Advanced Branch Predictors for Soft Processors

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

Abstract—This work studies implementations of the Perceptron [1] and TAGE [2] branch predictors for general purpose, in-order pipelined single core soft processors. It proposes FPGA-friendly optimizations whose goal is to achieve high operating frequency. This work discusses the design tradeoffs and proposes a highly accurate and fast branch predictor variant based on TAGE, *O-TAGE-SC*. Using a representative subset of the SPECCPU2006 benchmarks, this work shows that *O-TAGE-SC* delivers 5.2% better instruction throughput versus the previously proposed *gRselect* predictor [3].

I. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly being used in embedded and other systems. Such designs often employ one or more embedded microprocessors, and there is a trend to migrate these microprocessors to the FPGA platform primarily for reducing costs. While these soft processors cannot typically match the performance of hard processors, soft processors are flexible allowing designers to implement the exact number of processors desired and to customize them to efficiently fit the application's requirements.

Current commercial soft processors such as Altera's Nios II [4] and Xilinx's Microblaze [5] use in-order pipelines with five to six pipeline stages. These processors are often used for less computation-intensive applications such as system control tasks. To support more compute-intensive applications, a key performance improving technique is branch prediction. Branch prediction has been extensively studied, 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 as the tradeoffs are different for reconfigurable compared to custom logic. Wu et al. have shown that a branch predictor design for soft processors should balance its prediction accuracy as well as its maximum operating frequency [3]. They proposed *gRselect*, an FPGA-friendly, minimalistic branch predictor implementation for Altera's highest performing soft-processor Nios II-f.

Wu et al. limited the hardware budget of the gRselect predictor to just one M9K Block RAM [6] on Altera Stratix IV devices; Altera's Nios II-f also uses just a single M9K BRAM thus gRselect does not increase the overall soft processor budget disproportionally. Such a small hardware budget prohibits more elaborate and potentially more accurate, state-of-the-art branch prediction schemes such as Perceptron [1] and TAGE [2]. Accordingly, this work relaxes the hardware budget constraint and investigates FPGA-friendly implementations of the Perceptron and TAGE predictors. It studies their accuracy and speed as a function of hardware budget.

Specifically, this work makes the following contributions: (1) It studies the FPGA implementation of the Perceptron and TAGE predictors. It optimizes Perceptron's maximum operating frequency by introducing (i) a complement weight table to simplify the multiplication that is otherwise necessary at prediction time, and (ii) Low Order Bit (LOB) Elimination for faster summation. (2) It compares the branch direction prediction accuracy of the predictors showing that compared to the gRselect, Perceptron is 19.6% worse while TAGE is 2.3x better in terms of Mispredictions Per Kilo Instructions (MPKI). (3) It finds that TAGE is too slow for single-cycle access which negates its advantage in MPKI. Accordingly, this work proposes an overriding predictor O-TAGE-SC that uses a simple base predictor to provide an initial prediction in the first cycle which can be overridden in the second cycle should TAGE disagree with relatively high confidence. O-TAGE-SC achieves 5.2% better instruction throughput over gRselect.

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), i.e., the memory address, 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. This work targets the direction predictor. The *target* predictor guesses the address for "predicted as taken" branches and often includes a *stack* predictor for predicting function returns. 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 or stack predictor. Since, at this point in time, the actual instruction is not available in a typical ASIC pipeline 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. Due to limited capacity, multiple branches may map onto the same prediction entries. This *aliasing* tends to reduce accuracy.

A. Design Goals

This work aims to implement Perceptron and TAGE predictors that (1) operate at a high operating frequency while (2) achieving high accuracy so that they improve execution performance. As Section IV-B will show, a single-cycle TAGE is prohibitively slow. Therefore, this work considers an overriding TAGE predictor [7] that produces a base prediction in one
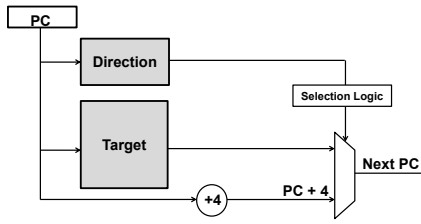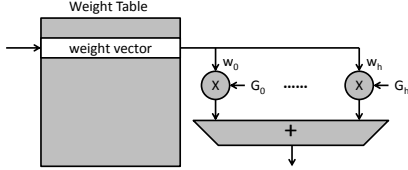
Fig. 1: Canonical Branch Predictor.

Fig. 2: The Perceptron branch predictor.

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

cycle while overriding that decision with a better prediction in the second cycle if necessary. Perceptron and TAGE both require large tables. Accordingly, this work investigates how their accuracy and latency vary with the amount of hardware resources they are allowed to use.

III. BRANCH PREDICTION SCHEMES

This section discusses the structure of the branch predictors considered: (1) the Perceptron and (2) TAGE direction predictors, and (3) the target predictor. Sections III-A and III-B discuss Perceptron and TAGE, while section III-C discusses the target predictor.

A. Perceptron Predictor

The Perceptron predictor uses vectors of weights (i.e., perceptrons) to represent correlations among branch instructions [1]. Fig. 2 shows the structure of a Perceptron predictor. It produces a prediction through the following steps: (1) A *perceptron* is read from the table. (2) The weights are multiplied with factors chosen based on the the corresponding global history bits. The weights are multiplied by 1 for taken and -1 for not-taken. (3) The resulting products are summed up and a prediction is made based on the sign of the result: predict taken if the sum is positive, and not-taken otherwise. Formally, for a Perceptron predictor using h history bits, let G_i, where $i = 1...h$, be 1 for taken and -1 for not-taken, each weight vector has h weights $w_{0...h}$, where the bias constant $w_0 = 1$. The predictor has to calculate $y = w_0 + \sum_{i=1}^{h} G_i w_i$, and predict taken if y is positive and not-taken otherwise.

B. Tagged Geometric History Length Branch Predictor

The TAGE predictor features a bimodal predictor as a base predictor T_0 and a set of M tagged predictor components T_i [2]. These 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 of various lengths. 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 judiciously. Essentially, the base predictor captures the bulk of branches that tend to be biased, while the remaining components capture exceptions by recording specific history events that lead to exceptions that foil the base predictor. Fig. 3 shows a 5-component TAGE predictor. Each
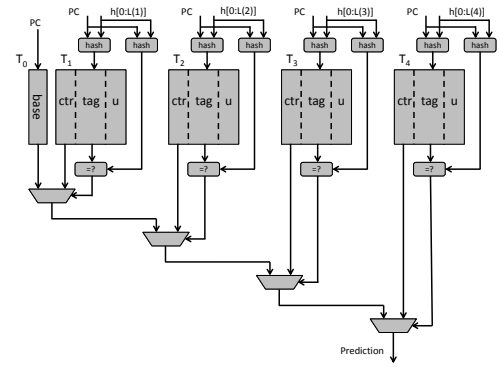
table entry has a 3-bit saturating counter *ctr* for the prediction result, a *tag*, and a 2-bit useful counter *u*. The table indices are produced by hashing the PC and the global history using different lengths per table $L(i)$. All tables are accessed in parallel and each table provides a valid prediction only on a tag match and provided that the corresponding useful counter is saturated. The final prediction comes from the matching tagged predictor component that uses the longest history.

C. Branch Target Predictor

For completeness, this section also reviews the target predictor used in this work. Branch Target Prediction usually requires a Branch Target Buffer (BTB), a cache-like structure that records the addresses of the branches and their target addresses. If a branch is predicted to be taken and there is also a BTB hit, then the next PC is set to be the predicted target. A BTB can be set-associative to reduce aliasing.

Another common structure used for branch target prediction is the Return Address Stack (RAS), a stack-like structure that predicts the target address of function returns. When a call instruction executes, the return address of that call is pushed onto the RAS. When the processor executes the corresponding return instruction, RAS pops the return address and provides a prediction. The prediction is accurate as long as the RAS' size is less than the current call depth. Most modern processors have a shallow RAS because typical programs generally do not have very deep call depths.

Wu et al. have shown that when using one M9K BRAM – a hardware budget on par with that of Nios II-f – eliminating the BTB and using *Full Address Calculation* (FAC) together with a RAS results in better performance [3]. FAC calculates the target address in the fetch stage and thus accurately determines the target addresses for direct branches, whose target can be calculated based on the instruction itself [4]. Wu et al. have shown that direct branches and returns comprise over 99.8% of all branches. Implementing FAC with RAS can cover these branches with 100% accuracy, therefore having a BTB to cover all branches results in negligible improvement in target prediction accuracy. On the other hand, eliminating the BTB and dedicating the entire BRAM for direction prediction improves direction prediction accuracy significantly.

Since, this work investigates how branch prediction accuracy can improve when additional hardware resources are used, adding a BTB for better target prediction coverage could improve target prediction accuracy. Accordingly, we consider
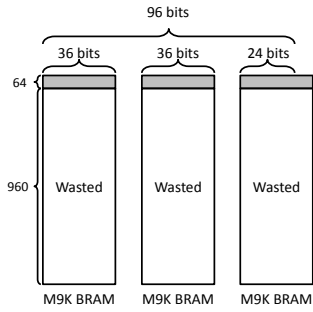
Fig. 4: Inefficient use of M9K BRAMs to implement wide but shallow perceptron tables.

Fig. 5: Perceptron multiplication implementation.

reintroducing a BTB. However, simulations show that accuracy is still better without a BTB. This is because when the target predictor only has FAC and RAS, it never predicts indirect branches that are not returns because it is not capable to do so. As a result, the destructive aliasing in the *direction* predictor is alleviated because fewer branches are being predicted. Based on this observation, this work uses FAC with a 16-entry RAS as the branch target predictor, the same one used in gRselect.

IV. FPGA IMPLEMENTATION OPTIMIZATIONS

This section discusses FPGA-specific implementation optimizations for Perceptron and TAGE. While this section assumes a modern Altera FPGA, the optimizations presented should be broadly applicable.

A. Perceptron Implementation

Section III-A explained that the Perceptron predictor maintains vectors of weights in a table and produces a prediction through three steps. Each of these steps poses difficulties to map to an FPGA substrate. The rest of this section addresses these problems.

1) Perceptron Table Organization: Each weight in a perceptron is typically 8-bit wide, and Perceptron predictors usually use at least a 12-bit global history [1]. The depth of the table, on the other hand, tends to be relatively shallow (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 of a M9K BRAM on Altera Stratix IV is 36-bit wide times 1K entries [6]. If we implement the 1KB Perceptron as proposed by Jiménez et al. [1], which uses 96-bit wide perceptrons with 12-bit global history, it will result in a huge resource inefficiency as shown in Fig 4. Stratix IV chips have another larger but slower and fewer M144K BRAM [6], which can be configured as wide as 72-bit times 2K entries, clearly the inefficiency problem persists and it would impact maximum operating frequency.

Since typically the Perceptron table does not require large storage space, the proposed Perceptron implementation uses MLABs as storage, which are fast fine-grain distributed memory resources. Since 50% of all LABs can be configured as MLAB on Altera Stratix IV devices, using MLABs does not introduce routing difficulty.

2) Multiplication: The multiplication stage calculates the products of weights in a perceptron and their global direction histories. Since the value of the global direction history can
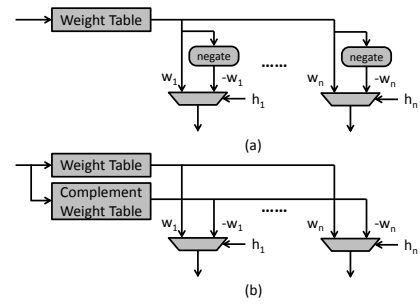
only be either 1 or -1, the "multiplication" degenerates to two cases, i.e., each product can either be the true form or the 2's complement (i.e., negative) form of each weight. A straightforward implementation calculates the negative of each weight and uses a mux to select, using the corresponding global history bit h_i, the appropriate result, as Fig. 5(a) shows. To improve operating frequency, when updating the perceptron in the execution stage where the branch is resolved, both positive and negative forms of the updated weight can be pre-calculated, and the negatives can be stored on a complement perceptron table. This way, the multiplication stage at prediction time requires only a 2-to-1 mux, as Fig. 5(b) shows. This optimization trade offs increased resources (it requires extra storage for the negative weights) for improved speed.

3) Adder Tree: The adder tree sums the products from the multiplication stage. As Section V-B will show, a global history of at least 16 bits has to be used to achieve sufficient accuracy. Implementing a 16-to-1 adder tree for 8-bit integers naïvely degrades maximum frequency severely. The maximum frequency has to be improved for Perceptron to be practical.

This work employs *Low Order Bit (LOB) Elimination* that was proposed by Aasaraai et al. [8]. LOB elimination ignores the Low Order Bits (LOBs) of each weight and only use the High Order Bits (HOBs) during prediction, while still using all the bits for updates. Section V-B shows that eliminating five LOB bits reduces accuracy by less than 1% compared to using all eight bits, but summing fewer bits results in 14.6% higher maximum frequency. Section V-D will show that using three HOBs for prediction achieves the best overall performance.

Cadenas et al. proposed a method to rearrange the weights stored in the table in order to reduce the number of layers of the adder tree [9]. Assuming a Perceptron predictor uses h history bits, instead of storing h weights w_i where $i = 1...h$, a new form of weights $\widetilde{w}_i$: $\widetilde{w}_i = -w_i + w_{i+1}$; $\widetilde{w}_{i+1} = -w_i - w_{i+1}$, for $i = 1, 3, ..., h-1$ is used. The perceptron prediction can now be computed by $y = w_0 + \sum_{i=1}^{h/2}(-G_{2i-1})\widetilde{w}_{2i-h_{2i-1}} \bigoplus h_{2i}$. This work applies this new arrangement because it pushes part of the calculation to the less time critical update logic of the Perceptron predictor so that only $h/2$ additions have to be performed, hence reduces the number of adders required by 50%.

Using fast adders such as carry-lookahead adders does not help to reduce the adder tree latency. This is because that the problem is not summing a few very wide numbers, but many narrow numbers. Most of the latency comes from going through layers of adders rather than propagating the carry bits. To further improve maximum frequency, this work adapts the

implementation of a Wallace Tree [10]. A Wallace tree is a hardware implementation of a digital circuit that efficiently sums the partial products when multiplying two integers, which is similar to the situation that a Perceptron predictor is facing. The Wallace tree implementation proves to be 10.5% faster than a naïve binary reduction tree implementation.

B. TAGE Implementation

Section V-B shows that TAGE is the most accurate amongst all the direction predictors considered in this work when they use the same hardware budget. Unfortunately, TAGE uses multiple tables with tagged entries that require comparator driven logic which does not map well onto FPGAs. Section V-D shows that the resulting frequency slowdown with TAGE is not amortized by the corresponding accuracy gains. Fortunately, TAGE can be used as an overriding predictor maintaining the accuracy gains and relatively high operating frequency.

The critical path of TAGE is as follows: (1) It performs an elaborate PC-based hashing to generate multiple table indices, one per table. (2) It accesses the tables and in parallel compares the tags of the read entries to determine whether they match. (3) Finally each matching entry has to pass through cascaded layers of multiplexers to select the longest matching prediction. Although the latency of these operations is high, the path can be easily pipelined to achieve much higher operating frequency. Based on this observation, this work explores an overriding branch predictor implementation using TAGE.

Overriding branch prediction is a technique to leverage the benefits of both fast but less accurate, and slow but more accurate predictors. This technique has been used commercially, e.g., in the Alpha EV8 microprocessors [11]. In an overriding predictor, a faster but less accurate base predictor makes a base prediction quickly in the first cycle, and then a slower but more accurate predictor overrides that decision, at a latter cycle, if it disagrees with the base prediction.

In this work, the base predictor is the simple bimodal predictor included in TAGE itself, i.e., T_0 in Fig. 3. The bimodal predictor provides a base prediction in the first cycle, and the tagged components of the original TAGE provide a prediction at the second cycle. Sections V-B and V-C show that an overriding TAGE predictor outperforms all the other branch prediction schemes in terms of both accuracy and maximum frequency.

With an overriding predictor, there is no guarantee that the overriding component will indeed be correct. Accordingly, it is essential that any benefits gained when the overriding component is right are higher than the performance lost when it is wrong. For this purpose, this work proposes the use of a confidence mechanism for applying overrides judiciously. Specifically, the confidence mechanism implemented is a small table with 256 entries that is indexed by eight bits from the PC. Each entry is a 10-bit saturating counter. The counter is updated whenever the basic and the overriding component disagree. When they disagree, the counter is incremented when the overriding component is correct and reset otherwise. Overrides are activated only after the counter saturates. Seznec also suggested using a similar confidence mechanism, a *statistical corrector*, in his ISL-TAGE improvement over the original

TAGE [12]. There the statistical corrector is used in a single-cycle non-overriding TAGE predictor to avoid using the tagged components whenever the bimodal component proves better. Seznec's observation was that the tagged components fail at predicting branches that are statistically biased towards a direction but not correlated to the history path. On some of these branches, TAGE often performs worse than a simple PC-indexed table, e.g., a bimodal predictor.

The confidence mechanism/statistical corrector used in this work is similar to those proposed by Jacobsen et al. [13], except that our specific statistical corrector is only updated when the basic and the overriding component disagree. The specific confidence mechanism performed better than Seznec's mechanism. This is no surprise as here it is used to guide overrides in an overriding TAGE predictor. Specifically, in Nios II-f where the branch resolution latency is only two cycles, the overriding TAGE saves one cycle for each correct override, but loses two for each incorrect override. Hence, the overriding TAGE must be very confident to make an overriding decision, which necessitates the specific statistical corrector.

As a result, this work proposes four TAGE-based designs that use one or two cycles, with or without a confidence mechanism: (1) the single-cycle TAGE, which requires TAGE to provide a prediction in one cycle (i.e., in the fetch stage), (2) the Overriding TAGE (O-TAGE), which uses just the bimodal predictor (i.e., T_0) to provide a base prediction in the first cycle, and *always* overrides the base prediction if TAGE disagrees at the end of the second cycle, (3) the single-cycle TAGE with a Statistical Corrector (single-cycle TAGE-SC), which forces the predictor to use the base prediction unless TAGE consistently disagrees over several encounters of the same event, and (4) the Overriding TAGE with a Statistical Corrector (O-TAGE-SC), which is similar to the single-cycle TAGE-SC except that TAGE overrides the base prediction in the second cycle. The accuracy and critical path of the Perceptron predictor did not justify investigating an overriding configuration based on Perceptron.

V. EVALUATION

This section evaluates the branch predictors. Section V-A details the experimental methodology. Section V-B compares the accuracy of the various direction predictors: bimodal, gshare, gRselect, Perceptron and TAGE. It shows that TAGE is the most accurate. Section V-C reports the maximum operating frequency. Finally, Section V-D reports the overall performance and the FPGA resource usage, showing that the overriding TAGE predictor is the best performing predictor.

As Section III-C discussed, all configurations use the same target prediction scheme, which includes a FAC and RAS, the same target predictor used in the gRselect predictor [3].

A. Methodology

To compare the predictors, this work measures: (1) Accuracy as Mispredictions Per Kilo Instructions (MPKI), which has been shown to correlate better with performance compared to prediction accuracy alone. (2) The Instruction Per Cycle (IPC) instruction execution rate, a frequency agnostic metric that better reflects the accuracy of each predictor factoring away their latency, (3) Instructions Per Second (IPS), a true
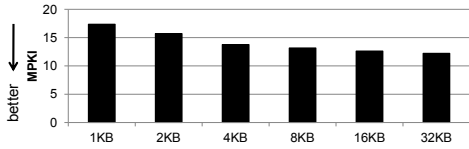
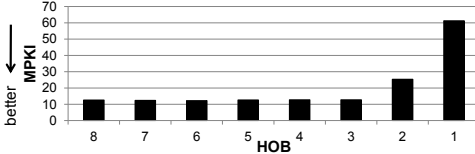Fig. 6: Perceptron: MPKI of the most accurate Perceptron configuration with various hardware budgets.

Fig. 7: Perceptron: MPKI when using different number of HOBs for the most accurate Perceptron configuration.

measure of performance which takes the operating frequency into account, (4) Operating frequency, and (5) Resource usage. Simulation measures MPKI and IPC using a custom, cycle-accurate, full-system Nios II simulator. The simulator boots ucLinux [14], and runs a representative subset of SPEC CPU2006 integer benchmarks with reference inputs [15].

The baseline predictors considered are: (1) bimodal, (2) gshare and (3) gRselect. These predictors are implemented to match as closely as possible the implementations of Wu et al. [3]. All designs were implemented in Verilog and synthesized using Quartus II 13.0 on a Stratix IV EP4SE230F29C2 chip in order to measure their maximum clock frequency and area cost. The maximum frequency is reported as the average maximum clock frequency of five placement and routing passes with different random seeds. Area usage is reported in terms of ALUTs used.

B. Branch Prediction Accuracy

This section first presents data that justify the final design of Perceptron and TAGE configurations, then a comparison with bimodal, gshare and gRselect is presented.

1) Perceptron: This work considers Perceptron predictors with a hardware budget ranging from 1KB to 32KB. For each hardware budget, the number of global history bits is varied and the best performing one is chosen. Fig. 6 shows the most accurate Perceptron configuration for each hardware budget. All of these configurations uses 16 history bits. As Section V-D will show, although the 32KB Perceptron is 3.2% more accurate than the 16KB Perceptron, its IPC saturates at the 16KB budget, therefore for the rest of this work the 16KB Perceptron predictor is used.

To determine how many HOBs the predictor should use, we took the 16KB Perceptron and experimented with all possible numbers of HOBs used. Fig. 7 shows the MPKI of this Perceptron when different number of HOBs are used. The data shows that using three HOBs degrades accuracy by less than 1% compared to using all eight bits. However, the MPKI doubles when using only two HOBs. Therefore the the implemented Perceptron designs use three HOBs to improve operating frequency without affecting accuracy.

The best performing Perceptron uses 16 global history bits. It has a 16KB perceptron table, which stores 1K perceptrons. Each perceptron contains 16 8-bit weights with the
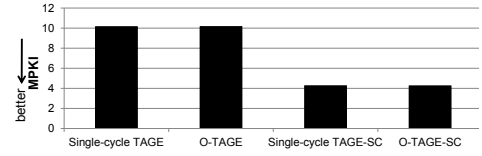

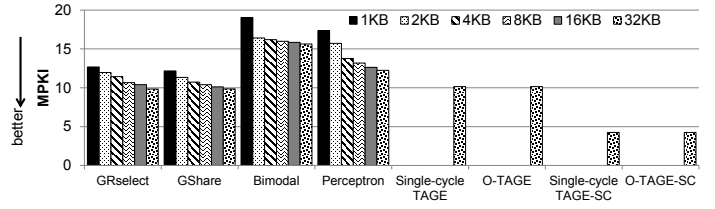Fig. 8: TAGE: MPKI of the four TAGE variations.
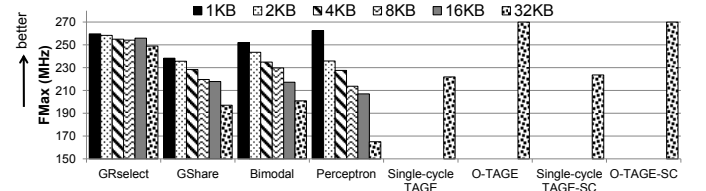

Fig. 9: MPKI of the direction predictors.

Fig. 10: Maximum operating frequency of the considered branch prediction schemes with various hardware budget.

arrangement discussed in Section IV-A3. It also has a 6KB complement table that stores three HOBs per weight in its 2's complement form to improve frequency. Thus we name this best performing Perceptron the *16KB+6KB Perceptron*. We will follow this convention in the rest of this paper, but the hardware budgets for various Perceptron configurations in the remaining figures only refer to their perceptron table sizes.

2) TAGE: All TAGE configurations studied in this work use Seznec's original table configurations [2] and are within 32KB hardware budget. Adjusting TAGE's size is a non-trivial task, moreover, the results of this work show that the 32KB O-TAGE-SC outperforms the other predictors. Accordingly, we do not vary TAGE's size in this work. Fig. 8 shows the MPKI of the four designs that incorporate TAGE. It shows that the single-cycle and overriding predictors have virtually identical MPKI, the statistical correcter improves MPKI by ~2.4x.

3) Accuracy Comparison: For fair comparisons, we scale bimodal, gshare and gRselect from 1KB to 32KB, which is the same hardware budget as TAGE and the largest Perceptron considered in this work. Fig. 9 shows the MPKI of various direction predictors. The TAGE variations use 32KB. All the branch predictors get more accurate as the hardware budget increases. The single-cycle TAGE-SC is the most accurate, followed by O-TAGE-SC with less than 0.06% difference. The single-cycle TAGE-SC is ~2.3x more accurate than the 32KB gRselect and the 32KB gshare.

C. Frequency

Fig. 10 shows the maximum operating frequency for each branch prediction scheme and for various hardware budgets.

The fastest predictors are O-TAGE-SC and O-TAGE, both of them are capped at 270 MHz, the maximum frequency for Nios II-f on Stratix IV C2 speed grade devices [16].
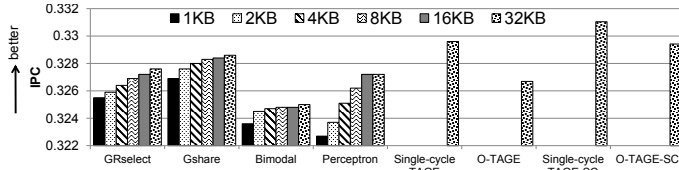
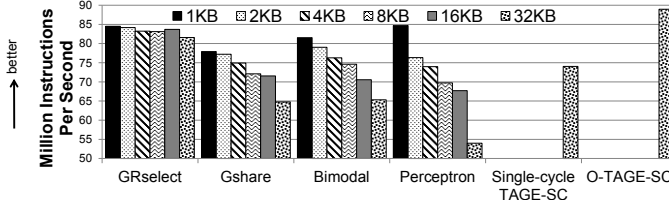Fig. 11: IPC of the considered branch predictors.

Fig. 12: Processor IPS comparison with various predictors.

The 1KB+384B Perceptron and the 1KB gRselect follow the O-TAGE variations. The maximum frequency of gshare, bimodal and Perceptron drop rapidly with increasing size, while gRselect's frequency does not suffer too much. Despite that the logic is larger and more difficult to place and route, the table indexing of gRselect comes from the GHR register. GRselect reads a wide entry and then using bits from the PC to select the appropriate ones. The indexing of gshare, bimodal and Perceptron uses the predicted PC. The PC is both the input and the output of the branch predictor. This loop forms the critical path of gshare, bimodal and Perceptron, which quickly gets slower as the sizes of the predictors increase. The single-cycle TAGE-SC operates at 223.7MHz, which is 14.8% slower than the 1KB gRselect and 17.2% slower than O-TAGE-SC.

D. Performance and Resource Cost

Fig. 11 shows the IPC of the predictors. Although the MPKI of the 32KB+12KB Perceptron is higher than the 16KB+6KB Perceptron, they deliver identical IPC. The single-cycle TAGE-SC has the highest IPC, however, as this section shows, its high IPC cannot amortize the slowdown in operating frequency. O-TAGE is much faster, but its IPC drops significantly. Finally, the IPC of O-TAGE-SC is within 0.5% of the single-cycle TAGE-SC.

IPC is a measurement that does not take operating frequency into consideration. The actual performance of a processor is measured by Instructions Per Second (IPS), which is the product of IPC and the maximum operating frequency. Fig. 12 reports the overall performance in terms of IPS.

The IPS of gRselect, gshare, bimodal and Perceptron drops as they scale, therefore we chose the smallest configurations of these schemes to maximize IPS. The best performing predictor is O-TAGE-SC, which delivers 5.2% higher IPS than the previously proposed 1KB gRselect. Although the single-cycle TAGE-SC is the most accurate, its IPS is lower than the best performing predictor in all other prediction schemes because its latency is too high. The 1KB+384B Perceptron ends up being 0.2% better than the 1KB gRselect, because of the optimization efforts into improving its frequency.

Finally, Fig. 13 shows the resource usage in term of ALUTs used. O-TAGE-SC uses 2.93x ALUTs and the 1KB+384B
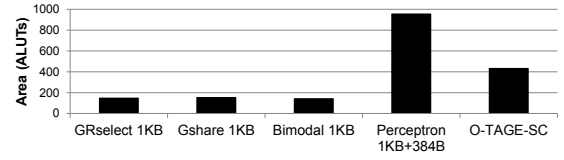

Fig. 13: ALUTs usage comparison with various predictors.

Percpetron uses 6.45x ALUTs as the 1KB gRselect.

VI. CONCLUSION

This work studied the implementation of Perceptron and TAGE branch predictors for general purpose soft processors. It explored the designs tradeoffs of Perceptron and TAGE, and proposed O-TAGE-SC, an overriding predictor that delivers 5.2% better instruction throughput over the best performing previously proposed predictor, gRselect. Several FPGA implementation optimization techniques were proposed to achieve high operating frequency. Although O-TAGE-SC is ~3x more accurate than the 1KB gRselect, the IPS improvement is smaller due to the processor's simple in-order pipeline. Future work may consider investigating the benefits of implementing O-TAGE-SC for more elaborated soft processors, e.g., an Out-of-Order soft processor.

REFERENCES

[1] D. A. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *Intl' Symposium on High-Performance Computer Architecture*.

[2] 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.

[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*, 2013.

[4] Altera Corp., "Nios II Processor Reference Handbook v9.0," 2009.

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

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

[7] D. A. Jiménez, S. W. Keckler, and C. Lin, "The impact of delay on the design of branch predictors," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000.

[8] K. Aasaraai and A. Baniasadi, "A power-aware alternative for the perceptron branch predictor," in *Advances in Computer Systems Architecture*, L. Choi, Y. Paek, and S. Cho, Eds., 2007.

[9] O. Cadenas, G. Megson, and D. Jones, "A new organization for a perceptron-based branch predictor and its fpga implementation," in *Proc. IEEE Computer Society Annual Symposium on VLSI.*, May 2005.

[10] C. S. Wallace, "A suggestion for a fast multiplier," *Electronic Computers, IEEE Transactions on*, vol. EC-13, no. 1, pp. 14–17, Feb 1964.

[11] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design tradeoffs for the alpha ev8 conditional branch predictor," in *Proc. 29th Annual International Symposium on Computer Architecture.*, 2002.

[12] A. Seznec, "A 64 kbytes isl-tage branch predictor," in *JWAC-2: Championship Branch Prediction*, 2011.

[13] E. Jacobsen, E. Rotenberg, and J. E. Smith, "Assigning confidence to conditional branch predictions," in *In Proceedings of the 29th ACM/IEEE International Symposium on Microarchitecture*, 1996.

[14] "Arcturus Networks Inc., uClinux," http://www.uclinux.org/.

[15] Standard Performance Evaluation Corporation, "SPEC CPU 2006," http://www.spec.org/cpu2006/.

[16] *Nios II Performance Benchmarks*, Altera Corp., Nov. 2013.