Comparison of Branch Prediction Techniques

Report compiled to fulfill the project requirements for CSE 240A

Amit Borase
Computer Science and Engineering Department
University of California, San Diego
A53095391
Email: aborase@eng.ucsd.edu

Owais Siddiqui
Computer Science and Engineering Department
University of California, San Diego
A53065443
Email: o1siddiq@eng.ucsd.edu

***Abstract*—Branch prediction techniques are an essential component of computer architectures. They help ensure that potential performance and power wastages are avoided. Instead of stalling when faced by a branch, instructions can be fetched speculatively and executed along a predicted path decided by the branch prediction scheme being utilized by the a pipelined processor. With increases in the number of instruction, the penalty of mispredictions are known to increase correspondingly. This report evaluates the impact of storage budget on the performance of branch predictors. The branch predictors considered for assessment are - 2-Level local predictor, Alpha 21264-like predictor, Perceptron predictor and G-Share predictor. These branch predictor's were analyzed using different budget sizes and compared to determine the mispredict rate. The evaluation revealed that the performance of these predictor improve with the increase in the available budget sizes. It also further revealed that the perceptron predictor provides misprediction rates (per 1000 instructions) as low as 1.8, resulting in 98% prediction accuracy, whereas the gshare predictor gives misprediction rates of as low as 3.5 and prediction accuracy of 96%. Additionally, Alpha and 2-Level Local Predictor were found to be lacking in performance compared to other two.**

I. Introduction

Branch predictors learn and store certain information that helps it determine the direction a branch would most likely choose before it is known. They improve the flow in the instruction pipeline and therefore increase the performance of pipelined architectures. This report is based on a study comparing the performance of four predictors across various budget sizes. The performance is analyzed on the following parameters - number of branches, number of taken branches, and mispredict rates.

The predictors, budget sizes and traces considered for the project are:

Branch Predictors	Budget Sizes	Trace Types
2-Level Local	8K + 64 bits	Floating Point
Alpha 21264 -like	16K + 128 bits	Integer
Perceptron	32K + 256 bits	Multimedia
G-Share	64K + 512 bits	Server
	128K + 1K bit	
	1M + 4K bits	

This paper describes and evaluates the four selected predictors and is organized as follows. Section 2 provides descriptions of each predictor followed by the observations we derived from the implementations, Section 3 details the experimentation we performed with the budget and how they were utilized to analyze prediction performance, and Section 4 provides the results of the analysis and details our reasoning for predictor behavious with different traces. Finally, In Section 5 we provide a conclusion ascertained from our study and the results obtained.

II. Branch Predictors

A. 2-Level Local Predictor

2-Level Local Predictors build upon the simple Bimodal Branch Predictor [5] by adding an additional table containing local branch history indexed by program counter. Bimodal branch predictor do not take into consideration the usually repetitive patterns followed by the branches. The local history table allows 2-level local predictor to improve predictions by taking local history of each branch into consideration. If there are more branches in the program, it may suffer from similar issues as the Bimodal Branch Predictor, which is contention of multiple branches for the same branch history entry, which further results in conflicting branch patterns for a saturating counter entry.

Owing to above mentioned drawbacks of 2-Level Local predictor, we expect that an increase in the budget size would substantially improve the performance of the 2-level local branch predictor as it will allow it to store more local history entries and saturating counters thereby reducing the contention between branches (with same PC-LSB) as well as between the history patterns. Additionally, we also found out that the varying the saturating counter sizes affected the prediction rates, our best results were obtained for the 3-bit saturating counters.

B. Alpha 21264-like Predictor

The Alpha 21264 branch predictor is based on the tournament prediction technique. It was designed exclusively for the Alpha 21264 microprocessors[3]. Alpha 21264 needed highly capable predictor to be able to take full advantage of its speculative execution capabilities and to avoid the

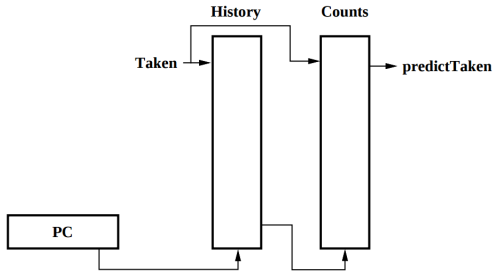

Fig. 1. Local History Predictor Structure

relatively expensive 7-cycle stall penalty for branch mispredict. It dynamically chooses between two set of predictors - one using the local history and another using the global history, to predict the future direction of a branch. Such predictors perform better when the branches exhibit both local correlation and global correlation [3]. In other words, it works well when the branches either follow a certain execution pattern or they simply depend on direction taken by preceding branches in execution.

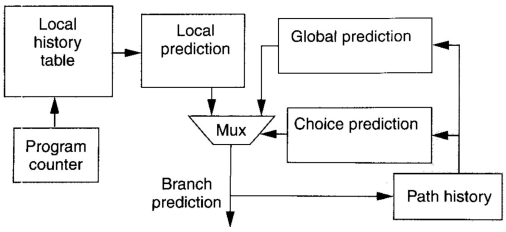

Fig. 2. Alpha 21264 Predictor Structure

Local history based predictor is a traditional 2-level local predictor which has been described earlier in this report. The global history based predictor simply employs a table of saturating counters to identify branch bias towards global branch history. Both these predictors are trained every time irrespective of which one got selected for prediction. A choice table is maintained to dynamically choose between these two predictors to make branch predictions, they are simple table of saturating counters indexed using the global history register. Saturating counters are adjusted based on prediction accuracy during the training phase. The upper half of values of the saturating counter results in selection of global predictor whereas the values in lower half of saturating counter results in selection of 2-level local predictor. Interestingly, we observed that altering this selection arrangement yields potentially worse results for given traces.

We expect that an increase in the budget size would improve the performance of the branch predictor as it has more data from the global history register to learn and improve predictions. Our experiments revealed that this trend was observed within the integer and the server traces. However, for the floating point and multimedia traces, we observed that the prediction performance increased and decreased unevenly as the budget size increased. We found unexpected results for the floating point and multimedia traces(only for 1M budget size).

C. Perceptron Predictor

Traditional predictors such as local history table based predictors, takes into consideration limited history of a branch due to hardware budget limitations, as a result it does not have as much accuracy as potentially possible. The neural method based Perceptron predictors address this drawback by making it possible to consider increased local history while making predictions. These predictors perform exceptionally well for the class of linearly separable branches.

Such branch predictor is based on the usage of perceptrons, which are a type of learning device that takes a set of input values and combines them with a set of weights (which are updated via learning) to produce an output value. Here, each weight represents the degree of correlation between the behavior of the past branch and the behavior of the branch being predicted. Positive weights represent positive correlation whereas negative ones represent negative correlation. Taken branch is represented by '+1' and not taken branch by '-1' values. The dot product of the weight set with branch history bits gives out an output. If the output obtained is positive, branch is predicted to be taken otherwise we predict not taken. The perceptrons are trained by an algorithm that increments the weights when the branch outcomes agrees with the weights correlation and decrements the weight otherwise. A bias weight as shown in Fig. 3 is also included to learn the bias of the branch independent of the branch history [2].

A perceptron is represented by a vector whose elements are weights. Here the weights are signed integers. The output is the dot product of the weights vector w_1, ... ,w_n and the input vector x_1, ... ,x_n. The output of the perceptron is computed as:

$$y = w_0 + \sum_{i=1}^{n} x_i \; w_i$$

Similarly, the weights are trained using the below algorithm [2]:

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
    for i := 0 to n do
        w_i := w_i + t x_i
    end for
end if
```

Fig. 3. Training ALgorithm

We expect that an increase in the budget size would improve the performance of the branch predictor as it has more data

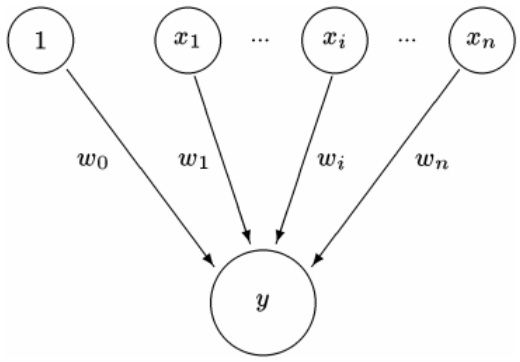

Fig. 4. Perceptron. The inputs x_1, ... ,x_n are propagated through the weighted connections by taking their respective products with the weights w_1, ... ,w_n. These products are summed, along with the bias weight w_0, to produce the output value y.

from the global history register to learn and improve predictions.

D. G-Share Predictor

The Bimodal predictors work well when each of the branch is highly biased in certain direction whereas the global predictors work well when the direction of a branch depends somewhat on the directions taken by the preceding branches in the execution. Either of these predictors do not perform well if the branches exhibit both of these tendencies. There is a scope for improvement if we can include both these branch tendencies when predicting a branch direction. G-share predictor achieves this by taking a exclusive OR of both the program counter of the branch and the global history to obtain the index into the saturating counter table. The exclusive OR obtained in this way has more information than either input component individually.

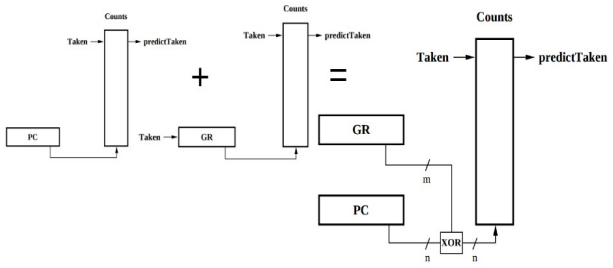

Fig. 5. G-Share Predictor Structure

III. Experimentation

Experimentation involved studying the performance of each of the 4 predictors against 6 types hardware budget as depicted in the introduction section. Additionally, we used 4 branch traces from different work-flows to analyze the prediction results.

The following tables present how the given budgets were utilized by each predictor:

A. 2-Level Local Predictor

There were 2 different decisions that we were required to make for budget allocation for 2-level local predictor. One, size of the saturating counter and second, the relative sizes of the local history table compared to branch predictor table. We found out after experimenting with various permutations of these parameters, that 3-bit saturating counter and the slightly bigger local history table than branch predictor table gives the best results. Below are the budget allocation scheme that performs better.

BSize: Budget Size
LHTS: Local History Table Size
BPTSize: Branch Prediction Table Size
NBLHT: Number of Bits per LHT entry
NBBPT: Number of Bits in BPT per counter

BSize	LHTS	BPTSize	NBLHT	NBBPT
8K	512	512	3	9 bits
16K	1024	1024	3	10 bits
32K	2048	2048	3	11 bits
64K	4096	4096	3	12 bits
128K	8192	8192	3	13 bits
1M	49152	65536	3	16 bits

B. Alpha 21264-like Predictor

For Alpha predictor, we split the given budget into 3 different parts, 1 part for each of 2 predictors and 3rd part for the choice prediction table. Majority of the budget (60%) was allocated to the 2-level local predictor (which was allocated in accordance with the allocations made for the same in stand-alone case). Remaining 40% of the budget was allocated to global predictor table (16%) and choice prediction table (24%). We found out that using 2-bit saturating counters and 3 bit saturating counters works best. (Hence the 16%-24% split). Below set of two table list out the allocation schemes for various budgets (1st table is for 2-level local predictor and 2nd table is for global predictor and choice table).

BSize: Budget Size
LHTS: Local History Table Size
BPTSize: Branch Prediction Table Size
NBLHT: Number of Bits per LHT entry
NBBPT: Number of Bits in BPT per counter

BSize	LHTS	BPTSize	NBLHT	NBBPT
8K	384	256	3	8
16K	768	512	3	9
32K	1536	1024	3	10
64K	3072	2048	3	11
128K	6144	4096	3	12
1M	32768	32768	3	15

BSize: Budget Size
GTS: Global Predictor Table Size
NGPT: Number of Bits per GPT Entry
CTS: Choice Prediction Table Size
NCPT: Number of Bits Per CPT Entry
GHRS: Global History Register Size

BSize	GTS	NGPT	CTS	NCPT	GHRS
8K	512	2	512	3	9
16K	1024	2	1024	3	10
32K	2048	2	2048	3	11
64K	4096	2	4096	3	12
128K	8192	2	8192	3	13
1M	65536	2	65536	3	16

C. Perceptron Predictor

In order to use the maximum possible global history to take full advantage of perceptron predictor, we focused on maximizing the weight count per perceptron. Furthermore, we found that the 8-bit weights gave more consistent results than others. We also played around with the θ and found out that the optimal value suggested for θ in [2] (which is θ = 1.93*weightcount + 14), works well for various weight counts per perceptron. Below table depicts the various budget type allocation for perceptron predictor.

BSize: Budget Size
NP: Number of Perceptrons
NWPP: Number of weights per perceptron
T: Threshold Value
BPW: Bits per Weight

BSize	NP	NWPP	T	BPW
8K	85	12	36	8 bits
16K	128	16	44	8 bits
32K	170	24	60	8 bits
64K	292	28	68	8 bits
128K	340	48	115	8 bits
1M	1024	128	270	8 bits

D. G-Share Predictor

G-Share Predictor maintains only one table and hence the budget allocation schemes is pretty consistent. We found out that the 2-bit saturating counters work better after trying various options. Below table depicts the allocation scheme for G-Share predictor.

BSize: Budget Size
NSC: Number of Saturating Counters
SCSize: Size of each Saturating Counter
GH: Global History

BSize	NSC	SCSize	GH
8K	4096	2 bits	12 bits
16K	8192	2 bits	13 bits
32K	16384	2 bits	14 bits
64K	32768	2 bits	15 bits
128K	65536	2 bits	16 bits
1M	524288	2 bits	19 bits

IV. Results

A. Traces

The section below details the results observed for various budget sizes for each trace and predictor combination. Note that, the when we talk about misprediction rates below, it signifies the number of mispredicts per 1000 instruction in trace.

1. Integer Trace

Integer trace contains 4184792 branch instances (for 424 unique branches). Our experiments showed high degree of uniformity in performance improvement for integer trace with the increasing budget sizes for all the four predictor types (shown in Fig. 6). We believe that even though number of unique branches is less, it requires higher set of tables (probably with sparse usage in reality) to uniquely distinguish one branch from another. The perceptron predictor specifically

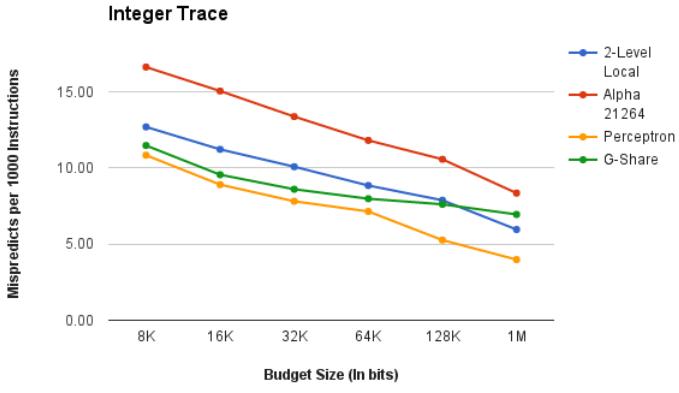

Fig. 6. The integer trace supports the pattern of improvement in performance with increase in budget size.

provides best in-class results across all budget sizes for this trace type and reaches mispredicts rate of as little as 3.9 (OR

accuracy as high as 97.2%) for 1M budget. G-Share predictor follows perceptron closely till 64K budget size but falls behind for higher budgets. 2-level local predictor performs slightly worse than the G-Share predictor. The surprising results for us was the consistently underperforming Alpha predictor. Based on this results we believe that the majority of branches in integer traces are highly biased in a particular direction. As a result, all the predictors are able to perform better.

2. Floating Point Trace

Floating point trace includes 2213673 branch instances(with 444 unique branches). Our experiments showed highly saturated results right from base budget size of 8K for all the predictors types except for the perceptron and G-Share predictor (shown in Fig. 7). Again Perceptron predictor gave best in-class results. G-Share also showed uniform improvement in prediction with increasing budgets but the improvement rates were not as noticeable as with the perceptron. We believe that the Floating Point involves considerable *linearly separable class of branches* and no other predictor than perceptron is able to distinguish it and take advantage of this inherent branch quality. Overall misprediction rates obtained for perceptron were as small as 1.7 (97.8% of accuracy), while the rest of the predictors hovered between 3.6 to 4.4 of misprediction rates.

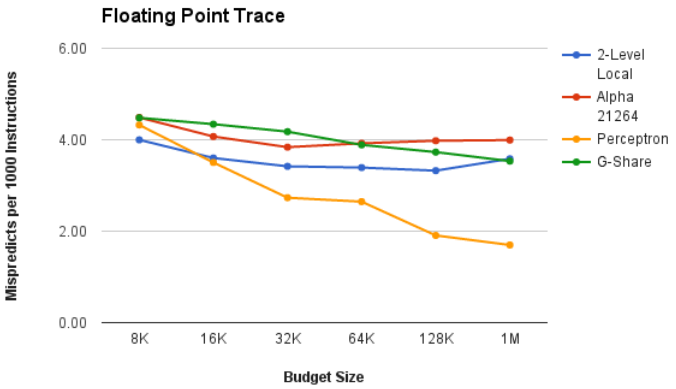

Fig. 7. The floating point trace shows a slight deviation from the pattern of improvement in performance with increase in budget size.

3. Multimedia Trace

Multimedia trace included 2229289 of branch instructions (with 460 unique ones). As shown in Fig. 8, our experiments showed fair amount of improvements for all the predictors with increasing budget availability, except at the 1M budget for both the 2-level local predictor and Alpha predictor. We speculate that at the 1M budget mark, the number of entries in both local history table and branch predictor tables of both stand-alone 2-level local predictor and the one in Alpha predictor suffer from increased deconcentration of local history information arising from vastly sparse tables. Furthermore, we believe that the branches in multimedia traces are more erratic in nature and don't really show a bias towards specific branch direction. This manifests into considerably declining improvements with increasing budgets for all predictors. Again the perceptron predictor tops the performance charts with misprediction rates as low as 7.5, followed by G-Share with misprediction rates 9.5.

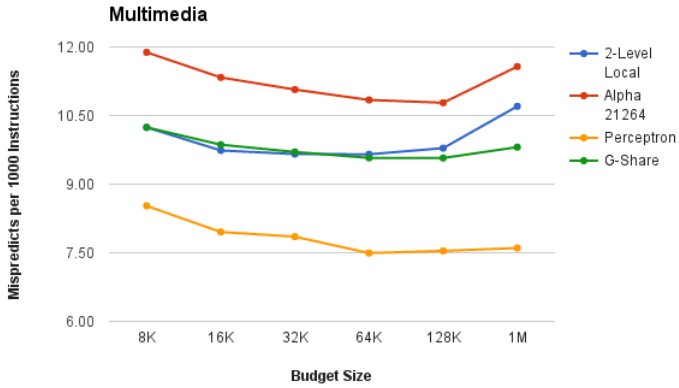

Fig. 8. The multimedia trace shows significant deviation from the pattern of improvement in performance with increase in budget size.

4. Server Trace

Server trace included 3660616 instances of branch instructions (with 10910 unique ones). As seen from Fig. 9, server traces follow distinctive uniform improvements with the budget sizes for all the predictors types studied. We believe this uniform increase can be somewhat attributed to the fact that are a lot more unique branches in server trace compared to all other traces. Thus, the increased table sizes arising from

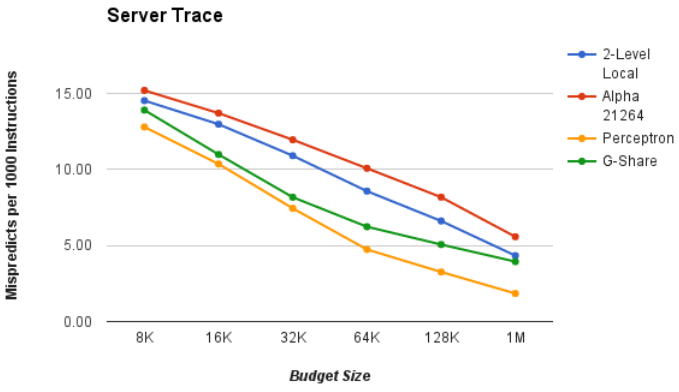

Fig. 9. The server trace supports the pattern of improvement in performance with increase in budget size.

increased budgets help in alleviating the contention for the local history entries as well as the saturating counter entries maintained in different predictors, thereby resulting in uniform performance increase with the increase in size of the various

tables. We also believe that the branches in server trace are either highly biased towards specific branch direction or they highly depend on the branch direction taken by preceding branches. These property allows predictors to efficiently learn the corresponding correlation between branches and use that to make future branch direction prediction. Keeping with the trends seen so far, perceptron predictor once again tops the class with misprediction rates as low as 1.8 (98.5% prediction accuracy).

B. Predictor Specific Observations

1. 2-Level Local Predictor

Our expectations for 2-level local predictor outlined in II.A were revealed to be true by our experiments for the integer and the server traces. Though, floating point and multimedia traces followed this trend but it wasn't as acute as the other two. We find the multimedia trace to be giving slightly erratic prediction rates, we suspect that this is because of the inherent unbiased nature of the branches involved in it.

2. Alpha 21264 Predictor

Since Alpha predictor is tournament based predictor involving both the 2-level local and global predictor, we expected it to perform at least better than 2-level local predictor. We believe that the, the additional choice logic needs to be optimized to be able to perform as good as 2-level local predictor. Additionally interestingly, we observed that the altering of the selection arrangement of choice predictor yields potentially worse results for given traces, best possible results were obtained when positive choices resulted in selection of global predictor and negative one in 2-level local predictor.

3. Perceptron Predictor

Our experiments revealed that the perceptron predictor showed uniform increase in performance with increasing budget sizes for all traces, except for multimedia trace. For the multimedia trace, we found unexpected results for the at the budget sizes of 128K and 1M. We believe that for multimedia traces, at this budgets the global history being considered is so large that the correlation being trained between branch weights gets adulterated by the distinctly independent branches in the global history.

One of the key aspects we observed for the perceptron predictor is its computationally expensive nature. We believe this to be the main inhibiting factor for them being relatively less practical solution for branch prediction even though they deliver top of the class performance.

4. G-Share Predictor

Despite its simplicity G-Share predictor consistently performs much better than other traditional prediction techniques. It only lacks slightly when compared to the perceptron predictor. We attribute this to G-Share inherent ability to combine both, inclination of branch in some direction and branches correlation with the outcomes of the preceding branches in execution. We ascertain that the simplicity and high performance are the 2 factors responsible for G-Share popularity in processor industry.

V. Conclusion

In this project, we studied and implemented four different branch predictors - namely 2-Level Local predictor, Alpha 21264 like predictor, Perceptron predictor and G-share predictor. We conducted experiments for these predictors for four types of branch traces - Integer, Floating Point, Multimedia and Server, with different sets of budget allocations. Our results show that with increasing budgets, performance of the all these predictors can be improved, this results from the fact that with larger budgets more state can be maintained to efficiently learn about the branch correlations and predict their future directions. We also found out that the perceptron predictor consistently performs better than all other predictors, it achieves misprediction rates as low as 2 (98% prediction accuracy). It is followed closely by G-Share predictor, which achieves misprediction rates as high as 3.5 (96% of prediction accuracy). The 2-level local predictor and Alpha like predictor do show performance improvements but they are not as high as other two.

Acknowledgment

We acknowledge the instructor team of previous offerings of CSE240A course for designing this project. We are also grateful to the teaching staff and Prof. Orailoglu for their constant help and guidance.

References

[1] Avinoam N Eden and Trevor Mudge. The YAGS branch prediction scheme. In: Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture. IEEE Computer Society Press. 1998, pp. 6977.

[2] Daniel A Jimenez and Calvin Lin. Neural methods for dynamic branch prediction. In: ACM Transactions on Computer Systems (TOCS) 20.4 (2002), pp. 369397.

[3] Richard E Kessler. The Alpha 21264 Microprocessor. In: IEEE micro 19.2 (1999), pp. 2436.

[4] Chih-Chieh Lee, I-CK Chen, and Trevor N Mudge. The bi-mode branch predictor. In: Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on. IEEE. 1997, pp. 413.

[5] Scott McFarling. Combining branch predictors. Tech. rep. Technical Report TN-36, Digital Western Research Laboratory, 1993.

[6] Andre Seznec et al. Design tradeoffs for the Alpha EV8 conditional branch predictor. In: Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on. IEEE. 2002, pp. 295306.