LOW-COST, HIGH-PERFORMANCE BRANCH PREDICTORS FOR SOFT PROCESSORS

Di Wu, Kaveh Aasaraai and Andreas Moshovos

Electrical and Computer Engineering Department
University of Toronto
{wudi7, aasaraai, moshovos}@eecg.toronto.edu

ABSTRACT

This work studies branch predictor implementations for general purpose, pipelined, single core soft processors. It shows that the existing designs do not map well onto reconfigurable hardware since they were optimized for custom logic implementation. This work then proposes an accurate and fast branch predictor that uses few resources on FPGAs. The proposed predictor uses: (1) an FPGA-friendly pattern based direction predictor, (2) a return address stack, (3) in-fetch target address calculation instead of a branch target buffer, and (4) instruction pre-decoding. Experimental measurements using a subset of the SPECCPU2006 workloads show that the presented FPGA-friendly branch predictor delivers high performance while operating at approximately 259 MHz using only 147 ALUTs and one BRAM on an Altera Stratix IV FPGA.

1. INTRODUCTION

FPGA-based designs often incorporate one or more general purpose soft processors. As the range of FPGA applications broadens and evolves, it is likely that the performance demand from soft processors will increase. A key performance enhancing technique that even simple general purpose processors use is branch prediction. Without branch prediction, a branch has to execute completely before the processor can fetch the instructions that follow. Branch prediction eliminates these stalls by guessing the target address of branches. Current state-of-the-art branch prediction techniques, e.g., TAGE [1], rely on dynamically collected information about past branch behavior. Such techniques have been proven to be very effective even in deeply pipelined, highly speculative, high-performance custom processor designs.

Branch prediction has been extensively studied in the context of application specific custom logic (ASIC) implementations. Since the tradeoffs are different for reconfigurable logic, naïvely porting ASIC-based branch predictors to FPGAs may prove slow and/or resource-inefficient. Accordingly, this work studies the FPGA implementation of several commonly used branch predictor designs and does so in the context of simple pipelined processors, the most commonly used general purpose soft processor architecture due to its excellent balance of performance and resource cost. For this purpose, it assumes a pipelined processor implementation representative of Altera's Nios II-f and investigates the performance and resource cost of various branch predictors. The analysis confirms that existing designs are not efficient nor high-performing on reconfigurable logic. Accordingly, this work proposes FPGA-specific modifications that improve accuracy, resource cost, or both.

In more detail, this work makes the following contributions: (1) It studies the FPGA-implementation of Branch Target Buffers (BTB), including designs that fuse the BTB and the direction predictor and shows that, contrary to ASIC implementations, it is best to avoid a BTB and instead to calculate branch target addresses on-the-fly. (2) It studies the FPGA implementation of the three most commonly used branch direction predictors (DIR): bimodal [2], gshare, and gselect [3]. The analysis corroborates the results of past studies showing that gshare achieves the best accuracy among the three for practical table sizes, but also shows that unlike an ASIC implementation, frequency suffers with gshare on FPGAs. It proposes *gRselect*, an FPGA-friendly gselect implementation that uses a simple indexing scheme to outperform gshare by 11.4%. (3) It demonstrates that a conventional Return Address Stack (RAS) maps well onto the MLABs of FPGAs improving performance with little additional cost.

The rest of the paper is organized as follows. Section 2 reviews branch prediction basics and details the goals of this work. Section 3 discusses the architecture of the various branch prediction components studied. Section 4 presents FPGA-specific optimizations. Section 5 presents the experimental evaluation results and finally Section 6 concludes.

2. BACKGROUND AND GOALS

Fig. 1 shows the organization of a typical branch predictor comprising: (1) a DIR, (2) a BTB, and (3) a RAS. 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 in-

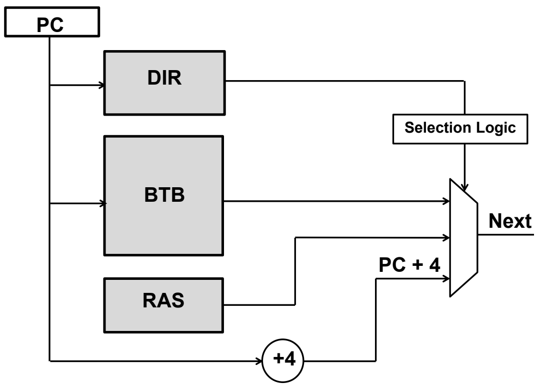

Fig. 1. Canonical Branch Predictor.

struction's PC and other dynamically collected information. The DIR guesses whether the branch will be taken or not. The BTB and the RAS guess 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), the target predicted by the BTB, or the target provided by the RAS. 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 predecode 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.

2.1. Design Goals

This work aims to design a branch predictor that (1) balances operating frequency and accuracy to maximize execution performance, and (2) uses as few on-chip resources as possible. Since Altera's highest performing soft-processor, Nios II-f, uses three BRAMs in total [4], this work limits the predictor's resource budget to one BRAM; each additional BRAM would represent a more than 1/3 overhead in terms of BRAM resources. This work considers the most commonly used direction predictors, bimodal, gshare and gselect. These predictors use a single lookup table and map relatively well onto a single BRAM. Other more accurate, albeit more elaborate predictors exist [1, 5, 6]. An investigation of these predictors is left for future work for two reasons: (1) These predictors often use multiple tables and tagged entries. Tagged entries require comparator-driven multiplexers and thus may not map well onto FPGAs. (2) For an in-order pipelined processor such as Nios II-f and the workloads studied, the potential for further performance improvement with more accurate branch predictors is negligible over the gshare and gselect predictors considered.

3. FPGA-FRIENDLY BRANCH PREDICTION

This section discusses the architecture of the various FPGA-friendly branch predictors considered. Section 3.1 discusses target prediction, while Section 3.2 discusses direction prediction.

3.1. Target Prediction

A conventional method to predict branch targets is to use Branch Target Buffers (BTB). A BTB is a table that caches branch target addresses. When a branch executes for the first time, the BTB stores the target address so that it can be used on subsequent encounters of the branch. Ideally, the BTB would be large enough so that each branch can use a separate entry. In a practical implementation, however, aliasing will occur reducing prediction accuracy.

The simplest BTB design does not use an address tag per entry and directly predicts the target address for all instructions. Not using a tag results in a fast design that uses one direct SRAM lookup. In addition, not filtering non-branch instructions is desirable since at the time of access the instruction opcode is not available. Unfortunately, as Section 5 shows, when all instructions use the BTB, high destructive aliasing results in poor accuracy. To reduce aliasing, a small decode logic can prevent non-branches from updating the BTB. However, as Section 5 shows, while this solution increases target prediction accuracy by 30%, the additional logic reduces the maximum frequency by 35%. An alternative is to calculate the target address during the fetch cycle.

3.1.1. Target Address Precalculation

ASIC processor implementations use a BTB since the cache latency dominates the clock cycle leaving no room for further action. This is not true in an FPGA implementation where memory is generally faster than logic. This creates an opportunity to precalculate the target address for branches and thus eliminate the BTB. In this scheme the processor fetches the instruction from the cache and then, during the same cycle, calculates the instruction's taken address. As an added benefit, address precalculation may improve accuracy since, if possible, it is always correct. Unfortunately, it is not possible to precalculate the target address for all branches. The Nios II ISA includes two types of branches: *direct* and *indirect*. The target of a direct branch can be calculated using the current PC and an offset that is embedded

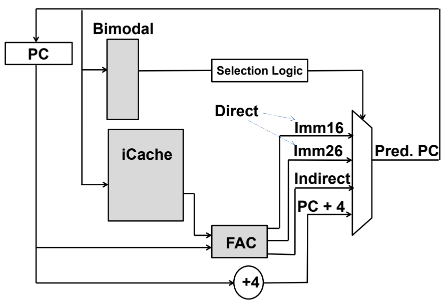

Fig. 2. BTB with Full Address Calculation.

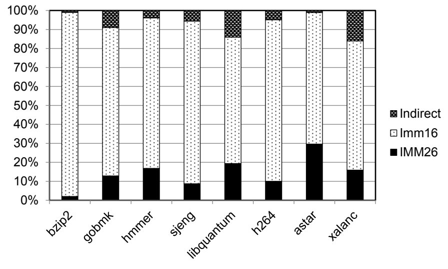

Fig. 3. Branch Target Type Distribution.

in the instruction. Indirect branch targets are read from the register file.

This work proposes enhancing target prediction with *Full Address Calculation* (FAC), which as Fig. 2 shows, calculates the target address for all direct branches and uses a BTB or some other storage for indirect branches. A selection logic identifies direct branches which can benefit from FAC. FAC selects among four possible addresses depending on the branch type. The Nios II ISA supports two schemes for direct branch target addresses, one uses a 16-bit offset (IMM16) and the other a 26-bit range (IMM26). Combined with the fall-through address (i.e., PC + 4) and the predicted address coming from the BTB, BTB+FAC uses a four-way multiplexer to select among these four possible addresses. Unfortunately, this multiplexer falls into the critical path.

A lower cost and faster alternative to FAC is *Partial Address Calculation* (PAC) which relies on typical program behavior to reduce the number of choices for the final address multiplexer. Fig. 3 reports the relative frequency of the various branch types (see Section 5.1 for the methodology). Since IMM26 branches are far less frequent than IMM16 branches, PAC precalculates IMM16 branches and uses the BTB for IMM26 and indirect branches.

3.1.2. Return Address Stack

The RAS is a hardware, stack-like structure that accurately predicts the target address of function returns. When a call instruction executes, it also pushes its return address onto the RAS. Upon fetching a return instruction, the branch predictor can pop the top value from the RAS accurately predicting the return address. As long as the RAS has enough entries, it will accurately predict all return instructions. Since the call depth of typical workloads is not deep, virtually all high performance processors incorporate a shallow RAS. For the workloads studied a 16-entry RAS proves sufficient. The RAS for a simple pipeline is simple to implement on an FPGA. Deeper pipelines may require support for speculative RAS insertions and deletions complicating its design.

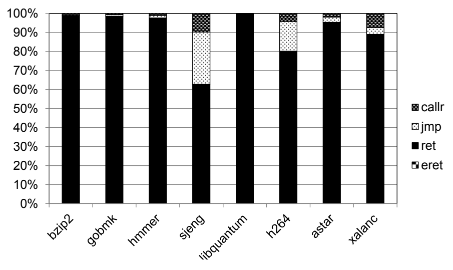

Fig. 4. Indirect Branch Instruction Type Distribution.

3.1.3. Eliminating the BTB

Fig. 4 shows that for the workloads studied in this work, 97% of the indirect branches are returns (other workloads, e.g., those using virtual functions, may behave differently). Once a RAS is included along with FAC, the BTB ends up being used for only less than 1% of all branches. Accordingly, the BTB can be eliminated, and instead use a static, not-taken predictor for all indirect branches other than returns. Section 5 shows that removing the BTB in the presence of RAS reduces accuracy negligibly. Section 4.1 explains that lower-level FPGA related considerations also favor eliminating the BTB.

3.2. Direction Prediction

A bimodal branch direction predictor is a table of two-bit saturating counters that is indexed with a portion of the PC [2]. The counters are updated up or down depending on whether the branch is taken or not respectively. The lower bit provides hysteresis to changes while the upper bit provides the prediction. As Section 5.3 corroborates, increasing the number of bimodal entries does not proportionally improve accuracy. Eventually, using a larger bimodal ceases to provide any improvement since bimodel is fundamentally limited on the branch prediction patterns it can predict.

Gshare is a pattern-based predictor wihch uses a combination of the PC and a global direction history register (GHR) to index the counter table [3]. GHR stores the direction of the last few branches in a bit vector. Each GHR bit stores the direction (taken or not) of a previous branch. Section 5 shows that gshare is far more accurate than bimodal. However, as Section 4 explains, latency suffers with gshare due to its more complex indexing scheme. Gselect, an alternative to gshare, indexes the counter table using a simple concatenation of the GHR and the PC [3]. For practical table sizes, gshare proves more accurate than gselect. Section 4.2 explains that with proper modification, gselect proves faster than gshare on an FPGA while sacrificing little in accuracy.

4. FPGA IMPLEMENTATION OPTIMIZATIONS

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

4.1. Eliminating the BTB

As Section 2.1 explained, this work aims to use one M9K BRAM. An M9K BRAM can be configured as wide as 36 bits with 256 rows [7], and it can be used to implement a fused BTB and direction predictor. Specifically, each BRAM row can store one BTB entry along with up to three direction prediction entries for a total of 768 direction entries and 256 target entries. This fused BTB+DIR predictor works well with a bimodal DIR. The PC indexes a row which contains a single target prediction entry and up to three direction prediction entries. Another portion of the PC selects one of these direction prediction entries. The target is used only from taken branches.

Unfortunately, it is not possible to use one BRAM for both a BTB and the most accurate of the direction predictors considered, gshare. There are two reasons why: (1) gshare uses a different indexing scheme than the BTB, and (2) there is a limited number of ports per BRAM [7]. As Section 3 discussed, the BTB can be eliminated when address precalculation and a RAS are used. Eliminating the BTB frees up the entire BRAM for direction prediction.

4.2. FPGA-Friendly Direction Predictor Indexing

This section investigates which of the three branch direction predictors is best to use on an FPGA. Focusing just on accuracy gshare would be the best. However, performance is not the highest with gshare since its indexing scheme results in low clock frequency. Fig. 2 depicts why the predictor's indexing scheme, when implemented on an FPGA, falls into the critical path. At every clock cycle the predicted PC is used to index the direction predictor table for the next instruction. Since BRAMs are synchronous, their index must arrive before the clock edge and thus it cannot be a registered signal of the Predicted PC [7]. Moreover, the setup time for the BRAM is longer than that of simple registers. Therefore, the entire path starting from the BRAM data output, through the prediction logic and back into the lookup

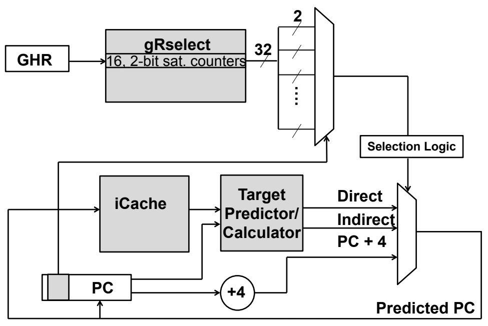

Fig. 5. FAC with gRselect.

address of the BRAM forms the critical path. This is especially a concern with gshare that uses the exclusive-or of the global history register (GHR) with Predicted PC to index the BRAM. This extra XOR logic prolongs the critical path, reducing the operating frequency.

Contrary to gshare, gselect has a simpler indexing scheme. Specifically, gselect uses a simple *concatenation* of the GHR with Predicted PC as index. Not only is gselect's indexing fast, but it can also be tailored to map well onto FPGAs. This work proposes *gRselect* which breaks the BRAM-to-BRAM critical path by breaking the BRAM access into two parts the first of which does not need to be "predicted PC". It uses GHR, which is a registered signal, to index the BRAM to retrieve one wide row of counters. Fig. 5 shows the gRselect scheme in more detail.

4.3. Instruction Decoding

To be able to select the appropriate target, the predictor needs to determine whether an instruction is a branch and if so, what kind of branch it is. This information is needed to select the corresponding predicted target through the output multiplexer. However, the decode logic lies in the critical path. To eliminate this delay, the predictor pre-decodes the instructions prior to installing them in the instruction cache. The pre-decode information is stored along with the instruction. This is similar to typical ASIC implementations.

4.4. RAS

The RAS can be implemented efficiently with MLABs, which are small-grain, distributed memory resources. Compared to a logic-based RAS, an MLAB-based RAS proves smaller and faster. Furthermore, because 50% of all LABs can be configured as MLABs, there is no routing penalty with MLABs.

5. EVALUATION

This section presents the experimental evaluation of the proposed branch predictors. Section 5.1 details the experimental methodology. Section 5.2 evaluates the accuracy of target address schemes showing that using a RAS with FAC is best. Section 5.3 compares the accuracy of various direction predictions, showing that a single BRAM gRselect is among the best performers. Section 5.4 reports resource usage and maximum operating frequency. Finally, Section 5.5 reports the overall performance showing that the predictor that combines FAC, RAS, gRselect and pre-decoding is best.

5.1. Methodology

To compare the predictors this work measures: (1) Accuracy as misses per kilo instructions (MPKI) which has been shown to correlate better with performance compared to prediction accuracy alone. Processor performance in (2) instructions per cycle (IPC), a frequency agnostic metric, that isolates the effects of implementation, and in (3) instructions per second (IPS), a true measure of performance. (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 [8], and runs a subset of SPEC CPU2006 integer benchmarks with reference inputs [9]. The evaluation uses a baseline predictor (BASE) with a fused BTB and bimodal, as discussed in Section 4.1 both with 256 entries. BASE does not decode instructions and thus uses the BTB and the bimodal for all instructions.

All designs were implemented in Verilog and synthesized using Quartus II 12.1 on a Stratix IV 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 and BRAMs used.

5.2. Target Prediction

This section measures the accuracy of target predictors. by using the baseline direction predictor while considering combinations of BTB, PAC, FAC, RAS, and larger BTBs. Fig. 6 reports the reduction in target address mispredictions when using various target prediction mechanisms compared to BASE. Using a decode logic to filter non-branches from the BTB (BTB-256) reduces mispredictions by 30%. However, increasing the BTB size to 512 (BTB-512) or 1024 (BTB-1024) entries does not improve accuracy noticeably. In the rest of this section, all BTB configurations except for BASE use instruction filtering.

Using PAC or FAC with a 256-entry BTB reduces mispredictions by 81% and 90% respectively, whereas using just FAC reduces mispredictions by 84%. Finally, using FAC with a RAS (FAC+RAS) proves best. In conclusion, eliminating the BTB and relying instead on a RAS+FAC is best in terms of accuracy. An added benefit of RAS+FAC is that it allows for a standalone, thus larger and more flexible direction predictor.

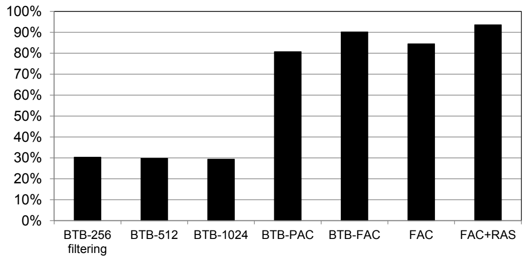

Fig. 6. Target Address Prediction Schemes: Reduction in target address misprediction over BASE.

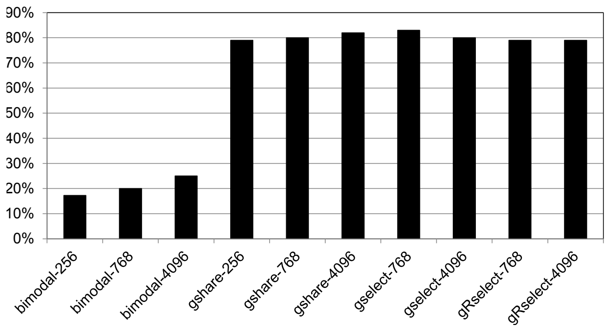

Fig. 7. Direction Predictors: MPKI improvement over BASE.

5.3. Direction Prediction

Fig. 7 reports the improvement in MPKI for various direction predictors relative to BASE. Decoding the instructions and performing prediction only for branches improves MPKI by 17% (bimodal-256). Using a larger bimodal with 4K entries further improves MPKI by only 8% suggesting that bimodal is fundamentally limited in the branch sequences it can predict. However, using a 256- or a 4K-entry gshare improves MPKI by 79% and 82% respectively.

Section 4 explained why gselect may be better to implement on an FPGA. Fig. 7 show that a conventionally indexed 4K-entry gselect results in competitive accuracy, improving BASE by 80%. Section 5.4 explained that the desired number of entries for the direction predictor is either 768 or 4K when fused with a BTB or not respectively. The figure shows that a conventionally indexed 4K-entry gselect improves MPKI by 80%, while the proposed FPGA-friendly organization, gRselect, improves MPKI by 79%. In conclusion, gshare achieves the best accuracy with gselect and gRselect offering competitive accuracies.

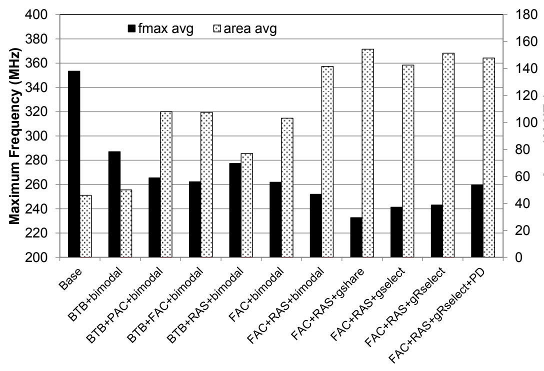

Fig. 8. Maximum frequency and area utilisation. (PD = pre-decoding)

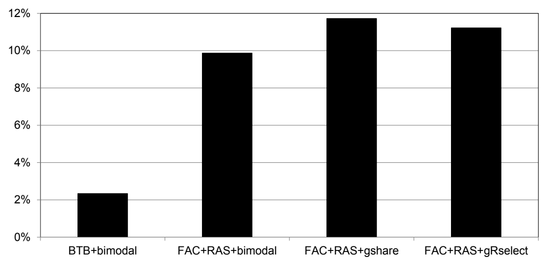

Fig. 9. Improvement in IPC over BASE.

5.4. Area and Frequency

Fig. 8 shows the maximum frequency and area utilisation for each predictor design. All configurations use one BRAM. As expected, BASE is the fastest and least expensive. Adding instruction filtering reduces fmax from 353 MHz to 287 MHz, a 18% drop. By adding address calculation, frequency drops even further. However, removing the BTB partially recovers from this frequency drop. Finally, adding a RAS to a gRselect with pre-decoding, results in a predictor that operates at 259 MHz and that uses only 147 ALUTs.

5.5. Performance

Fig. 9 reports average IPC gain compared to BASE. The bimodal predictor results in the lowest IPC while gselect performs almost as well as gshare.

IPC is proportional to performance only when the clock frequency remains the same. Actual performance depends on IPS, the product of IPC and clock frequency. Fig. 10 reports overall performance in IPS. This experiment assumes a 250MHz maximum clock speed for the processor, the maximum clock frequency of Nios-II-f on the Stratix IV [10]. The best performing predictor is a 4K-entry gRselect with FAC+RAS, no BTB, and that uses pre-decoded instructions.

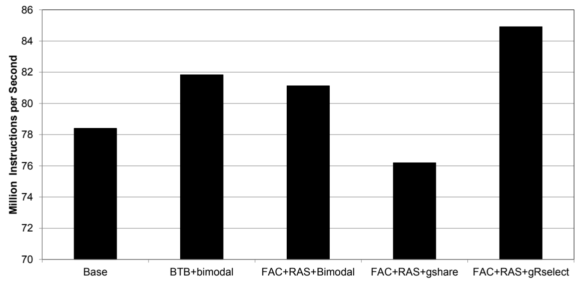

Fig. 10. IPS comparison of processors with various predictors.

6. CONCLUSION

This work studied the implementation of branch predictors for general purpose soft processors. This work targeted high frequency, low area overhead branch predictors for pipelined processors, and explored various branch predictor designs. These designs were combinations of a branch target buffer, a return address stack, three commonly used direction predictors, pre-decoding, in-fetch instruction decoding, and target address calculation. Several FPGA-specific optimizations were proposed resulting in a branch predictor that is FPGA-friendly in that it offers high accuracy, high operating frequency with few resources. Future work may consider more elaborate processor designs and/or other workloads that may benefit from more elaborate branch direction predictors than the ones considered in this work.

7. REFERENCES

[1] 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.

[2] J. E. Smith, "A study of branch prediction strategies," in *Annual Symposium on Computer Architecture*, June 1981.

[3] S. McFarling, "Combining Branch Predictors," Digital Western Research Laboratory," TN-36, June 1993.

[4] *Nios II Processor Reference*, Altera Corporation, May 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, "The O-GEHL Branch Predictor," in *The 1st JILP Championship Branch Prediction Competition*, December 2004.

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

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

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

[10] *Nios II Performance Benchmarks*, Altera Corp., Dec. 2012.