A Shadow Dynamic Finite State Machine for Branch Prediction: An alternative for the 2-bit Saturating Counter

Saleh Abdel-Hafeez[1], Ann Gordon-Ross[2*], Asem Albosul[1], Ahmad shatnawi[1] and Shadi Harb[2]
[1]Department of Computer Engineering,
Jordan University of Science & Technology
Irbid, Jordan 21110
sabdel@just.edu.jo
[2]Department of Electrical & Computer Engineering
University of Florida, Gainesville, FL 32611, USA
ann@ece.ufl.edu
Also with the NSF Center for High Performance Reconfigurable Computing (CHREC) at UF

Keywords: Branch Predictor, Bimodal, Finite State Machine, SDFSM, SPEC2000, Saturated Counter, SimpleScalar

Received: July 2009

We propose an adaptive learning machine-based branch predictor – the shadow dynamic finite state machine (SDFSM) – that enables more accurate branch predictions by learning unique branching patterns through a self-modifying technique. SDFSM states represent branch pattern bits. If a state mispredicts a branch, the state is swapped with its shadow state, which represents the correct branching pattern bit. Therefore, the prediction accuracy can reach 100% if the number of states matches a branch's pattern length. When compared to a 2-bit saturating counter using bimodal branch predictors, the SDFSM decreases average misprediction rates by 18.3%, with individual decreases as high as 55%.

1 Introduction and Related Work

In order to meet high performance demands, modern processor architectures exploit varieties of dynamic branch prediction topologies ([4]-[6] provide an excellent introduction and research coverage) to increase instruction-level parallelism (ILP).

Dynamic branch predictors use run-time branch execution history to predict branch direction. Most previous techniques use a branch pattern history table (known as PHTs, BHTs, or BPHTs) to record past branch behavior (e.g., global and/or local) and these tables are indexed using a function/subset of the branch address. Nearly all dynamic branch predictors explored in the last 10 years have been based on tables containing 2-bit saturating counters [7][8]. Extensive simulations of branch predictors reveal that the 2-bit saturating counter performs the best on average [9][10], and thus are used in modern commercial processors.

In recent years, research has explored more advanced branch prediction techniques such as neural networks [11][12] and other forms of machine learning. Despite their impressive simulation accuracy, to the best of our knowledge no commercial efforts have publicly announced incorporating such branch predictors because these branch predictors are commonly known to exhibit high prediction latency and long training periods with increased area and energy per prediction [13].

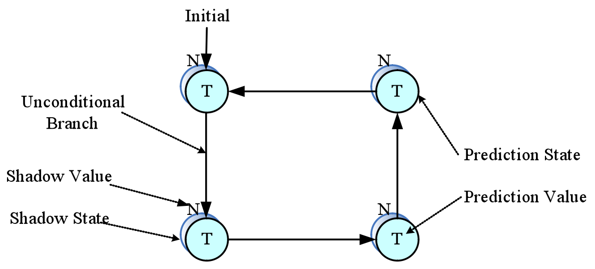


Figure 1: The proposed shadow dynamic finite state machine (SDFSM) using four states

In order to provide increased branch prediction accuracy with low area and power overheads, in this paper we propose a novel adaptive learning machine-based shadow dynamic finite state machine (SDFSM). The SDFSM learns/predicts an application's unique branching pattern using the prediction values (taken/not taken) stored in each state. Upon branch execution, state transition is input independent and the value of the target state predicts the branch outcome. Each state has a corresponding *shadow* state, which contains the alternate branch prediction value. In the event of a mispredicted branch, the SDFSM performs self-modification by swapping the current state with the current state's shadow state, which contains the correctly predicted branch outcome. This method of state swapping dynamically records unique branch patterns, thus specializing the branch predictor to the needs of an application. Extensive experimental results compare the SDFSM prediction accuracy to the commonly used bimodal [1][2] counter-based predictor and reveal that, for a subset of benchmarks, an SDFSM with six shadow states provides more accurate predictions than counter-base predictors with one-to-one prediction latency.

The remainder of this paper is organized as follows. Section 2 describes the proposed SDFSM as an alternative replacement for 2-bit saturating counters and presents the SDFSM architecture. Section 3 and Section 4 present our simulation methodology setup and branch predictor analysis, respectively. Section 5 compares counter-based predictors and SDFSM-based predictors. Section 6 presents a performance analysis and finally, section 7 gives conclusions and suggested future dynamic branch prediction development.

2 Shadow Dynamic Finite State Machine (SDFSM) Branch Prediction

In this section, we present our shadow dynamic finite state machine (SDFSM) branch prediction technique for learning/predicting an application's unique branching patterns.

2.1 SDFSM Operation

Figure 1 depicts the SDFSM using a 4-state SDFSM automaton (larger SDFSMs are similarly represented using more states). SDFSM state values record/predict branch outcomes. SDFSM operation consists of two phases: the training phase and the operational phase. During the training phase, SDFSM states are manipulated such that they *learn* the application's branching pattern. SDFSM state transition is deterministic upon each branch execution and the next state's value corresponds to the predicted branch outcome. In other words, branch prediction is determined by the branch history pattern and not by the input condition leading to the next state. If a state's prediction value is correct, no change is made to the SDFSM. If a state's prediction value is incorrect, the SDFSM self-modifies to adapt to the branching pattern.

In order to learn branching patterns, each state has a corresponding shadow state (positioned adjacent to the state), and the shadow state contains the opposite prediction value. Thus, if a state's value does not correspond to the branching pattern, the state is swapped with its shadow state in order to swap the state's branch prediction value. During the training phase, the states record the observed pattern and during the operational phase, the states predict taken/not taken. This implies that the SDFSM learns a distinct pattern on-the-fly and then predicts this pattern perfectly. Furthermore, the training and operational phases are not necessarily mutually exclusive as the SDFSM transitions to the training phase anytime there is a misprediction.

Figure 2 illustrates the 4-state SDFSM using a repeated pattern of 1010, which is commonly known to produce poor prediction rates for saturating counter techniques [14]. All state values are initialized to 0.

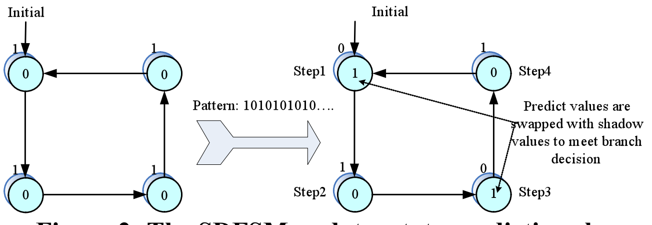
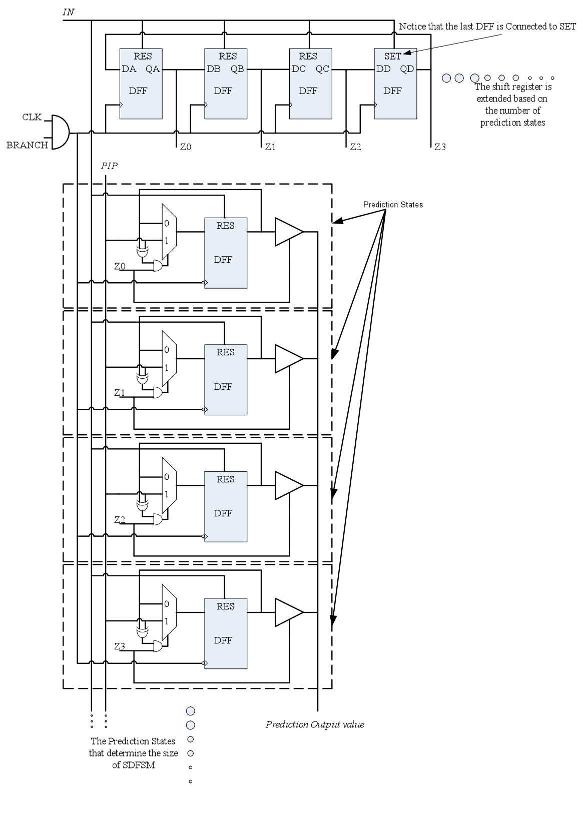


Figure 2: The SDFSM updates state predictions by swapping states with shadow states based on the observed pattern

Upon first execution of the branch, the SDFSM enters the initial state (step 1), whose state value is 0 and predicts the branch as not taken. After branch resolution, if the state mispredicted the branch outcome, the state is swapped with its shadow state and the state's predicted value becomes 1. On the next execution of the branch, the SDFSM transitions to the next state (step 2), which correctly predicts the branch as not taken. On the next execution of the branch, the SDFSM transitions to the next state (step 3), which predicts the branch as not taken. Again, branch resolution determines that the branch was mispredicted and the shadow state is swapped in. On the next execution of the branch, the SDFSM transitions to the next state (step 4), which correctly predicts the branch as not taken. On the next execution of the branch, the SDFSM transitions back to the initial state, which ends the training phase and begins the operational phase. The SDFSM now correctly predicts the branch outcome on every branch execution.

Perfect branch pattern prediction only occurs if the pattern repeats itself with a repetition cycle equal to (or a divisor of) the number of states. A 4-state SDFSM can perfectly predict any 2- or 4-entry branch pattern. This restriction can be generalized to any *x*-entry pattern, which would require an SDFSM with *x* states or any multiple of *x* states. In Section 4, we provide an in-depth analysis of numerous SDFSM sizes.

During context switching, in addition to traditional branch predictor state saving techniques, SDFSM operational state can be quickly saved and restored using special hardware to read and save state on a single clock cycle. State saving area overhead would be small, as



IN
Notice that the last DFF is Connected to SET
RES
DA QA
DFF
RES
DB QB
DFF
RES
DC QC
DFF
SET
DD QD
DFF
The shift register is extended based on the number of prediction states
CLK
BRANCH
Z0
Z1
Z2
Z3
PIP
Prediction States
0
1
RES
DFF
Z0
Z1
Z2
Z3
Prediction Output value
The Prediction States that determine the size of SDFSM

Figure 3: SDFSM predictor hardware structure

Component Type	Number of components
D-Type Flip-Flop (DFF)	2N
Two-input Multiplexer	N
AND gate logic	N
XOR gate logic	N
Tri-state gate logic	N

Table 1: Total number of hardware components based on the number of SDFSM prediction states (N).

only one n-bit counter is required for each context.

Currently, SDFSM operation is not pipelined, thus mispredicted branches and branch overlap are not accounted for. However, these operational enhancements could be easily incorporated into the SDFSM by adding additional steering logic and mispredicted rollback capabilities. These additions would be straightforward and could be done such that the prediction accuracy would be unaffected, and are a focus of our future work.

2.2 SDFSM Architecture

Figure 3 depicts the generalized SDFSM architecture (with *N* states) consisting of an array of *N* prediction states and a shift register to selectively enable the appropriate prediction state. Prediction state architectural components include a single D-type flip-flop (*DFF*) to store the state's predicted value, a two input multiplexor to swap the predicted value (effectively implementing a swap with the shadow state), and several gate level components. Prediction state inputs are similar to those used for 2-bit saturating counters, which are initialize (*IN*), prediction input pattern (*PIP*), enable (*Z*), and the clock (*CLK*) signal. Prediction states have a single output, which is the predicted value. The outputs of all prediction states are connected to a common output (*Prediction Output Value*) using tri-state buffers. The shift register is composed of *N* DFFs, whose outputs *Q* (also denoted as Z) are connected to the adjacent DFFs inputs *D* and selectively enable the prediction states. The shift register is clocked using the *BRANCH* signal, which is asserted each time the branch associated with this predictor is fetched.

At system startup, *IN* is asserted to reset the system. *IN* is connected to each DFF's reset (*RES*) port, effectively setting all register values to 0, except for the last DFF in the shift register. *IN* is connected to the set (*SET*) port of this DFF in order to set this DFF's value to 1. The shift register is responsible for selectively enabling a single prediction state, thus only one bit in the shift register should ever have a value of 1. Each time the *BRANCH* signal is asserted, the shift register updates its values, which enables the next sequential prediction state via the *Z* signal.

SDFSM prediction states consist of two operational phases: the *predict* operation and *correct* operation. The *predict* operation provides the branch prediction value while the *correct* operation swaps the branch prediction value with the shadow state value if the branch is mispredicted. During the predict operation, the enabled prediction state's output drives the *Prediction Output Value* using *Z*'s assertion to enable the tri-state buffer. During this time, the *PIP* input value should correspond to the *Prediction Output Value* (not shown in Figure 3) so that the DFF value does not change.

Parameter	Configuration
BTB, assoc, cache line size	128KB, 4-way, 32B
L2 unified size, assoc, cache line size	256KB, 4-way, 64B
L1 data size, assoc, cache line size	8KB, 4-way, 32B
L1 instruction size, assoc, cache line size	8KB, 4-way, 32B
Branch predictor techniques	Bimodal
Reorder buffer size	512
L3 unified size, assoc, cache line size	4 MB, 2-way, 64B
Pipeline depth	40

Table 2: Architectural parameters

If a branch is mispredicted, the *PIP* value will change to the branch outcome value and the prediction state enters the *correct* operational phase. During this phase, simple logic gates controlling the multiplexor's inputs and select line swap the DFF's stored value with the shadow value. Thus, in order to swap the DFF's stored value, the *PIP* must be different than the currently stored value *and Z* must be asserted.

The SDFSM has been architecturally designed to complete in one fast clock cycle. Assuming the DFFs are constructed using two levels of 3-input NAND gates and the multiplexor is constructed using standard two level logic gates, the longest register-register delay is seven gates (since DFF updating for the shift registers and prediction states is mutually exclusive, no phase flows through both DFFs). This situation occurs during the correct operational phase.

Table 1 depicts hardware area estimates in number of hardware components based on the number of prediction states *N*, where total hardware area grows at a rate of *O(N)*. To minimize the output steering logic, prediction state outputs share a common output wire using tri-state buffers. In addition, to minimize active power, the DFFs in each prediction state are only activated on a misprediction. Overall, the SDFSM architecture is highly cost-effective in terms of performance, area, and power.

3 Simulation Methodology and Evaluation Metrics

In order to perform an in depth analysis of the SDFSM, we exhaustively simulated the SPEC2000 benchmark suite [16] (we simulated each application in its entirety for all provided input stimuli) using the SimpleScalar PISA processor simulator version 4 [15]. We modified sim-bpred to implement the SDFSM and simulated the SDFSM with 2, 3, 4, 6, 8, 10, and 12 states. Our comparison framework focused on comparing the SDFSM to a popular branch prediction technique (bimodal) using 2-bit saturating counters with branch prediction table sizes ranging from 256k- to 16k-entries. We compare with the bimodal predictor because the bimodal predictor is a branch predictor cornerstone and allows us to establish the fundamental contribution of our SDFSM. Table 2 summarizes the base system's architectural parameters, which represent common modern system parameters, yet are conservative with respect to future technologies.

Each branch prediction table entry contains an FSM,

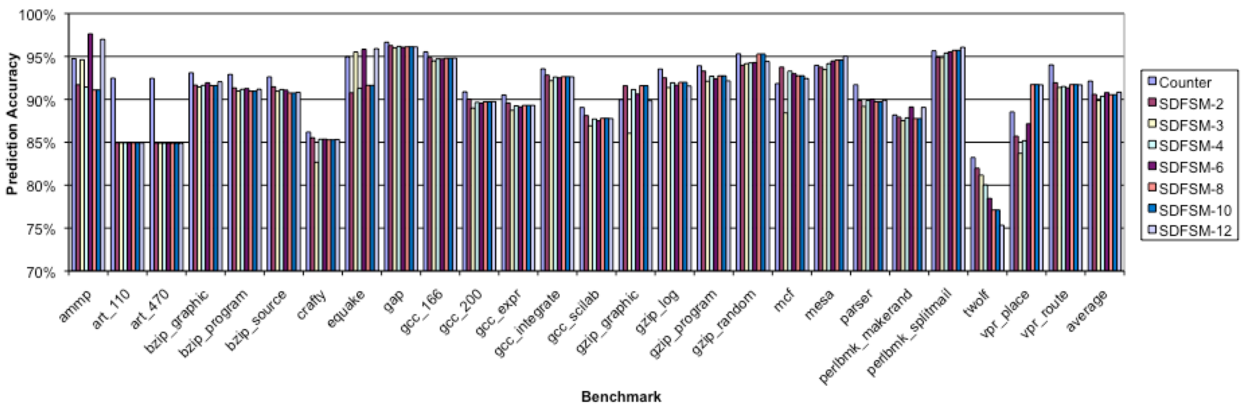


Figure 4: Prediction accuracy for each benchmark using a 4k-entry BHT for the bimodal branch predictor using a 2-bit saturating counter (counter) and SDFSMs with 2, 3, 4, 6, 8, 10, and 12 states (SDFSM-*X*)

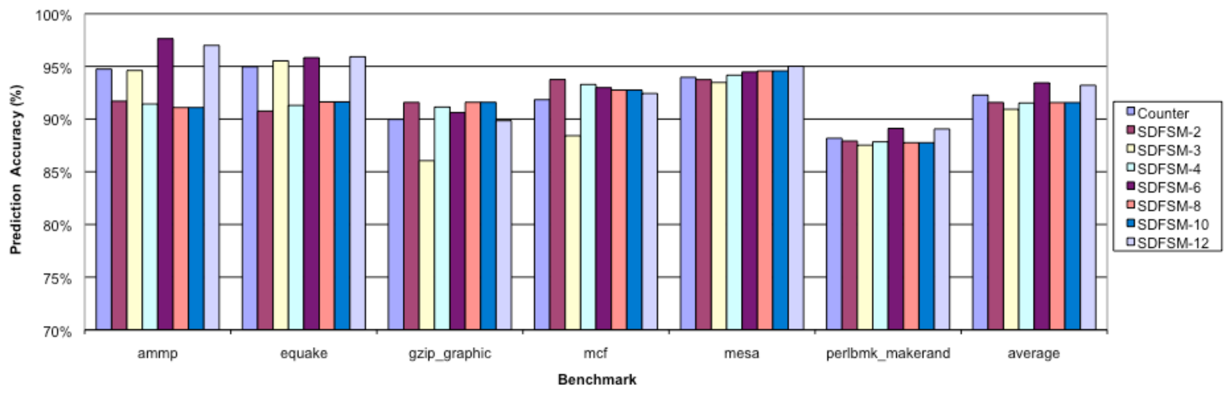


Figure 5: Prediction accuracy for the six advantageous benchmarks using a 4k-entry BHT for the bimodal branch predictor using a 2-bit saturating counter (counter) and SDFSMs with 2, 3, 4, 6, 8, 10, and 12 states (SDFSM-*X*)

which can be either the SDFSM or a 2-bit saturating counter. Hence, the predictor storage budget (PSB) in bits is determined by:

$$\mathrm{PSB} = 2^N \times \lceil \log_2(\text{number of States FSM}) \rceil$$

where N is the number of index bits used for the branch prediction table. In the conventional bimodal branch predictor, the low-order J bits of the branch address index into a branch history table (BHT) of size 2^J entries. The BHT entries can either be 2-bit saturating counters or can be replaced with SDFSMs of any size. Since it is difficult to precisely compare predictors with exactly the same hardware budgets, we compare predictors based on number of table entries, which provides a fair performance comparison because these tables account for the majority of the hardware budget.

Cumulative prediction rate accuracies are computed and analyzed using the arithmetic mean for averaging prediction rates, over all benchmarks, based on predictor storage budget. In addition, individual branch prediction accuracies for every benchmark and every branch prediction technique studied were measured for increasing hardware budgets, reflecting branch predictor sizes available in commercial microprocessors.

Improving processor performance, measured in number of instructions executed per cycle (IPC), is considered the key motivation for combining improved branch prediction accuracy with low latency branch prediction. High prediction latency nullifies any prediction accuracy advantages due to decreased IPC. For a 2-bit saturating counter, since each up-down counter only requires 2 bits to record the branch behavior, the technique requires simple hardware and little storage space. In addition, the 2-bit saturating counter's inherent simplicity results in simple single-cycle prediction computation, thus guaranteeing low prediction latency. In contrast, perceptron-based predictors require comparatively complicated computation using adder components. The prediction latency of the original perceptron predictor was more than 4 cycles [13], which required heavy pipelining to hide such latencies. This pipelining led to problems similar as those encountered when designing modern hyperpipelined execution cores [12]. Thus, since the SDFSM has the same access delay (single-cycle) as the 2-bit saturating counter, the key evaluation metric is SDFSM prediction accuracy compared to 2-bit saturating counters with a fixed hardware budget.

4 Experimental Results

Figure 4 shows the prediction accuracy for all benchmarks for the bimodal branch predictor with a 2-

bit saturating counter (counter) and SDFSMs with 2, 3, 4, 6, 8, 10, and 12 states (SDFSM-*X*) using a 4k-entry BHT. On average over all benchmarks, the 2-bit saturating counter outperforms all SDFSMs. However, we reiterate that branch predictors behave differently for all applications, and there is no one branch predictor that outperforms all other branch predictors for all applications.

Figure 5 subsets the results and depicts the six applications where the SDFSM shows improved prediction accuracy over the 2-bit saturating counter. On average, the 6-state SDFSM provides the largest prediction accuracy improvements with an average misprediction rate decrease of 18.3%, with individual decreases ranging from 6.3% to 55%. Figure 5 also reveals that for each benchmark, the optimal sized SDFSM is quite different. The optimal SDFSM sizes for *ammp*, *equake*, *gzip_graphic*, *mcf*, *mesa*, and *perlbmk_makerand* are the 6-state, 12-state, 8-state, 2-state, 12-state, and 6-state SDFSMs, respectively.

Figure 6 (a) depicts the arithmetic mean of the prediction accuracy for all benchmarks for the bimodal branch predictor with a 2-bit saturating counter (counter) and SDFSMs with 2, 3, 4, 6, 8, 10, and 12 states (SDFSM-*X*) for BHT sizes ranging from 256 to 128k-entries. The prediction accuracy increases as BHT size increases and saturates asymptotically. On average, the 2-bit saturating counter still outperforms all SDFSMs, with the 2-bit predictors prediction accuracy saturating at 92.3% and the next accurate predictor (6-state SDFSM) saturating at 91%. On average, the 2-bit saturating counter with 16k-entries (a practical hardware budget) provides 1.7% more accuracy than the next most accurate predictor.

Figure 6 (b) subsets the results from Figure 6 (a) and averages the six applications where the SDFSM shows improved prediction accuracy over the 2-bit saturating counter. For these benchmarks, the 6-state SDFSM is 1.2% more accurate than the 2-bit saturating counter, saturating asymptotically at 93.5%. This figure also shows that both the 6- and 12-state SDFSMs outperform the 2-bit saturating counter.

Overall, results reveal that our SDFSM has the potential to further enhance the accuracy of 2-bit saturating counters. Since literature shows that the most advanced branch prediction methods adopt neural or saturating elements, the SDFM has the potential to improve on these methods as a replacement for the saturating elements. The SDFSM is intended to enhance branch prediction for certain applications that exhibit particular behaviors such as aliasing, damping, and other irregularities such as those found in artificial intelligence and gaming applications (see Section 5 for details).

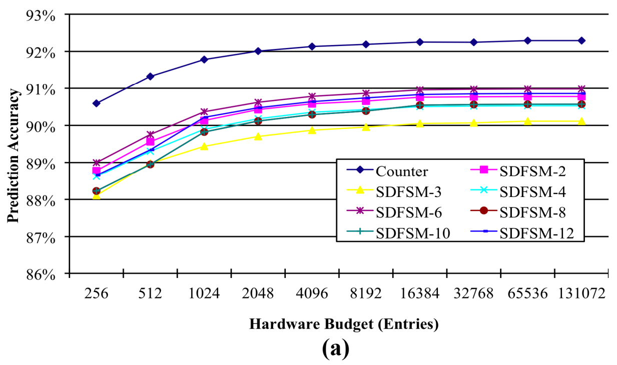
5 Comparison Analysis

In this section, we analyze the exhaustive results presented in Section 4 and discuss comparative advantages and disadvantages of the 2-bit and SDFSM branch predictors considering aliasing interference, damping, adaptability, training time, and latency.

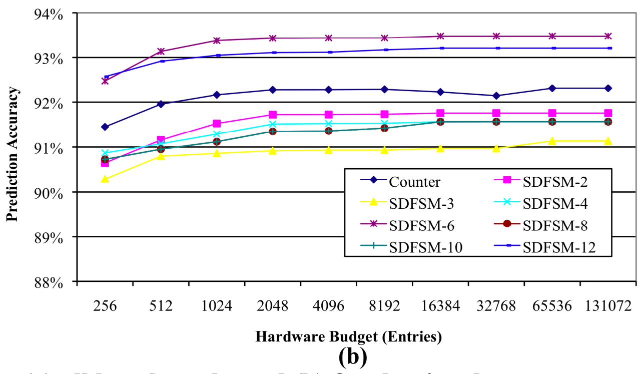
5.1 Aliasing Interference and Damping

Since the BHT size is generally much less than the total number of branches in an application, the bimodal branch predictor uses the low-order J bits to index into the BHT. Therefore, if two conditional branches have the same low-order J bits, their branch streams will be intermingled and sent to the same predictor. We define this situation as *aliasing interference*. Due to aliasing interference, and because we use the bimodal branch predictor, both the 2-bit saturating counter and our SDFSM-based predictor generally result in lower prediction accuracy in the presence of significant aliasing interference. Aliasing interference can be alleviated through two methods. Simply increasing the BHT size can significantly reduce aliasing interference. Additionally, using other branch prediction techniques such as per-address branch predictors (PAs) can reduce aliasing interference by using a two level indexing method [14]. The first level is indexed using a subset of H bits of the branch address to index into a pattern history table of size 2^H, which stores the unique local branch history pattern of that branch. This pattern is then used to index into the second level, which contains either global pattern histories (PAg) or per-address pattern histories (PAp) [3].

In general, aliasing interference does not directly imply prediction accuracy penalties. For example, if two branches alias to the same BHT entry but their executions are mutually exclusive, (the first branch



(a)



(b)

Figure 6: Arithmetic mean of the prediction accuracy for (a) all benchmarks and (b) for the six advantageous benchmarks for the bimodal branch predictor with a 2-bit saturating counter (Counter) and SDFSMs with 2, 3, 4, 6, 8, 10, and 12 states (SDFSM-*X*) for BHT sizes ranging from 256 to 128k entries

executes 1000 times followed by 1000 executions of the aliasing interference is negligible. However, if two branch executions are not mutually exclusive (the worst case being that the two branches alternate executions), then aliasing interference may lead to a significant decrease in prediction accuracy. To analyze the effects of aliasing interference in the case of two interfering branches, we define the most frequently executing branch as the *majority* branch and the least frequently executing branch as the *minority* branch. We further define a *majority run* as consecutive majority branch executions with no intervening minority branch executions. *Minority runs* are similarly defined.

Smith [1] observed that 2-bit saturating counters implicitly provided an appropriate amount of damping (or hysteresis) which alleviated some of the aliasing interference. The damping mechanism in 2-bit saturating counters requires two consecutive mispredictions before the prediction value changes, thus ignoring minority runs of length one. Damping trades off adaptability for vulnerability to short minority runs. In addition, damping also allows loop branches to incur just one misprediction per loop iteration, instead of two mispredictions (one on loop exit and one on loop entry).

On the other hand, the SDFSM's implicit damping mechanism is quite different than the 2-bit saturating counter. The SDFSM simply learns the branching pattern that maps to a particular BHT entry. Therefore, as long as the combined patterns of the interfering branches produce a learnable pattern, the SDFSM will learn that pattern. However, since these combined patterns are likely longer than individual branching patterns, this implies that SDFSMs with more states provide increased damping. The SDFSM predictor actually provides high/perfect prediction accuracy for applications with short minority runs as well as long minority/majority runs, by minimizing or even eliminating aliasing interference. On the contrary, in the presence of aliasing interference, damping in saturating counters only works well for long minority runs.

Literature shows that the bimodal predictor is widely known to have a significant amount of aliasing interference even as the hardware budget increases [2][4]. In our experiments, since both the 2-bit saturating counter and the SDFSMs use a bimodal predictor, large amounts of aliasing interference will favor the counter-

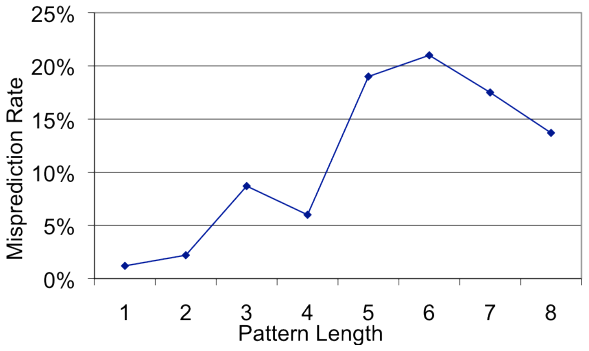


Figure 7: Bimodal predictor misprediction rates for various pattern lengths.

second branch) the prediction accuracy lost due to based predictor since the counter based predictor can better tolerate aliasing interference. Figure 4 shows that on average the 2-bit saturating counter can reduce the misprediction rate by 14.3% over the best SDFSM predictor (SDFSM-6). On the other hand, for the six benchmarks where SDFSMs are advantageous, short minority runs (which are considered a limitation of counter-based predictors) favor the SDFSM predictor. For these six benchmarks, Figure 5 shows that the SDFSM can decrease misprediction rates by 18.3% on average.

5.2 Recurring Patterns

Researchers have shown that aliasing in the pattern history tables can significantly degrade the performance of bimodal branch predictors. [3][4][21][23] showed that a repeating pattern of length one (i.e., "1111…1" or "0000…0") was detected for approximately 50% of the branches, indicating that a significant amount of branch inference may occur if the PHTs are updated for these branches. For these situations, a simple predictor such as a bimodal predictor would typically outperform the SDFSM predictor, which would incur every interference update.

In addition, research showed bimodal predictors could accurately predict branches with short repeating patterns, while branches with a repeating pattern of length six tended to have higher mispredication rates [21][22][23], as is show in Figure 7 from [23]. Since Section 4 revealed that the 6-state SDFSM was the best performing number of states on average, the 6-steate may provide improved performance for these branching patterns of length 6. In addition, our results demonstrated that SDFSMs with a smaller number of states suffered less branch interference penalty as compared to SDFMs with a larger number of states, which could explain why the 6-state SDFSM outperformed the 12-state SDFSM (or for any SDFSM with a multiple of 6 states).

5.3 Adaptability and Training Time

Branches typically exhibit high biasing (usually 70% [4]) towards one outcome (taken or not taken). This bell distribution (bell peaks at 70%) is key to a counter-based predictor's high prediction accuracy and explains why the 2-bit saturating counter outperforms the SDFSM for the majority of the benchmarks. To provide better prediction accuracy for low biasing applications, previous work shows [3][5] that applications with branches that show low biasing require dynamic adaptability in order to achieve high prediction accuracies. This dynamic adaptability enables the predictor to specialize itself to a branch's biasing during application execution. Dynamic adaptability provides the added benefits of not requiring any static profiling or branch predictor training during system/application design time. The 2-bit saturating counter lacks dynamic adaptability. On the other hand, our *N*-state SDFSM-based predictor can dynamically adapt to any branch pattern of length equal to (or a divisor of) *N*. The larger

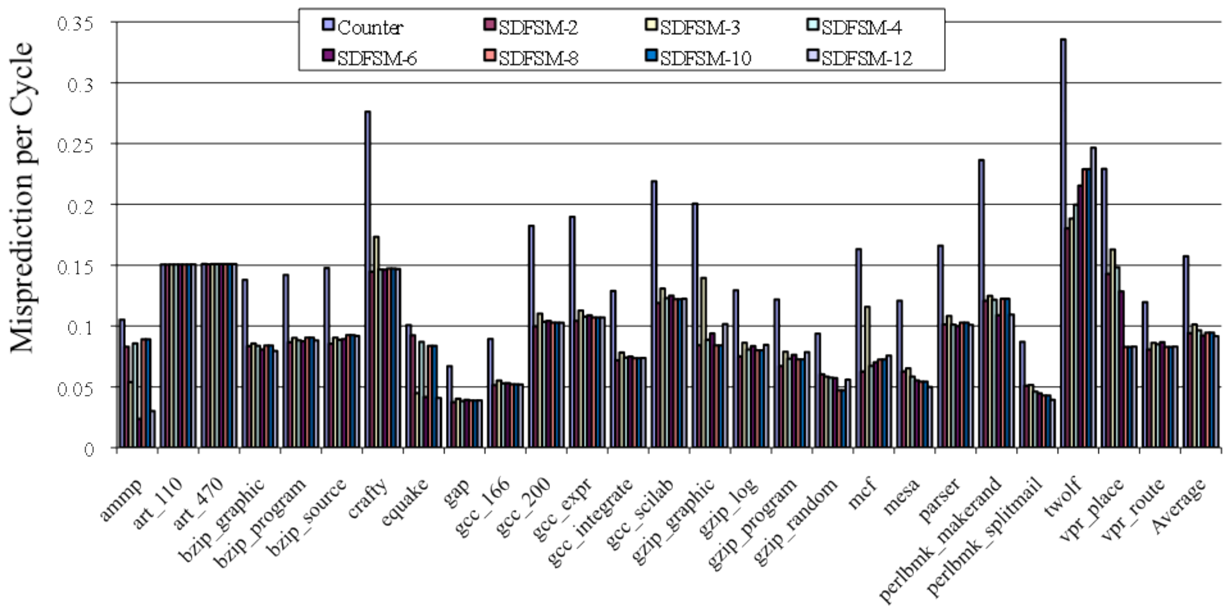


Figure 8: Mispredictions per cycle per benchmark with a hardware budget of 4KB.

the number of states, the more flexibility the SDFSM has for adapting to different pattern lengths.

However, SDFSMs with a large number of states can negatively impact the prediction accuracy due to longer training times. Figure 5 exemplifies this impact with the 6- and 12-state SDFSMs. *Ammp*, *gzip_graphic*, *mcf*, and *perlbmk_makerand* show increased prediction accuracy for a 6-state SDFSM even though the 12-state SDFSM captures the same branching pattern. On the other hand, *equake* and *mesa* show decreased prediction accuracy for the 6-state SDFSM because these benchmarks likely have longer branch patterns, thus requiring more SDFSM states. On average, the 6- and 12-state SDFSMs decrease misprediction rates by 18.3% and 15.4% compared to the 2-bit saturating counter, respectively. The 6-state SDFSM decreases misprediction rates by 4% compared to the 2-bit saturating counter. This overhead is due to the 12-state SDFSM's increased training time. Similar trends are evident when comparing 2- and 6-state SDFSMs, as well as any other SDFSM with common divisors.

5.4 Latency

Few hardware resources are required to implement both the 2-bit saturating counter and the SDFSM predictors and thus these techniques require only modest storage space. In addition, this inherent simplicity results in simple predictions and computations, which guarantees low prediction latency (a critical component for high performance in processors). The SDFSM-based predictor requires only a single cycle for training and prediction, while 2-bit saturating counter-based predictors require two cycles for training and predicting. Thus, the overall prediction latency of the SDFSM-based predictor is 50% less than that of the 2-bit saturating counter-based predictor, resulting in a higher instruction-per-cycle (IPC).

6 Performance Evaluation

Figure 6 (a) showed that the counter-based predictor was more accurate on average than the SDFSM with respect to the arithmetic mean. However, the counter-based predictor's misprediction latency cycles is twice that of the SDFSM, as was described in Section 5.4. The additional misprediction cycle adversely affects overall processor performance due to stalls while waiting for the training and subsequent prediction. Therefore, in order to more fairly compare complete predictor performance, we must consider the mispenalty latency in conjunction with the misprediction rate.

We evaluate the SDFSM and counter-based bimodal type predictors with respect to the misprediction per cycle (MPC) and the prediction accuracy rates (PAs) as determined by simulation. In order to provide an analysis that is independent of the processor clock speed, the misprediction rate is normally measured in cycles rather than in seconds, such that:

$$\mathrm{MPC}_{\mathrm{counter}} = [100\% - \mathrm{PA}_{\mathrm{counter}}] \times 2\ \mathrm{cycles}$$

and:

$$\mathrm{MPC}_{\mathrm{SDFSM}} = [100\% - \mathrm{PA}_{\mathrm{SDFSM}}] \times 1\ \mathrm{cycles}$$

Figure 8 shows the MPCs with respect to hardware budget in number of entries and Figure 9 subsets these results as in Figure 6 (a) (i.e., those where the SDFSM showed improvement over the counter-based predictor with respect to misprediction rates), Similarly to the misprediction rates for these subsetted benchmarks, the MPCs for all SDFSMs improves with respect to counter predictor, with an average overall performance increase of 37%. However, on average over all benchmarks the counter-based predictor still had the lowest misprediction rate.

Branch predictor performance can also be evaluated using the misprediction speedup, as derived in [17], such that:

$$\text{Speedup} = \frac{\text{MPC}_{\text{counter}}}{\text{MPC}_{\text{SDFSM}}}$$

Figure 10 shows the misprediction speedup verses hardware budget in number of entries for various SDFSM sizes compared to the counter-based predictor. These speedups are in line with speedups obtained for other recent innovations in branch predictors [18]-[20].

7 Conclusion and Future Work

This paper proposes the shadow dynamic finite state machine (SDFSM), a new branch predictor where the FSM states are dynamically trained during rum-time to learn unique branch pattern behaviors. Whereas the SDFSM can be generalized to any arbitrary number of states, we explored several SDFSM sizes and compared extensive simulation results on the SPEC2000 benchmark suite with 2-bit saturating counters using a conventional bimodal-based branch predictor. Results revealed that the SDFSM decreases average misprediction rate for six benchmarks, which have irregular branching tendencies (i.e. those seen in artificial intelligence and gaming applications). Furthermore, in the situations where the SDFSM was slightly less accurate than the 2-bit predictor, this reduced accuracy was due to the nature of the bimodal predictor architecture (and not a failure of the SDFSM), which inhibits a large percentage of aliasing phenomena that severely affects the performance of our SDFSM automaton on prediction accuracy. The SDFSM will likely show marked improvements when coupled with predictors that are less affected by aliasing such as PAs and GAs.

In addition, the SDFSM uses a simple hardware structure, which provides single cycle training and prediction latency; in contrast, the 2-bit counter predicts and corrects in two cycles. This single cycle advantage for the SDFSM offsets the accuracy advantage of the 2-bit counter by trading off performance with respect to the *instructions-per-cycle* (IPC) rate.

Finally, we explored and analyzed the number of SDFSM states in the scope of adaptability, training, damping, and aliasing in order to determine their affect on prediction accuracy. Results show that a 6-state SDFSM is a good average configuration for optimal length for bimodal predictor topology. Thus, our results encourage researchers to explore the SDFSM combined with more advanced predictor methods, thus improving the accuracy of those predictors.

Our future work is motivated by the per-application variation in optimal SDFSM size as shown in Figure 5. Consequently, choosing the *best* number of states is a key design decision since the SDFSM structure does not dynamically alter its number of states based on pattern entries. Therefore, our future work includes architecting an adaptive SDFSM capable of dynamically altering its number of states based on actual branch pattern length.

8 References

[1] J. Smith, "A Study of Branch Prediction Strategies," International Symposium on Computer architecture (ISCA), pp. 135-148, June 1981.

[2] C. –C. Lee, C. C. Chen, and T. N. Mudge, "The Bi-mode Branch Predictor," International Symposium on Microarchitecture (MICRO 30), pp. 4-13, December 1997.

[3] T. Yeh and Y. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," International Symposium on Computer architecture (ISCA), pp. 257-266, June 1993.

[4] C. Young, N. Gloy, and M. D. Smith, "A Comparative Analysis of Schemes for Correlated Branch Prediction," International Symposium on Computer architecture (ISCA), pp. 276-286, July 1995.

[5] A. Seznec, "Analysis of the O-Geometric History Length Branch Predictor," International Symposium on Computer Architecture (ISCA), pp. 394-405, June 2005.

[6] P. Biggar, N. Nash, K. Williams and D. Gregg, "An Experimental Study of Sorting and Branch Prediction," Journal of Experimental Algorithmic (JEA), Volume 12, Article 1.8, June 2008.

[7] Y. Ma, Hongliang Gao, and Huiayang Zhou, "Using Indexing Functions to Reduce Conflict Aliasing in Branch Prediction Tables," IEEE

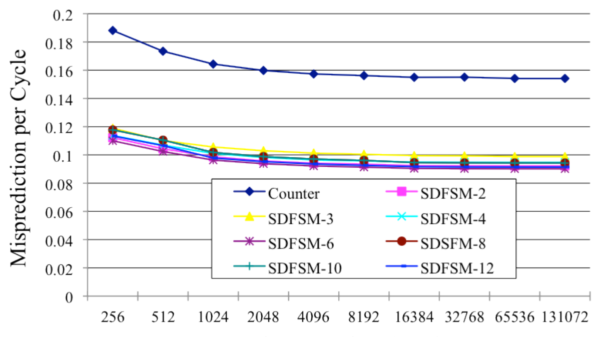


Figure 9: Average mispredictions per cycle verses hardware budget in number of entries

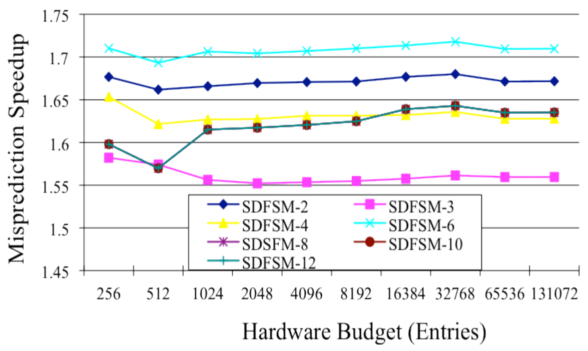


Figure 10: Misprediction speedup verses hardware budget in number of entries for various SDFSM state sizes compared to the counter-based predictor.

Transactions on Computers, Vol. 55, No. 8, pp. 1057-1061, August 2006.

[8] C. Y. Ho, K. F. Chong, C. H. Yau, and A. S. S. Fong, “A Study of Dynamic Branch Predictors: Counter versus Perceptron,” International Conference on Information Technology (ITNG’07), pp. 528-536, April 2007.

[9] R. Nair, “Optimal 2-bit Branch Predictors,” IEEE Transactions on Computers, Vol. 44 (5), pp. 698-702, Issue 5, May 1995.

[10] J. L. Hennessy and D. A. Patterson, Computer Architecture: A quantitative Approach, Morgan Kaufman Publishers, 3rd Edition, 2003.

[11] D. A. Jimenez and C. Lin, “Dynamic Branch Prediction with Perceptron,” International Symposium on High-Performance Computer Architecture, (HPCA), pp. 197-206, January 2001.

[12] A. S. Fong and C. Y. Ho, “Global/Local Hashed Perceptron Branch Prediction,” International Conference on Information Technology: New Generations ITNG '08, pp. 247-252, April 2008.

[13] D. A. Jimenez, “Improved Latency and Accuracy for Neural Branch Prediction,” Transactions on Computer Systems (TOCS), Volume 23 Issue 2, pp. 197-218, May 2005.

[14] K. C. Breen and D. G. Elliott, “Aliasing and Anti-Aliasing in Branch History Table Prediction,” Computer Architecture News, Vol. 31, No. 5, pp. 1-4, December 2003.

[15] T. Austin, D. Ernst, E. Larson, C. Weaver, R. N. Raj Desikan, J. Huh, B. Yoder, D. Burger, and S. Keckler, SimpleScalar Tutorial 2001 (for release 4.0).

[16] SPEC 2000, The SPEC 2000 Benchmark Report, Waterside Associates, Fremont, CA., January 1990.

[17] M. Burtscher and B. G. Zorn, ”Prediction Outcome History-based Confidence Estimation for Load Value Prediction,” Journal of Instruction-Level Parallelism, Vol 1, May 1999.

[18] T. H. Heil, Z. Smith, and J. E. Smith, “Improving Branch Predictors by Correlating on Data Values,” 32nd Annual international symposium on Microarchitecture (MICRO-32), pp. 28-37, Nov. 1999.

[19] R. Thomas, M. Franklin, C. Wilkerson, J. Stark, “Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Barnches from a large Global History,” 30th Annual International symposium on Computer Architecture, pp. 314-323, June 2003.

[20] R. Sendag, J. J. Yi, P. Chuang, and D. J. Lilja,”Low Power/Area Branch Prediction Using Complementary Branch Predictors,” IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), pp. 1-12, June 2008.

[21] P. Chang, M. Evers, and Y. N. Patt, “Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference,” Proceedings of the 1996 Conference on Parallel architecture and Compilation Techniques, PACT ’96, pp. 48-57, Oct. 1996.

[22] A. R. Talcott, M. Nemirovsky, and R. C. Wood, “The influence of branch prediction table interference on branch prediction scheme performance,” International Conference on Parallel Architectures and Compilation Techniques, pp. 89-98, June1995.

[23] J. Stark, M. Evers, and Y. N. Patt, “Variable Length Path Branch Prediction,” International Conference on Architectural Support for Programming Languages and Operating systems (ASPLOS), pp. 170-179, Dec 1998.