STEVES: Pushing the Limits of Value Predictors with Sliding FCM and EVES

Arpit Gupta, Parv Mor, Hritvik Taneja, Biswabandan Panda
{arpitrag, parv, hritvikt, biswap}@cse.iitk.ac.in
Indian Institute of Technology Kanpur

ABSTRACT

Value prediction has immense potential in improving performance of the modern day processors. Based on the CVP leaderboard, the ideal value predictor provides 107% speedup whereas the state-of-the-art EVES predictor provides 35.8% with an unlimited storage. Through this manuscript, we attempt to push the limits of EVES. We propose a finite context method based value predictor named Sliding FCM. A sliding window FCM stores the history of values for a particular instruction pointer and while predicting in the future, it matches the recent history of values with a relatively older set of values. In case of a match, it predicts the next value from the history of values as the value to predict. Sliding FCM coupled with EVES (we call it STEVES) improves the average performance to 37% with a maximum speedup of 1000%.

1. INTRODUCTION

Instruction Level Parallelism (ILP) is hindered by dependencies in a program. False dependencies arising due to reuse of registers or order of instructions can be eliminated using register renaming, out-of-order processing; while true dependencies arising due to control flow and data flow are difficult to eliminate as the results are required to be predicted reliably to justify their utility. State-of-the-art branch predictors are able to eliminate control flow dependencies with a high accuracy. Value prediction, however, is fundamentally hard mainly due to large sample space and high penalty on incorrect prediction (a load might visit main memory providing room for more instructions to be speculated). The usefulness of value prediction was first shown by Lipasti et al. [1] The idea was disposed off due to the complex architecture it seemed to demand. However, recent proposals such as EOLE [2], an {Early | Out-of-Order | Late} Execution architecture, has made value prediction to be practical. Value predictors, as proposed by Smith et al. [3], are either a hybrid of or belongs to the following categories:

1. ***Computational Predictors.*** The result of these predictors is an application of some function of previously seen values. Example of such predictors include, Last Value Predictor and Stride Predictor.

2. ***Context based Predictors.*** These predictors observe the context of the values seen from an instruction and predict a previously seen value on repetition of the same context in future. Example of such predictors include, Finite Context Method (FCM) Predictor.

In this paper, we propose Sliding FCM — a value predictor of the second kind. Sliding FCM is a context based predictor that maintains sliding windows per predictable instruction. We analyze the accuracy and performance of Sliding FCM with state-of-the-art first championship value prediction winner predictor — EVES [4]. We observe that Sliding FCM differs from EVES (in terms of coverage) by a significant amount with a maximum difference of over 25%. Further, we integrate Sliding FCM with EVES to build a hybrid predictor, STEVES — Sliding The EVES, and observe an improvement in Instructions Per Cycle (IPC) by a maximum of over 1.69.

Context based predictor learns previously observed values to predict values of subsequent occurrences of the instruction. FCM predictors, defined by their order k, use the previous k values of an instruction to index into a global or per instruction table. The table maintains count of values that immediately follow the k values and depending upon confidence the value with maximum count is chosen to be predicted. In the following subsections we argue the disadvantages of such designs of context based predictors and Sliding FCM which tries to eliminate the mentioned disadvantages. Further, we discuss appropriate parameters and optimizations that can help maximize the performance.

Need of a new Context Predictor:

a. In general, FCM is biased towards old history of a program and realizes the change in instruction's behavior after a considerable number of cycles. Since FCM weighs equally to the least recently occurred value and the most recent occurred value, FCM doesn't provide a bias to more recent patterns than earlier ones which is provided by Sliding FCM by only accounting for "m" most recent values for a particular PC. During this period it might miss the chance to predict or in the worst case even mispredict leading to poor performance. Hence, there is a need to adapt to the recent changes of program's behavior in a swift manner.

b. Conventional FCM can lead to inconsistent behaviour even with a small in-flight window. To resolve this one might have to store I, expected number of in-flight instructions, values immediately following an index which results in a lot of redundant data. Hence, there is a need for a space efficient model.

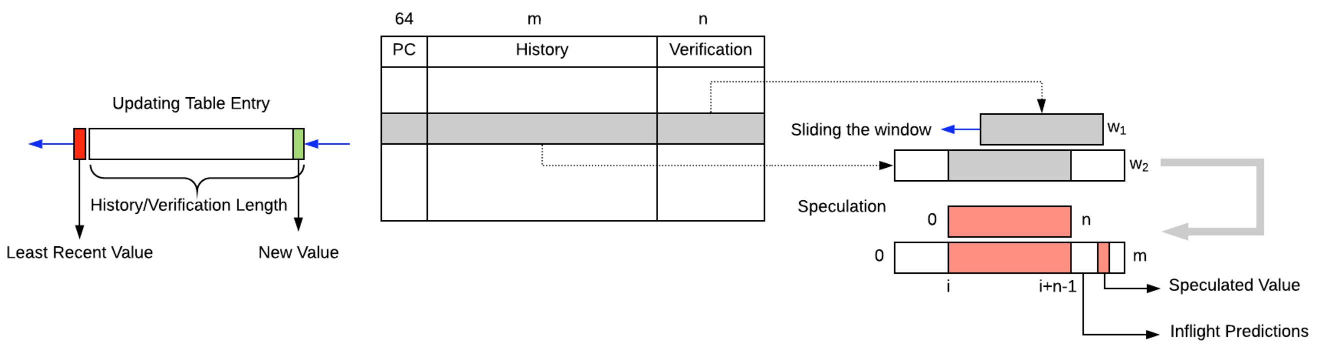


Figure 1: Value Prediction using Sliding FCM.

2. SLIDING FCM

Sliding FCM makes use of two value streams seen in recent past. The first stream i.e. the verification stream is of length n (the verification length) while second stream i.e. the history stream is of length m (the history length). Since we need to verify n values before the inflight pattern (of length I), m needs to be greater than $n+I$. We denote the streams in form of an array (zero-indexed) where the verification and the history stream corresponds to w_1 and w_2 respectively. Array w_1 is a sub-array of array w_2 with respect to some i, if and only if,

$$\forall\, 0 \leq j < n \implies w_1[j] = w_2[i+j]$$

Further, we call w_1 to be a early-matched sub-array of w_2 with respect to some i, if w_1 is a sub-array of w_2 and i is the maximum possible.

To make a prediction, we find an early-matched sub-array such that it can accommodate the in-flight instructions. If the match occurs at index i, then we predict the value to be $w_2[i+n+I]$. As the match occurs we first verify the n (verification length) values and now since we have I occurrences of the same instruction in inflight, we cannot verify them and assume them to be correct and predict the next value without verifying the I values which is $w_2[i+n+I]$ Figure 1 depicts the prediction mechanism pictorially. The predictor table is indexed using the Program Counter (PC). The content of each entry in the predictor table consists of a value history of length m, most recent m values, predicted values of inflight instructions and, correct and incorrect predictions (for blacklisting the entry). Updates to each window stream is made when the actual value of instruction is known, i.e., during commit. To update a window stream the value at lowest index is dequeued and the newly found value is queued at the highest index.

Since we only take last m values into account for the history, our predictor Sliding FCM has the capability to rapidly re-adjust to the program behaviour. Sliding FCM is a clean solution for handling in-flight occurrences without using too much space. Moreover, we do not need any probabilistic counters for our predictor since the verification length is a measure of the confidence in itself.

2.1 Parameter Tuning

We describe the optimal parameters used in our experiments and their effect on accuracy and coverage of the predictor. The different parameters are described as follows:

a. *Length of History Window.* We compare the most recent n values of the instruction by the sliding across the history window (of length m) of that particular instruction. The history length m relates to the program state. An extreme case where $m \to \infty$ tends to rely too much on early history of program. Experiments show that large values of m significantly increase number of incorrect predictions and small values of m leads to lower coverage. We found that $m = 64$ provides optimal coverage and accuracy.

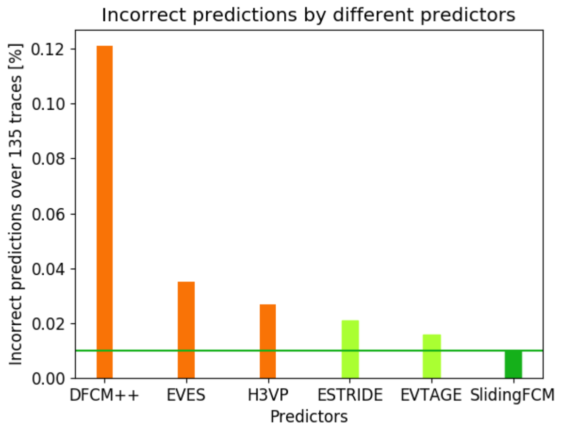
b. *Verification Length.* Verification length, n, is the length of the previous values of current context that we slide to match with history window. Small values of n increases coverage at the cost of large number of incorrect predictions, whereas large values of n decreases mispredictions at the cost of low coverage. The optimal value of n was found to be 32. We see that increasing the value of n beyond 32 doesn't increases accuracy whilst maintaining high coverage levels.

c. *Instruction Blacklisting.* In this work we aim to explore the limits of value prediction, hence we do not restrict ourselves to any storage budget. Yet to have practical realization of our approach we have included the feature for an instruction blacklisting in our model. With this feature we stop predicting an instruction with a threshold inaccuracy. Further, space optimizations such as aging and skewed-associativity can be employed.

d. *In-flight Length.* We verified that most of the traces provided by CVP do not require an in-flight window of more than 32. Our optimal history length and verification length luckily satifies the constraint mentioned in Section 2.

3. HYBRID PREDICTOR

While program exhibits patterns in their value stream, not all of them, can be predicted using a context-based predictor. Computational-based predictors are indeed necessary for better value prediction coverage and hence, better performance. In the following subsections we discuss two state-of-the-art



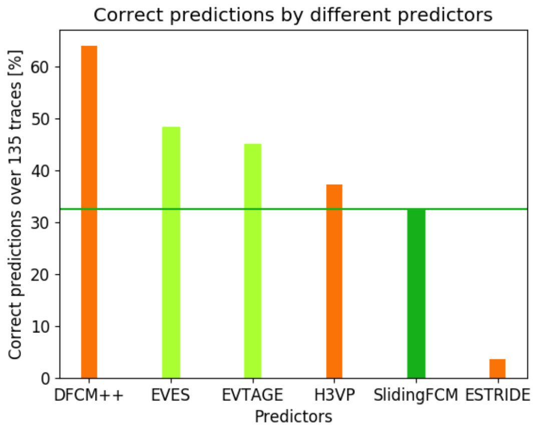


Figure 2: Average number of incorrect (left) and correct (right) predictions of Sliding FCM compared to unlimited budget predictors of CVP leaderboard.

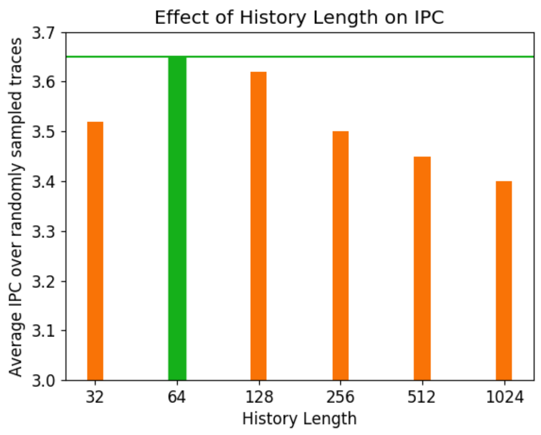


Figure 3: Effect of history length on IPC in STEVES.

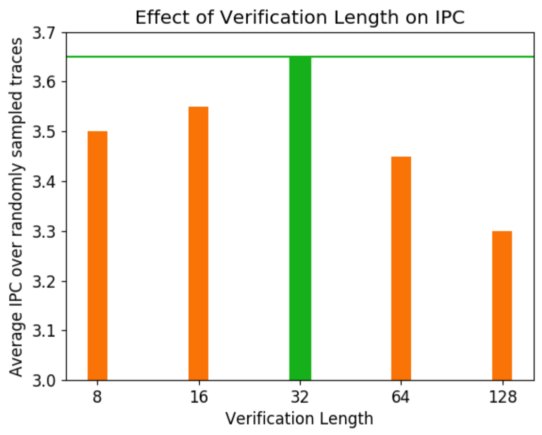


Figure 4: Effect of verification length on IPC in STEVES.

computational predictors and later on describe our hybrid predictor, STEVES — Sliding The EVES.

3.1 E-Stride: A Stride Predictor

Stride predictors learn the difference between consecutive occurrences of an instruction and predict value of an upcoming instruction by adding the difference to its last occurrence [1]. Formally, consider an instruction whose value stream known until now is $a_1, a_2, \ldots, a_n$ such that $\forall\ i, j\ a_i - a_{i-1} = a_j - a_{j-1} = s$ (say). Now suppose there are I occurrences of the same instruction in-flight. Then for the current occurrence the predicted value would be

$$a_n + (I+1) \times s$$

E-Stride or Enhanced Stride, proposed by Seznec [4], employs several enhancements over the basic stride predictor to improve performance. The enhancements are described as follows:

a. *Space Optimization.* Strides with null entries are avoided (as a last value predictor can be used) and only small strides are stored in a skewed associative cache. Further an aging mechanism is used to evict least recently used entries.

b. *Critical Predictions.* Values are predicted only when the corresponding instruction has high confidence. Confidence is maintained using probabilistic counters which are increased with high probability for the instructions that are speculated to be critical. This would prioritize instructions that tend to stall the pipeline for long periods.

c. *Reducing Mispredictions.* Instructions that tend to produce incorrect predictions are blacklisted until they regain confidence. Such an enhancement was also proposed by Deshmukh et al. [5]

3.2 E-VTAGE: A Last Value Predictor

Last value predictors are used to predict instructions with repeating patterns. VTAGE, based on ITTAGE [6] (an indirect branch predictor), is a last value predictor that takes into account the branch history path taken to reach the predictable instruction. VTAGE is composed of several tables where

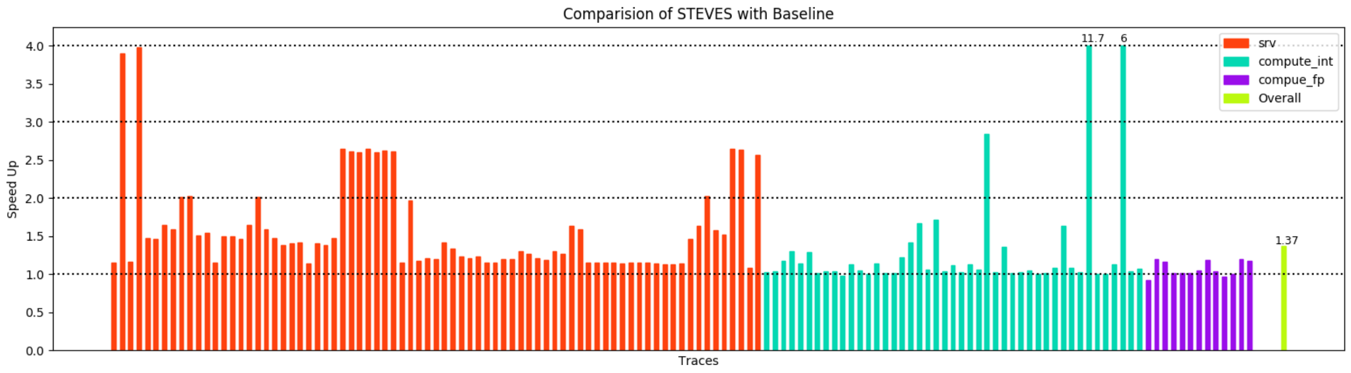


Figure 5: Speedup of STEVES compared to the baseline predictor over traces provided by CVP.

each table is indexed by PC hashed with different branch history length (that are in geometric progression). In case multiple tables provide prediction, one with longest history length is chosen.

Similar to E-Stride, E-VTAGE or Enhanced VTAGE was proposed by Seznec [4] with clever enhancements. These involve efficient space management by two level indexing and moving redundant values to a common skewed associative structure; improving performance by prioritizing critical instructions; and reducing incorrect predictions by blacklisting.

3.3 STEVES: A Composite Predictor

We analyzed the coverage, performance and accuracy of E-Stride, E-VTAGE and Sliding FCM and observed that different predictors have significantly different characteristics. As a result, we integrated the three to build a hybrid predictor STEVES — Sliding The EVES. Experiments show that STEVES has a better coverage than any of its component predictors without compromising the accuracy. This results in an overall performance improvement in terms of IPC. We observe that since all the component predictors have minimal mispredictions, prioritizing any predictor over other has negligible impact over performance. This obliviates the need for a dynamic scheduler and hence, we have the following static priority order: E-VTAGE, E-Stride, Sliding FCM.

4. EVALUATION

We evaluate the utility of our proposed predictors (Sliding FCM and STEVES) using a trace-based simulator provided by Championship Value Prediction (CVP). The predictors were simulated over the 135 traces provided by the CVP. First we depict the behavior of history and verification length on performance of Sliding FCM. Then we proceed to compare our predictors with state-of-the-art predictors mentioned on the leaderboard of CVP.

Length of Windows: From Figure 3 and Figure 4 we can conclude that optimal history length and verification length are 64 and 32 respectively. Increasing or decreasing either results in decrease in performance due to loss in coverage or increase in number of incorrect predictions.

Accuracy of Predictions: Figure 2 shows the average number of correct and incorrect predictions over given traces. We observe that Sliding FCM results in very low number of incorrect predictions owing to the window lengths. While the number of correct predictions is low they differ significantly from rest of the shown predictors. This justifies the construction of the hybrid predictor STEVES.

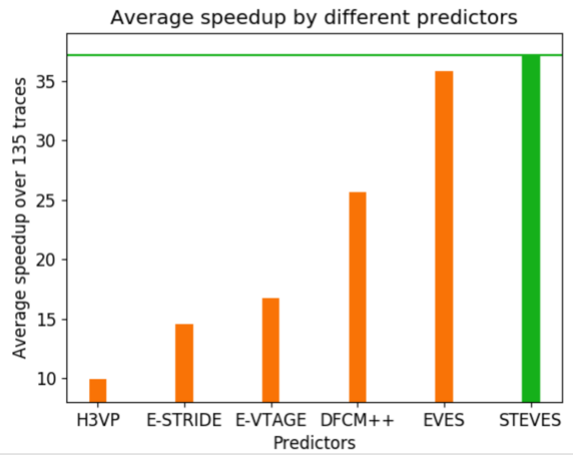


Figure 6: Speedup of STEVES compared to the unlimited budget versions of state-of-the-art predictors as per CVP-1.

Performance Improvement: Maximum improvement observed in IPC was over 1.69. Figure 5 presents the speed-up provided by STEVES over the baseline. We observe an average speedup of 37% and a maximum speedup of over **1000%**. Finally, Figure 6 shows the improvement of STEVES over unlimited budget versions of different state-of-the-art predictors [5] [7] [4].

5. CONCLUSIONS

In our work, we aim to realize the potential of value prediction for modern day workloads. Hence, we test different predictors without any constraints on the storage budget. Although the size of our predictor is dynamically growing, but to limit the size we can resort on techniques such as Blacklisting and Ageing to limit the size of our predictor. The experiments show that STEVES outperforms most of the state-of-the-art predictors in terms of performance and coverage. On an average, STEVES provides an IPC of 3.812 over 135 traces compared to 3.773, provided by the championship value prediction winner EVES.

We foresee future work on several different fronts. Firstly, a predictor for load instructions can be made via speculatively capturing the dependencies between value generating instructions, store instructions and load instructions and, maintaining a small data cache. We tried this approach but were not able to work it through because of the restrictions posed by the championship simulation framework. We believe that this approach can be easily realized in hardware. Secondly, because of the constraints posed by the current simulator, a more exposed and realistic simulator can be proposed which can expose hardware structures like load-store queue, that can help in developing better predictors.

6. REFERENCES

[1] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," vol. 30, (New York, NY, USA), pp. 138–147, ACM, Sept. 1996.

[2] A. Perais and A. Seznec, "Eole: Paving the way for an effective implementation of value prediction," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 481–492, IEEE, 2014.

[3] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 30, (Washington, DC, USA), pp. 248–258, IEEE Computer Society, 1997.

[4] A. Seznec, "Exploring value prediction with the eves predictor," in *1st Championship Value Prediction (CVP-1)*, 2018.

[5] N. Deshmukh, S. Verma, P. Agrawal, B. Panda, and M. Chaudhuri, "Dfcm++: Augmenting dfcm with early update and data dependence-driven value estimation,"

[6] A. Seznec, "A 64-kbytes ittage indirect branch predictor," in *JWAC-2: Championship Branch Prediction*, 2011.

[7] K. Koizumi, K. Hiraki, and M. Inaba, "H3vp: History based highly reliable hybrid value predictor,"