LA-UR-17-25469
Approved for public release; distribution is unlimited.

Title: A Probabilistic Monte Carlo Framework for Branch Prediction

Author(s): Kalla, Bhargava
Santhi, Nandakishore
Chennupati, Gopinath
Eidenbenz, Stephan Johannes
Badawy, Abdel-Hameed

Intended for: IEEE Cluster 2017, 2017-09-05/2017-09-08 (Honolulu, Hawaii, United States)

Issued: 2017-07-06 (Draft)

A Probabilistic Monte Carlo Framework for Branch Prediction

Bhargava Kalla*, Nandakishore Santhi†, Abdel-Hameed A. Badawy‡, Gopinath Chennupati§, Stephan Eidenbenz¶
*School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA
†§¶Information Sciences Group, Los Alamos National Laboratory, Los Alamos, NM, USA
‡Klipsch School of Electrical & Computer Engineering, New Mexico State University, Las Cruces, NM, USA
*bkalla@asu.edu,‡badawy@nmsu.edu,{†nsanthi,§gchennupati,¶eidenben}@lanl.gov

***Abstract*—Branch prediction is crucial in improving the throughput of microprocessors. It reduces branching stalls in the pipeline, which helps to maintain the instruction execution flow. Of these instructions, conditional branches are non-trivial in determining the microprocessor performance and throughput. Modern microprocessors accurately predict the branches using advanced branch prediction techniques. Appropriately estimating the branch mis-predictions benefits to improve the overall performance of an application through effectively saving the CPU cycles. In general, collecting branch prediction statistics using state-of-the-art simulators is time consuming and not scalable. We present a novel Monte Carlo simulation framework that predicts branch mis-prediction rate. Our framework produces results that suggest that the mis-prediction rates on three scientific applications are similar (with an average difference of 0.3%) to that of a Markov model of a 2-bit saturating branch predictor.**

I. INTRODUCTION

Reducing branching stalls in microprocessors through accurate branch prediction techniques improves the performance by reducing the CPU stall cycles, thereby reducing power. Branch prediction often influences the execution of the following instructions, which in turn has adverse effects on the execution flow of a program. These changes in the execution have significant impact on the instruction throughput and latency. Using hardware simulators such as GEM5 [4] to predict branch behavior is often time consuming and such simulations are not scalable.

We present a novel framework which consists of a Probabilistic Finite State Machine (PFSM), that uses a Monte Carlo approach and employs a 2-*bit saturating counter* branch predictor in predicting the branch misses. PFSM relies on the concept of LLVM basic blocks (has a single entry and a single exit without any iterations) while accepting the control flow graph (measured through static analysis) among the basic blocks as an input. PFSM is flexible, easy to understand and scalable, which serves the users with branch mis-prediction rate without executing the benchmark under investigation. In contrast to branch predictors in simulators such as GEM5 [4], PFSM mimics the actual program execution without really executing it. Additionally, PFSM provides the mis-prediction rate of individual branches. The experiments on three applications: `BlackScholes`, Matrix Multiplication (`MM`) and `STREAM` indicate that the branch predictions are approximately similar to that of Markov analytical model for a 2-bit saturating branch predictor.

II. PROBABILISTIC FINITE STATE MACHINE (PFSM)

Our framework, PFSM uses a Monte Carlo approach to simulate the execution of a program that mimics the execution of a program on a target hardware/machine. First, we compile the source code into LLVM's intermediate representation (IR) that contains the basic blocks. Through an off-line static analysis, we derive the control flow graph of the basic blocks with edges representing the probability of a transition from a predecessor basic block to a successor. PFSM accepts the branching probabilities of the basic blocks and the control flow graph as input, where each node acts as a state in the FSM. We maintain a record of individual branches in a Branch History Table (BHT) using 2-bit saturating counter branch predictor (see Section II-A).

The PFSM probabilistic simulation works as follows. For example, if an i^{th} basic block contains three conditional branches, we randomly select one of the three outgoing branches using a uniform random number generator. Basically, we generate a random number uniformly in the interval (0,1), where we compare the random number with the above three outgoing probabilities of the i^{th} basic block and select one among those branches. The remaining basic block branches are considered to be not taken. Depending on the decision of either a branch being taken or not, we update the entries in BHT with the help of the 2-bit saturating counter. If we encounter an unconditional branch and/or a function call, the model continues from the initial basic block of that function or the unconditional branch/basic block. When we encounter a terminating basic block (which has no branches), that is the basic block with no outgoing probabilities available, we check whether it is returning to a function call or the end of the control flow. Depending on the termination condition, we either return to the calling function or terminate the execution of the program.

A. 2-bit Saturating Counter Branch Predictor

A 2-bit saturating counter branch predictor is a four state FSM that traverses between the user-defined states Strongly Taken(ST), Weakly Taken(WT), Weakly Not Taken(WN) and Strongly Not Taken(SN) as shown in Figure 1. We use a BHT

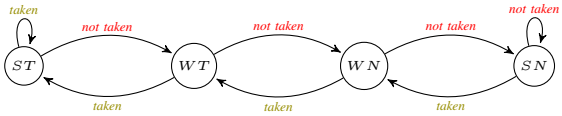

Fig. 1. FSM of a 2-bit saturating counter branch predictor. The application's PFSM is context dependent, typically complex, and not shown here.

which is stored in memory. Each BHT entry holds a 2-bit value that records the history of an individual branch instruction to one of the four mentioned states depending on its actual outcome. BHT is indexed by the least k significant bits of the branch instruction's address (i.e. Program Counter).

B. Mis-Prediction Rate

We compute the mis-prediction rate of an individual branch as the sum of the product of the three probabilities: probability of a basic block being executed (P_{BB_i}), transition probability of a predecessor basic block to the successor (Π_{ij}) and conditional mis-prediction rate of transition from the i^{th} basic block to the j^{th} basic block ($P_{miss-predict_{ij}}$). Therefore, the sum of these individual branch mis-prediction rates for all the basic blocks of a program as shown in Equation 1 represents the branch mis-prediction of a program ($P_{mis-prediction}$).

$$P_{mis-prediction} = \sum_i \sum_j P_{BB_i} \times \Pi_{ij} \times P_{mis-predict_{ij}} \quad (1)$$

C. Characteristics of PFSM

A few unique characteristics of PFSM are: (1) the modular nature of PFSM offers the flexibility of adding advanced branch prediction techniques along with the existing techniques. Such flexibility helps to compare the mis-prediction rates across multiple branch prediction techniques. We can also replace the 2-bit saturating counter with a superior technique. (2) PFSM is architecture, hardware and OS independent since we are employing LLVM instrumentation and conducting a static analysis before hand. (3) Another advantage is that we are statically producing a probabilistic FSM, without actually executing the application on any hardware. Our probabilistic framework acts as an efficient proxy for the application, which is different from existing simulated environments.

III. Experimental Results

We have computed the mis-prediction rates for three benchmarks `STREAM`, `Matrix Multiplication (MM)`, and `BlackScholes` [5], while varying the BHT size (k). In order to obtain stable results, we run the simulations at least $\lceil 100/min(\Pi_{ij}) \rceil$ times. Figure 2 shows the mis-prediction rate at for different BHT sizes (k) for the three applications. For `BlackScholes`, for k between $(2-6)$, there is fluctuation in the mis-prediction rate due to the aliasing effects while indexing into the BHT, as the BHT size can be less than the number of branches in the program. For the other two benchmarks, the mis-prediction rate remains almost constant with slight difference.

We can infer from the results that for BHT sizes greater than the number of branches in the benchmark, the mis-prediction

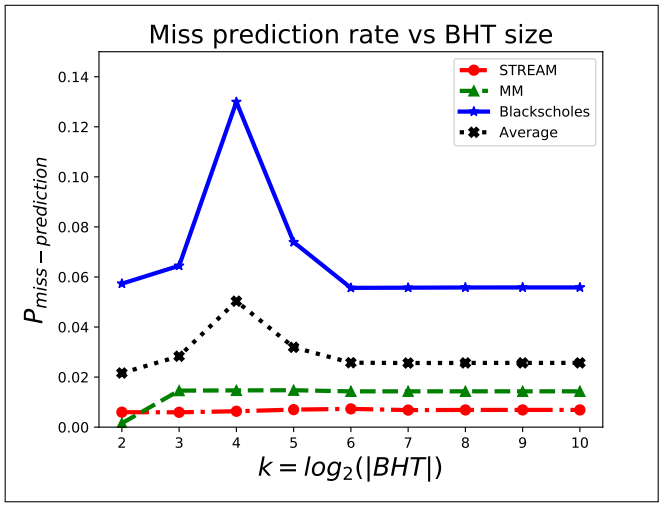

Fig. 2. Mis-prediction rate for different BHT sizes, when a 2-bit predictor is used. Branch aliasing effects can be observed for small k values.

rate is low and remains constant because of the fact that every branch has its own BHT entry and also there is no possibility of aliasing while indexing BHT.

We verify our framework with an analytical Markov model of the 2-bit saturating counter branch predictor [1]. We compute the mis-prediction rates for these three benchmarks. The mis-prediction rates computed for `BlackScholes`, `MM`, and `STREAM` using the analytical model [1] (without any aliasing effects) are: 6.04%, 1.41%, and 0.34% respectively while they are: 5.56%, 1.48%, and 0.66%, respectively for PFSM. The average difference between the PFSM and the analytical model [1] is 0.3%. The slight discrepancy can be attributed to the nature of the Monte Carlo simulation. Thus, from the obtained results we conclude that our framework is functionally correct in terms of mimicking the execution behavior and estimating the branch prediction statistics.

IV. Conclusions & Future Work

Our framework follows a novel approach to estimate the branch prediction accuracy without program execution. In the future, we will incorporate more advanced branch prediction schemes such as tournament predictor in our framework to compare various predictors. Also, we will combine PFSM branch prediction capability with the hardware memory models, pipelines and network interconnects in order to predict the performance of various applications at scale.

References

[1] Elkhouly, Reem and El-Mahdy, Ahmed and Elmasry, Amr, 2-Bit Branch Predictor Modeling Using Markov Model, in: Procedia Computer Science, Vol. 62., pp. 650–653, Elsevier, 2015.
[2] J. E. Smith, A study of branch prediction strategies, in: Proceedings of the 8th annual symposium on Computer Architecture, IEEE Computer Society Press, 1981, pp. 135148.
[3] T. Sherwood, B. Calder, Automated design of finite state machine predictors for customized processors, in: Proceedings of ISCA, 2001.
[4] Nathan Binkert, et.al., The gem5 simulator. SIGARCH Comput. Archit. News 39, 2, August 2011.
[5] Gopinath Chennupati, et.al., An Analytical Memory Model for Performance Prediction. In Press WinterSim 2017.