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 1% worse while TAGE is 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.

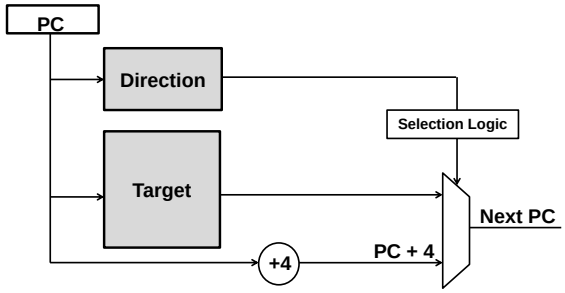

Fig. 1: Canonical Branch Predictor.

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 execution performance. As section III-B will show, a

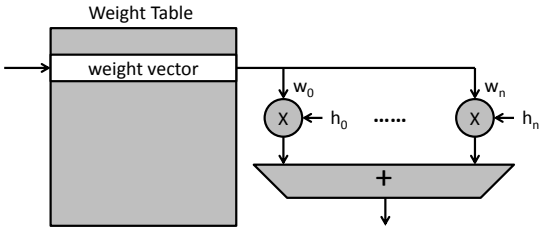

Fig. 2: The preceptron branch predictor.

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. Branch Prediction Schemes

This section discusses the structure of the branch predictors considered, including perceptron and TAGE direction predictor and the target predictor. Sections III-A and III-B discusses perceptron and tage, while section III-C discusses the target predicor.

A. 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.

When making a prediction, a weight vector is loaded from the table. Then, each weight is multiplied by 1 if the corresponding global branch history is taken, and by -1 otherwise. Finally, the products are summed up, the perceptron predicts taken if the sum is positive, and not taken otherwise.

B. 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., $L(i) = (int)(\alpha^{i-1} \times L(1) + 0.5)$. 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.

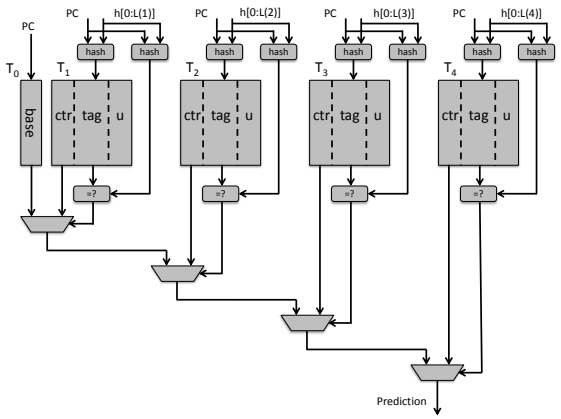

Fig. 3: A 5-component TAGE branch predictor.

C. Branch Target Predictor

Branch Target Prediction usually requires a Branch Target Buffer (BTB), a cache-like structure that records the addresses of the branches and the target addresses associated with them. 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 also have set-associativity to reduce the impact from aliasing.

Another common structure for branch target prediction is a Return Address Stack (RAS). It is a stack-like structure that accurately predicts the target address of function returns. When a call instruction executes, the return address of that call is pushed on RAS. When the processor executes the corresponding return instruction, RAS pop the return address and always provides the accurate prediction. Most modern processors have a shallow RAS because typical programs generally do not have very deep call depths. RAS will fail to provide correct target prediction if it is overflowed.

Wu et al. has shown that within the same hardware budget as Nios II-f (i.e., 1 M9K BRAM), eliminating the BTB and use *Full Address Calculation* (FAC) together with RAS results in better performance [3]. FAC is a technique that calculates the target address in fetch stage to accurately predict target addresses for direct branches, whose target can be calculated based on the instruction itself [1]. Wu et al. has shown that direct branches and returns consist of 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 dedicate the entire BRAM for direction prediction improves direction prediction accuracy significantly.

Since we remove the hardware budget constraint in this work, adding a BTB for better target prediction coverage can improve target prediction accuracy. However, our simulations show that the accuracy of the branch predictor 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 less branches are being predicted. Based on this observation, this work uses FAC with RAS as the branch target predictor, which is the same as 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 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. Mathematically, for a perceptron predictor using h history bits, 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.

Each of these steps poses difficulties to map to FPGA platform. 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 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 chips is 36-bit wide times 1k entries [4]. If we implement the 1KB perceptron as proposed by Jiménez et al. [5], 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 on Stratix IV chips [4], 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 a 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 compliment (i.e., negative) form of each weight. A straight forward implementation is to calculate the negative of each weight and use a mux to select between them based on the corresponding global history, as shown in Fig. 5(a). To improve maximum frequency, when updating perceptron in execution stage where the branch is resolved, both positive and negative forms of the updated weight can be calculated, and the negatives can be stored a complement perceptron table. In this way, the multiplication stage requires only a 2-to-1 mux, as shown in Fig. 5(b). As a tradeoff, it requires extra storage for the negative weights.

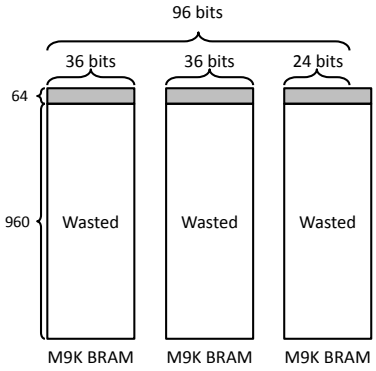

Fig. 4: Inefficiency using M9K BRAMs to implement wide but shallow perceptron tables.

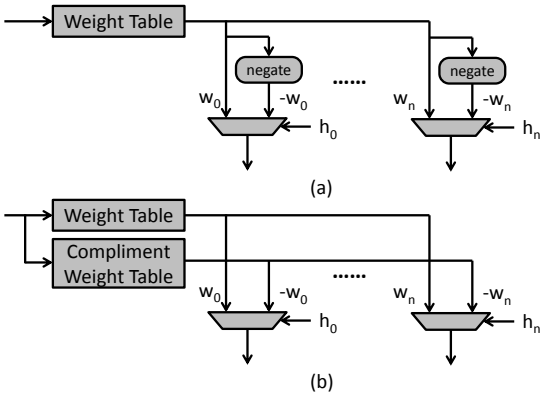

Fig. 5: Perceptron multiplication implementation.

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

This work employs *Low Order Bit (LOB) Elimination* proposed by Aasaraai et al. [7]. The idea is to ignore the Low Order Bits (LOBs) of each weight and only use the High Order Bits (HOBs) during prediction, but use all the bits during update. Section V-B shows that eliminating 5 LOB bits is only (TODO: getting this data soon)x?% less accurate than using all 8 bits, but summing fewer bits results in significantly higher maximum frequency. Our experiments have shown that using 3 HOB for prediction achieves the best balanced performance.

Cadenas et al. [8] proposed a method to rearrange the weights stored in the table in order to reduce the number of layers of the adder tree. Assuming a perceptron predictor uses h history bits, in stead 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 $i = 1, 3, ..., h-1$. The perceptron prediction can now be computed by $y = w_0 + \sum_{i=1}^{h/2}(-G_{2j-1})\widetilde{w}_{2i+G_{2i}G_{2i}}$. The new arrangement 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 adder required by 50%.

The adder tree implementation of this work uses LOB elimination and the new arrangement of weights. To further improve maximum frequency, the adder tree is also hand optimized at the granularity of *Full Adder* (FA) and *Half Adder* (HA), and it is proven to be 10% faster than the logic synthesized by the CAD tool with straight forward Verilog code.

B. TAGE Implementation

Section V-B shows that TAGE predictor is the most accurate amongst all the direction predictors considered in this work with the same hardware budget. However, TAGE uses multiple tables with tagged entries that require comparator driven logic, which does not map well onto FPGAs. Section **??** shows that the maximum frequency slowdown of TAGE cannot be justified by the accuracy gain.

The critical path of TAGE is as follows: it performs a complicated PC based hashing to generate the indices to the tables, compares the tags of the loaded entries to determine whether they are hits or not, finally each decision has to fall through cascaded layers of multiplexers. Although the latency to performing 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 between fast predictors and accurate predictors. This technique has been used on the Alpha EV8 [9] microprocessors. In an overriding predictor, a faster but less accurate base predictor makes a base prediction quickly, and then a slower but more accurate predictor overrides that decision if it disagrees with the base prediction.

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

V. Evaluation

This section presents the evaluation of the branch predictors considered in this work. Section V-A details the experimental methodology. Section V-B compares the accuracy of various direction predictors including bimiodal, gshare, grselect, perceptron and TAGE. It shows that the overriding TAGE is the most accurate. Section V-C reports maximum operating frequency as well as FPGA resource usage. Finally, Section **??** reports the overall performance, showing that the overriding TAGE predictor is the best performing predictor.

As discussed in section III-C, all the evaluation results presented in this section assumes the same target prediction scheme, which includes a FAC and RAS, the same target predictor used in the gRselect predictor [3].

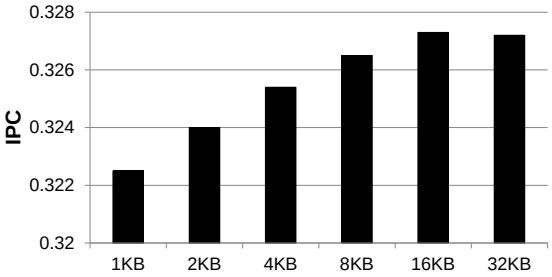

Fig. 6: IPC of the best performing perceptron configuration within various hardware budget.

A. Methodology

To compare the predictors this work measures: (1) Accuracy as Instruction Per Cycle (IPC), a frequency agnostic metric that isolates the effects of implementation, (2) Instructions Per Second (IPS), a true measure of performance, (4) Operating frequency, and (5) resource usage. Simulation measures IPC using a custom, cycle-accurate, full-system Nios II simulator. The simulator boots ucLinux [10], and runs a subset of SPEC CPU2006 integer benchmarks with reference inputs [11].

The predictors used as comparison includes bimodal, gshare and grselect. These predictors are implemented faithfully with the same techniques presented by 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 versus bimodal, gshare and grselect is presented.

1) Perceptron: This work experiments perceptron predictor with hardware budget ranging from 1KB to 32KB. For each of the hardware budget, this work also experiments different configuration with various number of global history bits used. Fig. 6 shows the best performing perceptron configuration for each hardware budget. The most accurate perceptron configuration with 16KB budget.

To determine how many HOBs the predictor should use, we take the most accurate perceptron configuration and experimented with all possible numbers of HOBs used. Fig. 7 shows the IPC of the same perceptron configuration but different number of HOBs used. The data shows that using 3 HOB achieves virtually identical IPC (less than 0.06% difference) versus using all 8 bits, but the accuracy drops significantly when only using 2 HOB. Therefore the final perceptron design uses 3 HOB to improve operating frequency without affecting accuracy.

The most accurate perceptron configuration uses 16 global history bits with the arrangement discussed in section IV-A3, so in the end it only requires space to store eight 8-bit

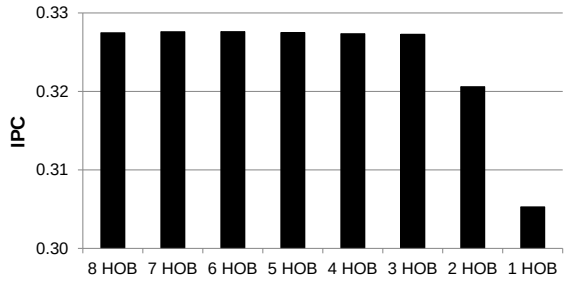

Fig. 7: IPC impact on using different number of HOBs of the most accurate perceptron configuration.

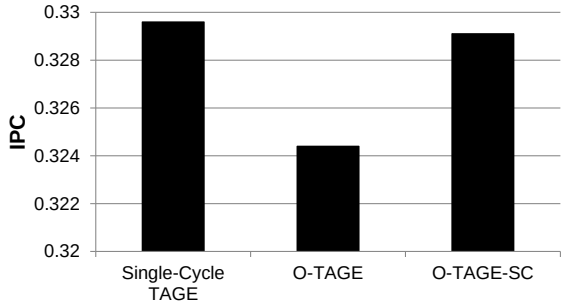

Fig. 8: IPC of the three designs incorporating TAGE.

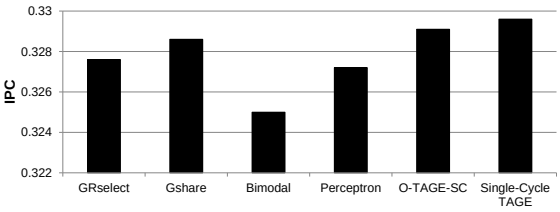

Fig. 9: IPC of the considered branch predictors.

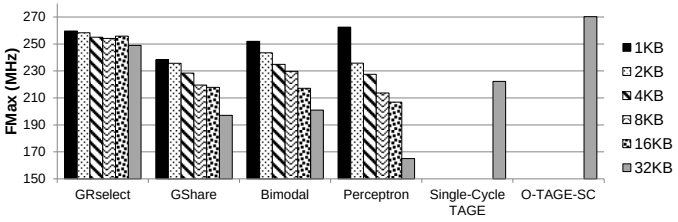

Fig. 10: Maximum operating frequency of the considered branch prediction schemes with various hardware budget.

weights $\widetilde{w}_i$ for $i = 1...8$ plus 3 HOB per weight in its 2's complement form. The perceptron table has 1024 entries. Therefore, the total storage used is: $((8+3)\ bits/weight \times 8\ weights/entry) \times 1024\ entries = 90,112bits = 11KB$.

2) TAGE: This work uses the TAGE predictor introduced by Seznec et al. **??** directly without tuning. This work proposes three designs incorporating TAGE: (1) the singel-cycle TAGE, which requires TAGE to provide a prediction in one cycle (i.e., in fetch stage), (2) the Overriding TAGE (O-TAGE), which uses a simple bimodal predictor to provide a base prediction in the first cycle, and *always* overrides the base prediction if TAGE disagrees, and (3) the Overriding TAGE with a Statistical Corrector (O-TAGE-SC).

Seznec points out that TAGE fails at predicting branches that are statistically biased towards a direction but not correlated to the history path [12]. On some of these branches, TAGE often performs worse than a simple PC-indexed table, e.g., a bimodal predictor. This may result in many miss-overrides on these type of branches. More importantly, in Nios II-f where the branch resolve latency is only 2 cycles, the overriding TAGE saves one cycle for each correct override, but loses two cycles for each incorrect override. It means that the overriding TAGE must make two right decisions for every one wrong decision to break even. Therefore, the overriding TAGE must be very confident to make a overriding decision, which necessitates the statistical corrector.

The statistical corrector we implemented has a small table with 256 entries that is indexed by 8 bits from PC. Each entry is a 10-bit saturating counter. If TAGE correctly overrides a decision, the corresponding counter is incremented by one, otherwise the counter is **reset** to zero.

Fig. 8 shows the IPC of the three designs that incorporate TAGE. It shows that the single-cycle TAGE delivers the highest accuracy, however, as section **??** shows, its high accuracy does not justify the slowdown in operating frequency. O-TAGE is much faster, but its accuracy drops significantly. Finally, the accuracy of O-TAGE-SC is within 0.2% of the single-cycle TAGE.

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 IPC of the best performing configuration of each prediction scheme. All the branch predictors except for perceptron shown in this figure uses 32KB storage. The 32KB and 16KB perceptron deliver identical IPC, therefore the 16KB version is selected as the final implementation. The Single-Cycle TAGE is the most accurate, followed by O-TAGE-SC and Gshare. Perceptron's accuracy comes close to gRselect but still slightly less accurate.

C. Frequency

Fig. 10 shows the maximum operating frequency for each branch prediction scheme with various hardware budget. The Single-Cycle TAGE and O-TAGE-SC only have 32KB version.

The fastest predictor is O-TAGE-SC operating at 270 MHz, followed by the 1KB perceptron and the 1KB gRselect. The maximum frequency of gshare, bimodal and perceptron drops rapidly with increasing size, while gRselect does not suffer too much with this problem. Despite that the logic is larger and more difficult to place and route, more importantly, the indexing of gRselect comes from the GHR register, while the indexing of gshare, bimodal and perceptron comes from the predicted PC, which is both the input and the output of the branch predictor. This loop forms the critical path of gshare,

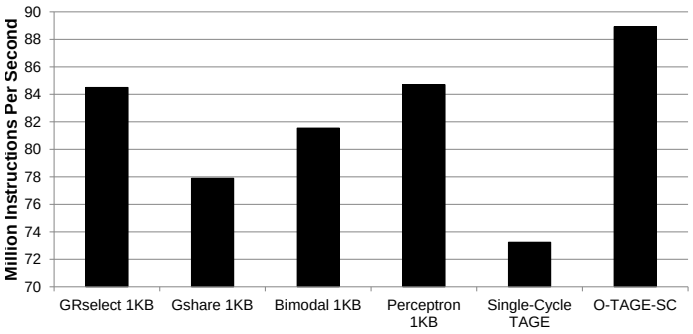

Fig. 11: Processor IPS comparison with various predictors.

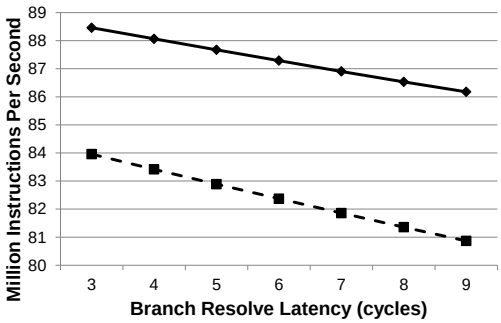

Fig. 12: xxxxxxxxxxxxx?????????????.

bimodal and perceptron, which quickly gets slower as the size of the predictors increase. The Single-Cycle TAGE operates at 222MHz, which is ~15% slower than the 1KB gRselect and ~18% slower than O-TAGE-SC.

D. Performance

IPC is a measurement that does not taken operating frequency into consideration. The actual performance of a processor is measured by Instruction Per Second (IPS), which is the product of IPC and the maximum operating frequency. Fig. 11 reports the overall performance in terms of IPS. This experiment assumes a 270 MHz maximum clock frequency for the processor, the maximum clock frequency of Nios II-f on Stratix IV C2 speed grade devices [13].

The best performing predictor is O-TAGE-SC, which delivers 5.2% higher IPS than the 1KB gRselect. Although the Single-Cycle TAGE is the most accurate, its IPS is the lowest because its latency is too high. The 1KB perceptron ends up 0.2% better than the 1KB gRselect, because of the optimization efforts into improving its frequency.

E. Increasing Branch Resolve Latency

As discussed in section V-B2, the branch resolve latency is so short that O-TAGE-SC has to make at least two correct overrides for every incorrect overrides just to break even. Even with this harsh condition, O-TAGE-SC is still 1.1% more accurate than the 1KB gRselect. Assuming the branch resolve latency is n cycles where $n > 1$, the required ratio of correct and incorrect overrides is $n/(n-1)$, which approaches 1 as n gets larger. Potentially the performance gap between O-TAGE-SC and gRselect can also get larger. Fig. 12 shows the IPS of the 1KB gRselect and O-TAGE-SC in a Nios II processor that requires more than two cycles to resolve a branch.

There is a trend to scale soft processors to support more computation-intensive applications, inevitably the pipeline of soft processors will get deeper.

VI. 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] K. Aasaraai and A. Baniasadi, "A power-aware alternative for the perceptron branch predictor," in *Advances in Computer Systems Architecture*, ser. Lecture Notes in Computer Science, L. Choi, Y. Paek, and S. Cho, Eds. Springer Berlin Heidelberg, 2007, vol. 4697, pp. 198–208.

[8] O. Cadenas, G. Megson, and D. Jones, "A new organization for a perceptron-based branch predictor and its fpga implementation," in *VLSI, 2005. Proceedings. IEEE Computer Society Annual Symposium on*, May 2005, pp. 305–306.

[9] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design tradeoffs for the alpha ev8 conditional branch predictor," in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*. IEEE, 2002, pp. 295–306.

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

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

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

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