Branch Predictor Project

Danbing Zhu
Computer Science and Engineering
UC San Diego
Email: daz054@ucsd.edu

Kulshreshth Dhiman
Computer Science and Engineering
UC San Diego
Email: kdhiman@ucsd.edu

***Abstract*—Branch predictor plays a very important role in modern micro-processors especially with depth pipeline. In this report we evaluate the impact of using different storage budgets for 2-level ocal predictor, alpha, Perceptron and G-Share predcitor. Instruction level pipeline is a very important aspect to improve the performance of modern microprocessors. However, the existence of branch instructions causes considerable amount of stalls between instructions and consequently limits the improvement of the throughput of the pipeline. Branch prediction means to make decision for the branches even before their branch conditions are evaluated and thus eliminates the stalls. However, we can see that if the prediction is wrong, the penalty is also huge. So we need to improve the accuracy of the predictors and thus achieve a better performance of the whole pipeline. In this report, we are going to discuss four kinds of predictors. We will evaluate their performance as well as analyze the reasons behind their behaviors.**

I. Introduction

In this project, we use C++ program to simulate the behaviors of four branch predictors. Namely, 2-level local predictor, alpha 21264-like predictor, perceptron predictor and G-share predictor. We evaluate the impact of using different storage budgets for each predictor type, as well as analyzing which (and why) predictors work better for each budget size and trace type.

II. Conceptual Description of Predictors

A. 2-Level Local Predictor

Two level local predictor is based on the idea that miss-rate can be reduced if we take a local history of each branch in consideration. There are two level in this predictor. At the first level, there is Per Branch History Table (PBHT) which has h bit shift registers representing branch results of the most recent h branches.The h bits index into Global Pattern History Table (GPHT) of size 2^h. Each entry in GPHT is a s bits saturating counter which represents branch results of last h branches. The 2-level local predictor takes a least significant bits of each branch address and use them to index PBHT. These h bits index into GPHT provide a saturating counter. We compare this saturating counter to final prediction. We update the contents of history register and saturating counter when the branch is evaluated. Fig 1 illustrates how 2-level local predictor works and Equation (1) shows how to calculate its cost.

$$COST = 2^a \cdot h + 2^h \cdot s \tag{1}$$

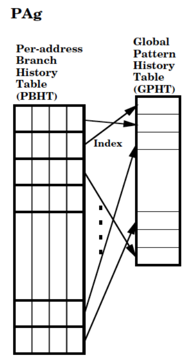
PAg
Per-address Branch History Table (PBHT)
Global Pattern History Table (GPHT)
Index

Fig. 1: Simulation results for the 2-level local predictor using PAg[4]

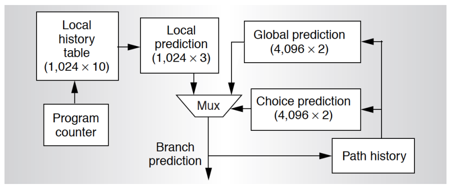
Local history table (1,024 × 10)
Local prediction (1,024 × 3)
Global prediction (4,096 × 2)
Program counter
Mux
Choice prediction (4,096 × 2)
Branch prediction
Path history

Fig. 2: Alpha 21264-like predictor[2]

B. Alpha 21264-like Predictor

Some branches are better predicted by local history while some can be better predicted by global history. Alpha 21264-like predictor is a tournament predictor which accounts both local and global history a branch. For the local part, we have 2-level predictor with local history table and local predictor table. For the global part, we have a global predictor. A choice predictor decides which of these predictors (local or global) is best suited for that branch. a least significant bits of program counter index to local history table whose width is h_l, and these local histories further index to local predictor table whose width is s_l. Global predictor has a table of s_g bits saturating counter index by the h_g bits global history register. The global history register also index to the choice predictor table, where the selected s_c bits saturating counter will decide if we are going to use global or local prediction.Fig 2 illustrates how alpha 21264-like predictor works and Equation (2) shows how to calculate its cost.

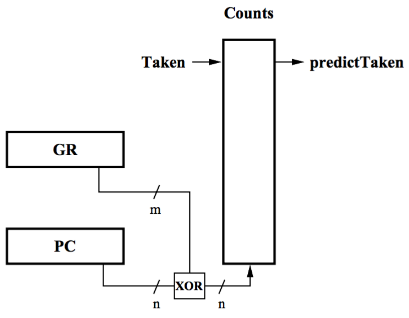

Fig. 3: G-share predictor[7]

$$COST = 2^a \cdot h_l + 2^{h_l} \cdot s_l + h_g + 2^{h_g} \cdot s_g + 2^{h_g} \cdot s_c \quad (2)$$

C. Perceptron Predictor

This predictor utilizes neural methods to predict a branch. Saturating counters in a 2-level predictor are replaced by perceptron. Each perceptron takes the weighted sum of last h branches outcomes. Each weight is s-bit integer and PBHT stores outcomes of most recent h outcomes of a branch. Each perception has $h + 1$ weights each s bits. Output function is calculated as follows

$$y = w_0 + \sum_{i=1}^{h} w_i * hist_i$$

$hist_i$ represents is i-th history bit(+1 bit for 0 and -1 for 0 bit). This way perception is able to learn the correlation between a particular branch outcome and current branch behavior. If there is a strong correlation, then weights for that branch outcome increases, and decrease otherwise. We have a table of history register (h bits) indexed by last a bits of the branch address. We have a table of N perceptions, which are indexed by the hash of these h bit history. A simple hash function would be modulo. We update the weights by 1 if history bit and outcome matches and decrease otherwise. This update is done only when there is misprediction or y is less than the threshold(floor(h*1.93+14)). Equation (3) shows how to calculate the cost of perceptron predictor.

$$COST = 2^a \cdot h + N \cdot ((1 + h) \cdot s) \quad (3)$$

D. G-Share Predictor

This prediction scheme is based on the idea that aliasing of branches with same lower address bits and history bits. G-share predictor uses XOR of address and history bits to calculate the index to GPHT. Use of XOR functions improves the utilization of GPHT. If the number of address bits and history bits differs, we take h most significant bits of the address. GPHT entry which is a saturating counter corresponding to XOR outcome will give the final prediction. Fig 3 illustrates how alpha 21264-like predictor works and Equation (2) shows how to calculate its cost.

$$COST = h + 2^a \cdot s \quad (4)$$

III. Simulation Summary

We will summarize the simulation results of the four predictors with different budget sizes and target trace types in this section.

A. Traces

We benchmark our predictor for 4 traces namely INT(integer), FP(floating-point), MM(multi-media) and SER(server). In floating programs, branches have spacial locality as they would be part of some for-loop. Also, patterns of these branches are more regular and shorter patterns. We expect that for FP traces, as we increase budget, the number of address bits will have little impact on miss-rate as aliasing is low in FP traces. Also, small but sufficient history bits are sufficient to saturate the miss-rate of FP traces. Branches in integer programs do have spatial locality but not as good as FP. In the case of integer programs branch outcome has more irregularity and data dependence than FP traces. and thus the branch patterns in integers programs tends to be longer. It requires longer history to efficiently predict the incoming branch. We expect that as the number of history bits increases INT traces would improve miss-rate significantly. Muti-media traces are expected to have high miss-rate as branches are more data-dependent and hence difficult to predict. There is high spatial locality so increasing address bits would have little impact on miss-rate. In the case of MM programs, branches operate on different data each time, for example, pixels in an image. So, a shorter history is preferred for MM as nearby pixels are more correlated that far pixel. We except that MM would work better on shorter history length and number of address bits would have little impact on miss-rate. Server programs execute multiple requests a time. These requests tend to operate in different part of the program. This means that branches in SER would have poor spacial locality. There would an aliasing of branches with same lower LSB bits. Therefore, server trace has a strong correlation between address bits and miss-rate. In server programs, branches have poor spacial locality due to which branches with same LSB bits will aliase. We expect the server miss-rate to decrease sharply on increasing the number of address bits and number of history bits to have little impact on impact on miss-rate.

B. 2-Level Local Predictor

1) Trace Analysis: We analyze the traces to find the correlation between miss-rate and address and history length. As in Fig 6 and Fig 5, we keep history bits constant, but sufficiently high, and increase the address bits (vice-versa for history). We observe that there is a strong correlation between history bits and INT trace. INT traces requires more history bits, or longer history patterns to better predict the branches. SER traces requires more address bits as budget increases. MM traces doesn't work well on longer history as expect. FP programs require less number of history bits as expected. After 9 bits,

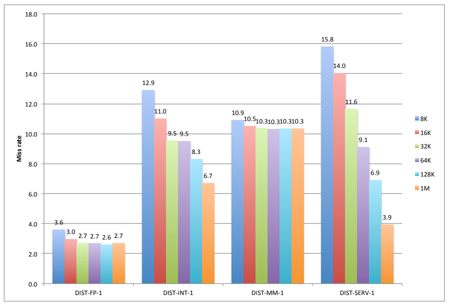

Fig. 4: Simulation results for the 2-level local predictor

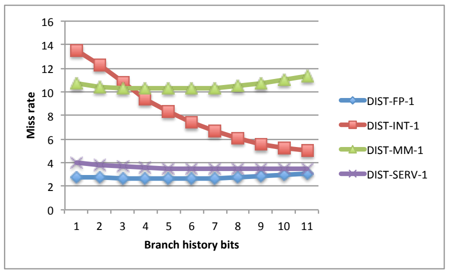

Fig. 5: Address bits = 2, branch history bits 9 to 19

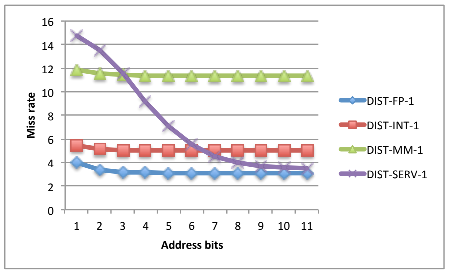

Fig. 6: Branch history bits = 2, address bits from 9 to 19

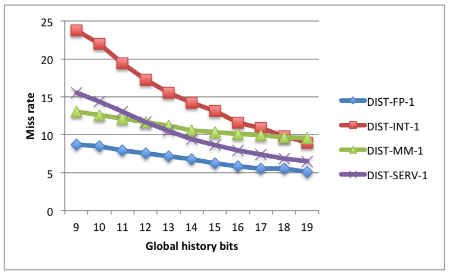

Fig. 7: Alpha Analysis: Local addr bits = 2, Local hist bits = 2, Global hist bits = 9 to 19

there isn't much improvement for MM, INT, FP on increasing address bits and SER, FP on increasing history bits.

2) Results: We expect that by increasing the address bits we use, the miss rate of the server trace will decrease dramatically due to the reduced aliasing. On the other hand, by increasing the width of the branch history table, we will be able to track branch history better. As a result, we should expect a significant decrease in integer trace's miss rate. Simulation results for 2-level predictor are shown in Fig **??**. For INT trace, there is a significant reduction in miss rate mainly due to increase in history bits as budget increase. For SERV, there is a sharp decrease in miss-rate as budget increases. This is mainly due to increase in the number of address bits which reduces branch aliasing. Miss rate is high for multi-media as expected and it doesn't improve significantly as budget increases. Most of the improvement can be incorporated to the reduction in aliasing as address bits increases. FP traces also doesn't improve significantly after 16K budget as history bits are sufficient to capture the branch outcome trends. There is a marginal improvement due to the reduction in aliasing and marginal improvements in branch history lengths. We do notice some small exceptions which are discussed as follows. For integer trace, the miss rate of the predictor doesn't change from 32K budget to 64K budget. After referring to the raw data sheet you will find that the corresponding configurations are 11 bits address and 12 bits history for 32K budget and 12 bits address and 12 bits history for the 64K budget. So this matches our expectation, namely, the number of history bits is the main factor that affects the predictor's performance over integer type trace. On the other hand, you will also notice that the address bits keep increasing as the budget increases. At the same time, the miss rate of the predictor over server trace keeps decreasing sharply, which also matches our expectation.

C. Alpha 21264-like Predictor

1) Trace Analysis: We have already analyzed 2-level local predictor in the previous section. We analyze the global predictor for different traces. We observe in Fig 7 that all trace significantly benefits from increasing history bits. Global predictor is able to capture the global trends to branches across all traces. For MM trace miss-rate reduces as history bits are increased. This can be related to exploitation of global patterns of branches. For example, if we are in a particular section of an image, there would be a global pattern among the branches which can be exploited by the global predictor. We observed in the previous section that local branch history doesn't improve MM miss-rate.

2) Results: Simulation results are shown in Fig 8. We observe that every trace improves from Alpha prediction scheme. Some trace works better with local prediction while some global. Alpha predictor is able to choose the right balance between local and global predictor. For example, MM traces doesn't work good for local predictor but would work better for global. So, alpha predictor would choose global predictor for MM trace and hence we get significant improvements in miss-rate of MM trace. All results are as expected except few exception. FP trace doesn't improve much as expected. Both global and local predictors saturate as budget increases, so not much improvements. INT and SERV improve sharply as history and address bits increases as budget increases.

3) Analysis:

D. Perceptron Predictor

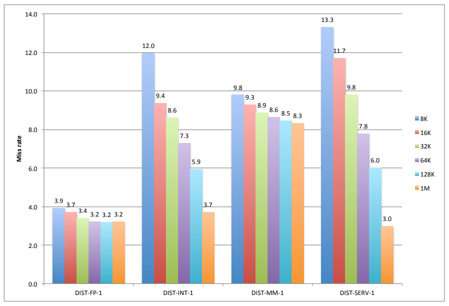

Fig. 8: Simulation results for alpha 21264-like predictor

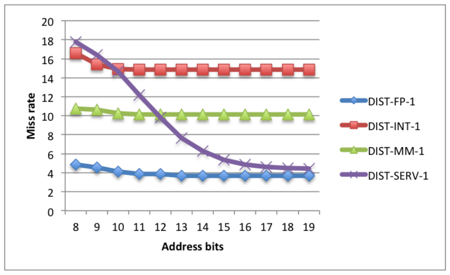

Fig. 9: Perceptron: Trace analysis for addr bits

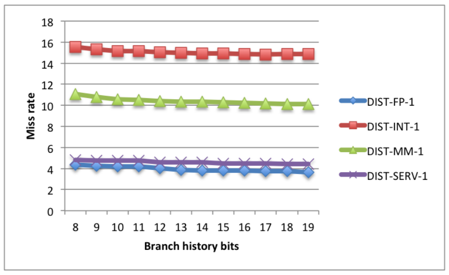

Fig. 10: Perceptron: Trace analysis for hist bits

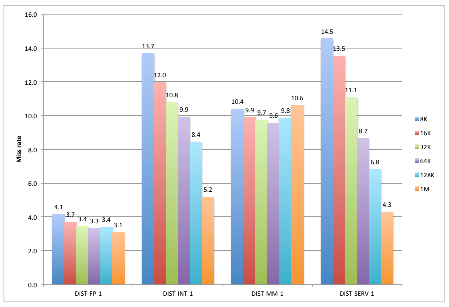

Fig. 11: Simulation results for perceptron predictor

1) *Trace Analysis:* Fig 9 and Fig 10 shows the trace analysis for address and history bits respectively. We keep address bits high when analyzing history to rule out any aliasing factor (vice-versa for history bits). We observe that SERV miss-rate decreases sharply due to reduction in aliasing as budget increases. Number of history bits doesn't significanlty improve miss-rate when address bits are high.

2) *Results:* Simulation results for Perceptron for different budgets are shown in Fig 11. We observe that INT and SERV

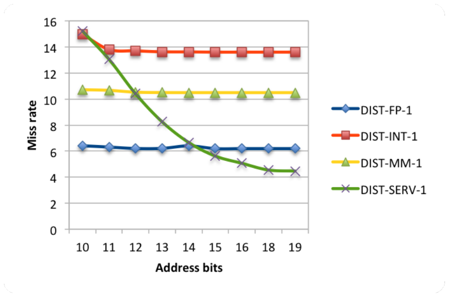

Fig. 12: GShare: Trace analysis for addr bits

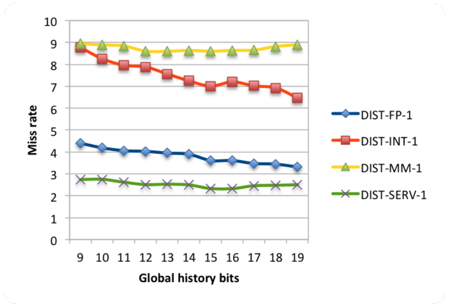

Fig. 13: GShare: Trace analysis for hist bits

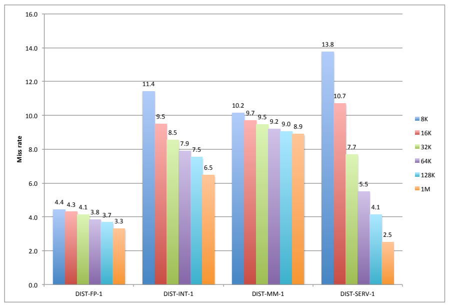

Fig. 14: Simulation results for G-share predictor

traces improves shaply while FP trace also improve significanlty. Expceptinally MM trace miss-rate decreases initially but increases at high budget.

E. G-Share Predictor

1) *Trace Analysis:* Fig 12 and Fig 13 shows the trace analysis for address and history bits respectively. We observe that SERV miss-rate decreases sharply with increasing address bits which is expected. In history analysis figure, we observe that INT trace benefits from increasing history bits which is also expected.

2) *Results:*

3) *Analysis:* Fig 14 shows simulation results for GShare. We choose the number of address bits and number of history bits equal for Ghsare simulation. G-share predictor resolves the aliasing problem by XOR the address bits with the global history. So we expect that it should have been the best predictor for server trace type. And we can see from Fig 18 that G-share indeed has the best performance over server trace type. GShare

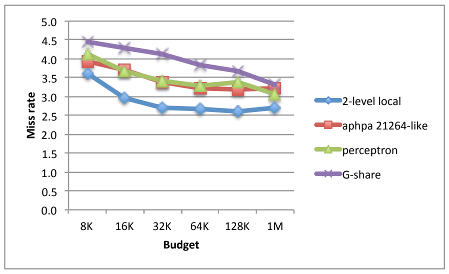

Fig. 15: Simulation results for floating point trace type

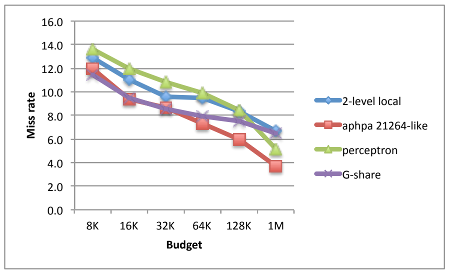

Fig. 16: Simulation results for integer trace type

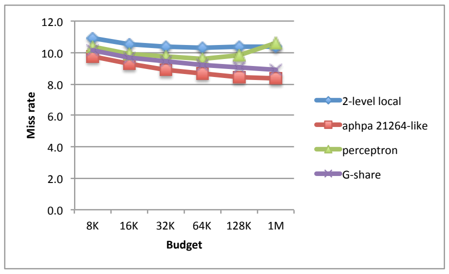

Fig. 17: Simulation results for multimedia trace type

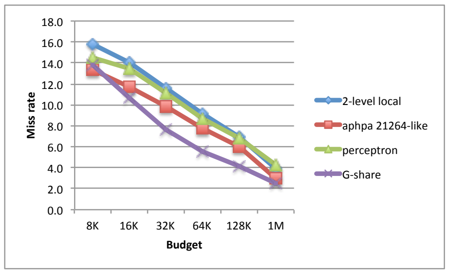

Fig. 18: Simulation results for server trace type

is able to sperate aliasing branches efficiently. Also, traces are benefited from the fact that we are using global history for prediction. As we saw in Trace Analysis of Alpha's global predictor, every trace improved from using global history. GShare adds the benefit of separating aliasing branches. Miss rate for all traces decreases significantly even MM trace. FP and INT traces miss-rate also reduce significantly.

IV. Optimal Predictor

A. *Floating Point*

From Fig 15 we can see that 2-level local predictor has the best performance when working with floating point trace. Floating point branches mostly relies on local history of a branch. Two level predictor captures this local branch pattern and hence the best performance for this trace. All other except GShare have high cost than two-level predictor. So, for a given budget, two level can work better than other predictor. Alpha, Neural predictors have parallel graph to two level predictor.

B. *Integer*

For integer trace, G-share predictor and alpha 21264-like predictor have comparable performance at relatively lower budget. But for higher budget, G-share predictor has better performance than others. As we have mentioned before, integer trace have the longest history pattern among all the four traces. And G-share can have the largest number of history bits under the same budget. So that's why G-share predictor has the best performance here, because it can better record the history of the trace.

C. *Multimedia*

For multimedia trace, alpha 21264-predictor has the best performance for all budgets. Multimedia trace works best on global predictor. Alpha and GShare uses global history to predict the branch. Alpha performs best on multimedia trace followed by GShare. Alpha has local predictor which adds to global predictor. So, alpha selects the right balance hence works best on this trace.

D. *Server*

Finally, for server type trace, G-share predictor perform better than all other predictors. We have discussed in the previous section that for server type trace, the bottleneck that limits its performance is the aliasing problem. And G-share predictor eliminates the aliasing by XOR the address bits with the global history bits. This is the reason behind the observation. At the same time, server type trace can also benefits from G-share predictors' long global history.

V. Conclusion

We observe that all predictors' miss-rate decreases as we increase budget. Integer trace works best for predictors with large history length while Server trace requires more address bits to reduce aliasing. Global predictors improves miss-rate of all the traces as predictors are able to exploit global history patterns. Multimedia trace works best on predictor which uses global history. Floating point trace can be improved by exploiting local history a branch. We conclude that in order to have best performance we can choose 2-level predictor for floating point trace, GShare for integer trace, Alpha for multimedia and GShare for server trace.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Kessler, Richard E. ”The alpha 21264 microprocessor.” *IEEE micro* 19, no. 2 (1999): 24-36.

[3] Lee, Chih-Chieh, I-CK Chen, and Trevor N. Mudge. ”The bi-mode branch predictor.” In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pp. 4-13. IEEE, 1997.

[4] Yeh, Tse-Yu, and Yale N. Patt. ”Alternative implementations of two-level adaptive branch prediction.” In *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 124-134. ACM, 1992.

[5] Jimenez, Daniel A., and Calvin Lin. ”Neural methods for dynamic branch prediction.” *ACM Transactions on Computer Systems (TOCS)* 20, no. 4 (2002): 369-397.

[6] McFarling, Scott. *Combining branch predictors*. Vol. 49. Technical Report TN-36, Digital Western Research Laboratory, 1993.

[7] Daya, Bhavya. ”Analysis of Combined Bimodal and GShare Branch Prediction Schemes.”