in area and power between the BTB configurations with 32 and 128 sets is not proportional to the increase in the number of sets. This is because for small SRAM blocks, the peripheral circuitry is dominant. Only after sizes of 4,096 sets and up, the area of the BTB starts scaling in a more linear manner.

VII. Conclusion

In this paper, we have described the simulation, implementation, and synthesis of different branch predictor configurations and obtained preliminary values for area, power, and performance. From the obtained results, we evaluated the branch predictor configurations and made suggestions on which configuration to use in our 7-stage pipeline. We draw the conclusion that large two-level predictors offer the best performance, but exhibit poor (disproportionate) performance-power and performance-area ratios compared to the simpler predictors, which is in agreement with previous work [10]. Using a BTB with large table sizes and high associativity reduces the executed cycles, at the cost of higher area and power. The values for area and power have to be related to the hardware budget of the whole pipeline design in order to properly evaluate what is feasible. Using a large direction predictor will only make sense if it is combined with a large enough BTB. Further improvements can be made to improve the performance of the branch predictor, one such attempt being the optimization of the BTB content.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface, Fourth Edition*, 4th ed. Morgan Kaufmann Publishers Inc., 2008.

[2] I. McLoughlin, *Computer Architecture: An Embedded Approach*. McGraw-Hill Education, 2011.

[3] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. Int. Workshop on Workload Characterization*, Dec. 2001, pp. 3–14.

[5] J. L. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, July 2000.

[6] C. Perleberg and A. Smith, "Branch target buffer design and optimization," *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 396–412, 1993.

[7] T.-Y. Yeh and Y. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 257–266.

[8] ——, "Alternative implementations of two-level adaptive branch prediction," in *Proceedings of the 19th Annual Int'l Symposium on Computer Architecture*, 1992, pp. 124–134.

[9] S. McFarling, "Combining branch predictors," Western Research Laboratory Technical Note TN-36, 1993.

[10] D. Jimenez, "Reconsidering complex branch predictors," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003, pp. 43–52.

[11] V. Saljooghi, A. Bardizbanyan, M. Sjalander, and P. Larsson-Edefors, "Configurable RTL model for level-1 caches," in *NORCHIP, 2012*, 2012, pp. 1–4.