Branch Predictor Project Report (CSE 240A)

Alexandria Shearer
PID: A53045665
University of California San Diego
Computer Science and Engineering

Abhinav Mishra
PID: A53103749
University of California San Diego
Computer Science and Engineering

***Abstract*—The goal of this project is to evaluate the impact of the storage budget on the performance of branch predictors. We have written a set of branch predictors that consume a trace of branches based on a standard interface. Each predictor was tested using different storage budgets, (e.g. 32K + 256 bits). The budget encompasses all storage (all tables, plus GHR, if necessary but not PC) used by the predictor. The output of our simulators show a number of branches, number of taken branches, and the misprediction rate. We evaluate the impact of using different storage budgets for each predictor type, as well as analyze which (and why) predictors work better for each budget size and trace type. The predictors we evaluate are the 2-level local predictor, Alpha 21264, Perceptron, and G-Share. After running our experiments, we have determined that the perceptron is the most robust and produces the best overall performance given a storage budget, and the 2-level local predictor needs the parameters like number of local history bits and size of branch address used specifically tuned for the application type to gain the most performance.**

I. Introduction

Branch prediction is a technique for minimizing the amount of work wasted by executing instructions that are more likely to be used. There are two approaches to branch prediction. One is static prediction. However, for complex applications more sophisticated methods are required. Dynamic branch prediction is another approach which entails predicting the outcome of the branch given the history of branching outcomes during runtime. All of the predictors we evaluate for this project are dynamic branch prediction methods.

A. 2-level Local Predictor

The 2-level local predictor builds on the concept of the 1-level predictor. Another level, or table, is added. This table is the Pattern History Table (PHT). The PHT is meant to improve the prediction rate of the 1-level local predictor by using the pattern or local history of outcomes as well as the branch address as an index instead of just the current prediction state. The PHT is indexed using the k LSB of the branch address. Each entry in the PHT records the pattern or history of the branch outcomes. The entries of the PHT are used as indices into the BHT, which holds state or the prediction of the branch. The states in this state machine correspond to the following prediction outcomes: most likely taken, taken, most likely not taken, and not taken. When the outcome of a branch is determined, the outcome alters the state of the particular state machine at that index. The affects the prediction of the next branch that maps to the same location in the BHT. The state machine can have an arbitrary amount of states, however this increases the amount of storage required in the BHT. Increasing the amount of bits used from the branch address will increase the size of the BHT exponentially, but it may also increase the accuracy of the predictor by increasing the ratio of local predictors to branches. If a particular local branch has its own predictor, it will likely have better performance since it is less likely another branch will map to the same predictor and interfere with the results.

B. G-Share Predictor

The G-Share predictor is built on the global branch predictor. Global branch predictors record the history of all of the branch outcomes combined into a single register (BHR). Every time a new branch is evaluated, the global BHR register is shifted and the outcome of the latest branch is recorded in the leftmost bit. Finally, the BHR is used as an index into a Pattern History Table (PHT) as introduced in the 2-level local predictor, which gives the predicted outcome. The G-Share predictor uses the concept of the global BHR and uses an XOR operation with the address of the specific branch as an index into the PHT, which gives the final predicted outcome of the branch.

C. Alpha 21264 Predictor

The Alpha 21264 predictor comes from the Alpha 21264 microprocessor, which implemented a tournament style branch prediction scheme. This branch predictor uses both global and local predictors and then dynamically chooses the predictor which performs better for a given application. This yields a higher prediction accuracy than either predictor individually. The local branch predictor used by Alpha 21264 is the same 2-level local predictor described in this paper. The global branch predictor has a global BHR just like G-Share, which is used to index into a table of saturating counters. A mux receives the predictions from both the local and global branch predictors, and the outcome is selected by another component, the choice predictor. The choice predictor is a table of 2-bit saturating counters as well where four states represent which of the predictors is weakly or strongly chosen [1].

D. Perceptron Predictor

A key issue with many branch predictors is aliasing, where multiple branches map to the same predictors. This can interfere with the performance. Perceptrons address the problem

of aliasing by replacing the tables of saturating counters with perceptrons, which effectively learns the input even if multiple branches map to the same perceptron as long as the branches are linearly independent. A perceptron is a single layer neuron that has several weighted input units and one output unit. It learns a target function that produces a boolean output, which indicates whether or not the branch is taken. The inputs to the perceptron are the n bits of the global BHR. The equation below models a perceptron.

$$w_0 + \sum_{i=1}^{n} x_i w_i = 0$$

The inputs x_1 through x_n are the bits from the global BHR. The weights w_0 through w_n are learned. They correspond to the bits of the global BHR. The perceptron learns the relationship between the global branch history and the outcomes of the relative branches by adjusting the weights. The products of the inputs and the relative weights are summed to create y, the output. The larger the weight, the stronger the correlation is between an individual branch outcome in the global history and the output. A perceptron is trained iteratively with each branch outcome. The branch address is hashed to an index that corresponds to a perceptron in the table. The global BHR and the selected perceptron are both used to compute the branch outcome using the above equation. Following the update, the perceptron is written back to the table[3].

II. Methods

We have written a set of branch predictors that consume a trace of branches. Each predictor was tested using different storage budgets for all tables and registers. Our simulators return the number of branches, number of taken branches, and the misprediction rate. The storage budgets outlined in the project requirements are the following in bits: 8K + 64, 16K + 128, 32K + 256, 64K + 512, 128K + 1K, and 1M + 4K. For all experiments, the misprediction rate is defined as the number of mispredicted instructions per 1000 instructions. For each predictor we picked the ratios of the particular parameters that corresponded to the minimum misprediction rate for each budget size and application type by writing shell scripts that varied the parameters until we found the minimums. We adjusted the provided Makefile so we could divide the source code for each predictor into a different file, per the project requirements. We committed our source code to a git repo, a link to which we provide in the references section[4].

A. 2-Level Local Predictor

The predictor takes in the amount of bits used for the branch address or PC and the amount of bits used for each entry to record the pattern into the local history table. The amount of storage needed given these two parameters would be

$$phtLength * localHistBits + bhtLength * 2$$

where $phtLength = 2^{pcBits}$ is the number of entries in the pattern history table based on the size of the branch address, $localHistBits$ is the number of bits per entry in the pattern history table, and $bhtLength = 2^{localHistBits}$ is the number of entries in the branch history table. The $bhtLength$ is multiplied by 2 because each entry has 2 bits. Increasing the amount of bits used from the branch address increases the size of the tables exponentially, however higher performance should be expected due to the smaller impact of aliasing. Increasing the amount of local history bits used has a similar tradeoff. Table 1 shows the average misprediction rates for the 2-level local predictor across all trace types for the fixed values of $localHistBits$ and $pcBits$.

TABLE I
2-Level Local Predictor Average Misprediction Rates

Budget	localHistBits	pcBits	% Used	Avg. Mispredictions
8K + 64	10	9	86.8%	10.772
16K + 128	11	10	93%	9.625
32K + 256	12	11	99.2%	8.558
64K + 512	12	12	86.8%	7.896
128K + 1K	13	13	93%	7.023
1M + 4K	17	15	77.8%	5.908

B. G-Share Predictor

The predictor takes in the amount of bits used for the global history register in addition to the budget size. The amount of storage used given this parameter would be

$$globalHistBits + bhtLength * 2$$

where $globalHistBits$ is the amount of bits used for the global history register and $bhtLength$ is $2^{globalHistBits}$. Table 2 shows the average misprediction rates for the g-share predictor across all trace types for the fixed values of $globalHistBits$.

C. Alpha 21264 Predictor

The predictor takes in the amount of bits used for the branch address or PC, the amount of bits used for each entry to record the pattern into the local history table, and the amount of bits in the global branch history register. The amount of storage needed given these three parameters would be

$$phtLength * localHistBits + bhtLength * 2$$
$$+choiceLength * 2 * 2 + globalHistBits$$

where $phtLength = 2^{pcBits}$ is the number of entries in the pattern history table based on the size of the branch address, $localHistBits$ is the number of bits per entry in the pattern history table, $bhtLength = 2^{localHistBits}$ is the number of entries in the branch history table, $globalHistBits$ is the

TABLE II
G-Share Predictor Average Misprediction Rates

Budget	globalHistBits	% Used	Avg. Mispredictions
8K + 64	12	88.4%	9.948
16K + 128	13	99.3%	8.553
32K + 256	14	99.3%	7.454
64K + 512	15	99.2%	6.615
128K + 1K	16	99.2%	6.099
1M + 4K	19	99.6%	5.297

number of bits in the global BHR, and $choiceLength = 2^{globalHistBits}$ is the number of entries in the choice predictor table which chooses between the 2-level local predictor and the global predictor. Table 3 shows the average misprediction rates for the alpha 21264 predictor across all trace types for the fixed values of $globalHistBits$, $localHistBits$, and $pcBits$.

TABLE III
ALPHA 21264 PREDICTOR AVERAGE MISPREDICTION RATES

Budget	global bits	local bits	pc bits	Used	Mispredictions
8K + 64	10	4	10	99.7%	10.095
16K + 128	11	10	9	93.1%	8.910
32K + 256	12	11	10	96.2%	7.695
64K + 512	13	12	11	99.2%	6.602
128K + 1K	13	14	12	93.0%	5.906
1M + 4K	16	16	15	87.2%	4.218

D. Perceptron Predictor

The perceptron predictor requires a global BHR for the global history, the PC or branch address, and a table of perceptrons. A perceptron is essentially a set of weights since they all share the same inputs. Our command line accepts the number of bits used for the global BHR, the number of bits in the PC address, as well as the budget size in bits. The amount of storage needed given these two parameters would be

$$(globalHistBits + 1) * pLength * BW + globalHistBits$$

where $globalHistBits$ is the number of bits in the global history register, $pLength = 2^{pcBits}$ is the number of entries in the perceptron table, and BW is a constant that determines the range of the weights. For our experiments, BW has the value of 6, which means a range of 31 to -32. This is a very common choice. Table 4 shows the average misprediction rates for the perceptron predictor across all trace types for the fixed values of $globalHistBits$ and $pcBits$.

TABLE IV
PERCEPTRON PREDICTOR AVERAGE MISPREDICTION RATES

Budget	globalHistBits	pcBits	% Used	Avg. Mispredictions
8K + 64	9	7	93.1%	8.874
16K + 128	20	7	97.8%	7.589
32K + 256	20	8	97.7%	6.366
64K + 512	20	9	97.7%	5.481
128K + 1K	32	9	76.8%	4.781
1M + 4K	34	12	81.7%	4.075

III. RESULTS

In order to evaluate the performance of the predictors for varying budget types for one set of experiments we fixed the parameters described in the methods such that they would provide the best performance on average for each application and budget. Tables 1 through 4 report the values of the parameters to the predictors that provided the minimum average misprediction rates *across all trace types.* In some cases, these values were different from the values that provided the minimum misprediction rate for a particular application type as shown in the tables in the appendix. For example, for the G-Share predictor, the Integer and Multimedia applications with a budget size of 1M + 4K performed better with 18 bits in the global BHR than 19, though on average 19 bits had a lower misprediction rate for all applications (Table 6: Appendix).

A. Trace Type Results

Figure 1 shows the results of each predictor and budget size for the integer trace type. Figure 2 shows the results for the floating point trace type. The multimedia and server trace types are shown in Figures 3 and 4, respectively. We discuss these results in more detail in the following section.

IV. DISCUSSION

The goal of this project is to evaluate the impact of the storage budget on the performance of branch predictors for different application types. Ultimately, some branch predictors are better suited for some applications, and while the storage budget can play a critical role in the performance of a particular predictor, sometimes additional storage does not provide much of a benefit if the predictor is not suited to an application type. **Ideally, when the storage budget is increased, the performance of the predictor should increase and thus the misprediction rate should decrease.**

A. Expectations

For the local predictor, a larger budget would entail either more predictors which would reduce the effects of aliasing, or it would mean more storage per entry in the pattern history table which would enable the predictor to recognize more complicated branching patterns.

For the g-share predictor, a larger budget would entail more storage for the global history register. A longer global history register would entail a larger branch history table which would reduce the effects of aliasing and better recognition of more complicated global branching patterns.

Since the alpha 21264 branch predictor encompasses both the 2-level local predictor and global predictor, a larger budget would entail similar effects except it would leverage the advantages of both.

For the perceptron predictor a larger budget would entail more inputs from the global history register, which would enable it to recognize more complicated global branching patterns, and a larger amount of perceptrons. Since aliasing is less of an issue with the perceptron, which learns the linear relationships among its inputs, perceptrons should have a greater gain for budget increases than should the other predictors for an agreeable trace type.

When it comes to trace types, we expected that the floating point trace type would be the most predictable since such applications have a lot of for loops and typically do not have complicated branching conditions. All of the predictors should perform well with it. We also expected that the server trace type would not perform as well with the local predictor as the rest of the predictors because server applications generally have coarser-grained levels of parallelism like transaction level

parallelism and therefore could not be sufficiently recognized by local predictors. Integer trace types should perform well with increasing budget sizes since both local and global predictors capture more complicated branching patterns with an increasing budget. Perceptrons too, should perform well with the integer trace type since most integer applications and therefore branching conditions are linear relations. The multimedia applications can take several forms with the introduction of user interactivity. Multimedia applications should have finer-grained parallelism due to similar operations on data, for example, filtering pixels or audio, but those operations may also be non-linear and the loops may therefore have interdependencies. With the introduction of user interactivity, multimedia applications may also have layers of coarser-grained parallelism like handling events set in motion by user input. For these reasons, we predict that alpha 21264 and perceptron predictors will perform the best with the multimedia trace type. The alpha predictor is at least more adaptable in terms of being able to switch between global and local prediction and the perceptron should be able to adapt to changes in the branching behavior by continually adjusting its weights. Both the alpha and perceptron predictors should be able to perform better with an increasing budget since a greater amount of perceptrons and the ability to handle more complex branching patterns with more storage are features of the perceptron and alpha predictors, respectively[2].

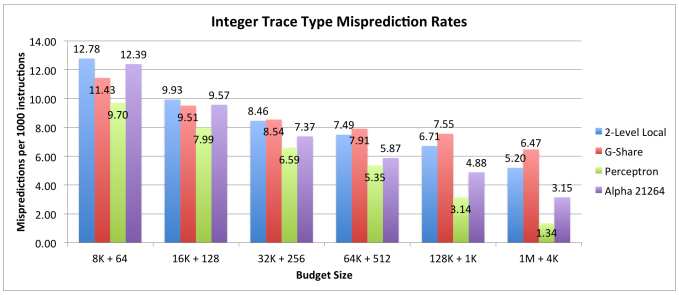

Fig. 1. Integer trace type misprediction rates for varying budget sizes and predictors.

B. Integer Trace

We predicted that integer trace types should perform well with increasing budget sizes for all predictors since both local and global predictors capture more complicated branching patterns with an increasing budget and it is more likely for integer applications to have branching conditions with linear relationships for the perceptron to recognize. These predictions too, were correct (Figure 1). All predictors were robust and increased in performance for each budget size, although it is clear for the higher budgets that g-share has some limitations in that it doesn't recognize the local branching behaviors of the trace. It makes sense that the local and global predictors never provide the best performance for the same budget because global prediction is necessary to understand the interdependency of the branching conditions of several integer applications, yet without the local prediction in alpha 21264 the g-share is limited. Given this, the perceptron is the best performer. We suspect that this is because the perceptron uses a global branch history register to capture the global branching behavior, yet the perceptrons are also able to effectively learn the local patterns as well. The learning component of the perceptron predictor proved more effective than the alpha which simply chooses between the best of local and global prediction.

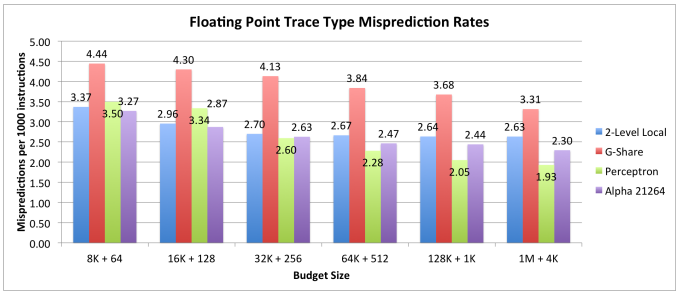

Fig. 2. Floating point trace type misprediction rates for varying budget sizes and predictors.

C. Floating Point Trace

We found that many of our predictions turned out to be true. For one, the floating point trace type had the lowest misprediction rates for all predictors and budget sizes (Figure 2). It was in fact the most predictable trace type. In general, the 2-level local and alpha predictors were more effective for floating point than the G-Share predictors. We suspect this is because the floating point trace has more local branching behavior than complicated global branching behavior because of simple for loops. Perceptron was the most robust in terms of improving performance with each storage increase. Oddly, we had initially suspected that floating point trace types would respond best to local prediction, and for the largest possible budget bracket the 2-level local predictor performance hardly improved when comparing the 128K + 1K budget performance to 1M + 4K.

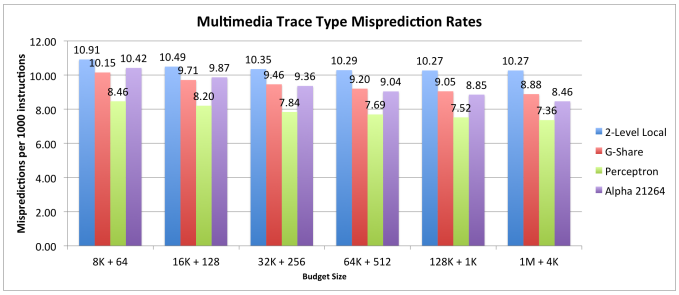

Fig. 3. Multimedia trace type misprediction rates for varying budget sizes and predictors.

D. Multimedia Trace

As expected, the predictors we the least robust to the multimedia trace type when increasing budget sizes since

the multimedia applications are the least predictable with the possible introduction of user input or the non-linearity of the operations performed on media data (Figure 3). While the server trace type initially has worse misprediction rates for a budget of 8K + 64 across all predictors, the predictors improve with an increasing budget to achieve smaller misprediction rates for a budget of 1M + 4K. The same is not true for the multimedia trace, though we did predict that the alpha and perceptron predictors would work better than g-share and 2-level local, which turned out to be somewhat true. The perceptron performed the best across all budget types as expected, but the alpha predictor started out the worst and became mildly robust, having the second best result of 8.46 mispredictions for the budget of 1M + 4K. The 2-level local and g-share predictors were hardly robust. The 2-level local predictor basically stagnated in performance after we increased the budget to 16K + 128, and even produced almost same results for 1M + 4K and 128K + 1K (10.27).

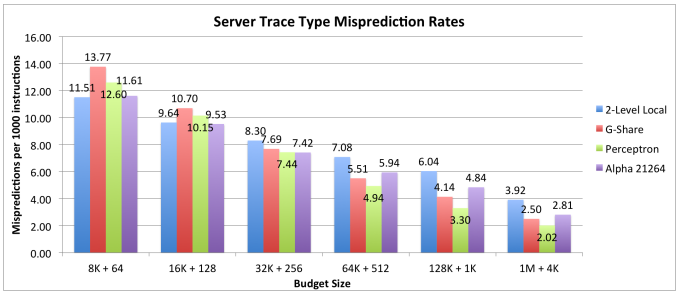

Fig. 4. Server trace type misprediction rates for varying budget sizes and predictors.

E. Server Trace

For the server trace type, we predicted local prediction would not perform as well. This prediction also turned out to be correct. The local predictor was consistently the worst performer across storage budgets greater than or equal to 32K + 256 (Figure 4). The G-Share and Perceptron predictors were the most robust in recognizing the coarse-grained transaction-level parallelism of server applications for larger budget sizes. While the alpha predictor is supposed to leverage the best of local and global predictions, it did not perform as well as g-share for all budgets because the storage required for the local and choice predictors interfered with the ability of the global predictor in alpha to capture the global branching patterns of the server trace type.

V. Conclusion

We determined that **the impact of budget size on the performance of branch predictors was highly context dependent on the trace type**. The alpha 21264 predictor was robust to increasing budget sizes for the integer and server trace types and performed the best for floating point, but was not robust at all to multimedia traces. Similarly, the perceptron predictor was robust to increasing budget sizes for integer, floating point, and server traces, but not as much for multimedia although it was overall the best performer. Additionally, the 2-level local predictor has parameters that need to be tuned to the trace type or else increasing storage can accrue more mispredictions, although it is robust for server and integer applications. Global prediction was very strong for server and integer traces, and not as strong for floating point or multimedia. In general, the most adaptable and robust predictor was the perceptron, which was a strong contender for all trace types and robust for all but the multimedia trace type. When considering which predictors perform best across a specific budget size, we find that for 2/4 trace types, the 2-level local predictor performs poorly for the smallest budget compared to the other predictors. Additionally, **for ALL of the trace types, the perceptron performs the best for a large budget**. If we had a large storage budget, the perceptron is arguably the best predictor to implement because it performs so well across all trace types. It is also important to consider cases where we have a very small budget available. Perceptron is also the best performer for 2/4 traces at the smallest budget size. And even in the floating point and server trace types it basically results in only an extra missed prediction cost above the best performing predictor. For budgets that fall between 16K + 128 and 128K + 1K and the restriction that we cannot use perceptron, g-share and alpha predictors perform similarly across all trace types, although alpha provides slightly better performance overall. Alpha is comparable to g-share in the 16K to 128K budget range for multimedia and server, and outperforms g-share for floating point and integer with a larger budget.

References

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

[2] Gabriel H. Loh. "Simulation Differences Between Academia and Industry: A Branch Prediction Case Study".

[3] Daniel A. Jimenez and Calvin Lin. "Dynamic Branch Prediction with Perceptrons". The University of Texas at Austin.

[4] Our Branch Prediction Git Repository. https://github.com/abhinav-mishra/branchprediction.

[5] Championship Branch Predictor Contest. url: http://www.jilp.org/cbp2016/.

[6] CSE240A-Fall 2016 Webpage. url: http://cseweb.ucsd.edu/classes/fa16/cse240A-a/.

[7] 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.

[8] IEEE Manuscript Templates for Conference Proceedings. url:http://www.ieee.org/conferences_ events/conferences/publishing/templates.html.

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

[10] 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.

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

[12] 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.

[13] UCSD Policy on Integrity of Scholarship. url: https://students.ucsd.edu/academics/academic-integrity/policy.html.

Appendix

TABLE V
2-LEVEL LOCAL PREDICTOR MINIMUM MISPREDICTION RATES PER BUDGET

Input File	Target Budget (bits)	Local History Bits	PC Bits	Percent Budget Used	Misprediction Rate
DIST-INT-1	8k + 64	11	8	83.7%	12.778
	16k + 128	12	9	86.8%	9.931
	32K + 256	13	10	89.9%	8.460
	64K + 512	14	11	93.0%	7.492
	128K + 1K	15	12	96.1%	6.714
	1M + 4K	18	14	77.8%	5.195
DIST-FP-1	8k + 64	7	10	89.9%	3.369
	16k + 128	11	10	93.0%	2.956
	32K + 256	12	11	99.2%	2.701
	64K + 512	13	11	65.1%	2.665
	128K + 1K	13	13	93.0%	2.636
	1M + 4K	13	16	82.5%	2.634
DIST-MM-1	8k + 64	10	9	86.8%	10.912
	16k + 128	11	10	93.0%	10.494
	32K + 256	12	11	99.2%	10.349
	64K + 512	12	12	86.8%	10.286
	128K + 1K	13	13	93.0%	10.268
	1M + 4K	13	16	82.5%	10.268
DIST-SERV-1	8k + 64	2	12	99.3%	11.509
	16k + 128	2	13	99.3%	9.637
	32K + 256	2	14	99.2%	8.303
	64K + 512	4	14	99.3%	7.084
	128K + 1K	8	14	99.6%	6.042
	1M + 4K	15	16	99.6%	3.916

TABLE VI
G-SHARE PREDICTOR MINIMUM MISPREDICTION RATES PER BUDGET

Input File	Target Budget (bits)	Global History Bits	Percent Budget Used	Misprediction Rate
DIST-INT-1	8k + 64	12	99.4%	11.432
	16k + 128	13	99.3%	9.505
	32K + 256	14	99.3%	8.541
	64K + 512	15	99.2%	7.909
	128K + 1K	16	99.2%	7.533
	1M + 4K	18	99.6%	6.470
DIST-FP-1	8k + 64	12	99.4%	4.443
	16k + 128	13	99.3%	4.298
	32K + 256	14	99.3%	4.131
	64K + 512	15	99.2%	3.838
	128K + 1K	16	99.2%	3.676
	1M + 4K	19	99.6%	3.314
DIST-MM-1	8k + 64	12	99.4%	10.151
	16k + 128	13	99.3%	9.710
	32K + 256	14	99.3%	9.458
	64K + 512	15	99.2%	9.204
	128K + 1K	16	99.2%	9.049
	1M + 4K	18	49.8%	8.884
DIST-SERV-1	8k + 64	12	99.4%	13.767
	16k + 128	13	99.3%	10.699
	32K + 256	14	99.3%	7.686
	64K + 512	15	99.2%	5.508
	128K + 1K	16	99.2%	4.138
	1M + 4K	19	99.6%	2.504

TABLE VII
ALPHA 21264 PREDICTOR MINIMUM MISPREDICTION RATES PER BUDGET

Input File	Target Budget (bits)	Global History Bits	Local History Bits	PC Bits	Percent Budget Used	Misprediction Rate
DIST-INT-1	8K + 64	8	11	8	96.2%	12.39
	16K + 128	9	12	9	99.3%	9.572
	32K + 256	11	13	9	94.6%	7.374
	64K + 512	12	14	10	96.1%	5.873
	128K + 1K	13	15	10	86.1%	4.884
	1M + 4K	17	17	13	87.9%	3.148
DIST-FP-1	8k + 64	7	7	10	96.2	3.270
	16K + 128	10	10	10	99.3%	2.872
	32K + 256	11	10	11	93.1%	2.629
	64K + 512	13	12	11	99.2%	2.466
	128K + 1K	14	13	11	82.2%	2.441
	1M + 4K	17	13	13	61.5%	2.296
DIST-MM-1	8k + 64	10	9	8	90.0%	10.416
	16K + 128	11	10	9	93.1%	9.868
	32K + 256	12	11	10	96.2%	9.362
	64K + 512	13	12	11	99.2%	9.042
	128K + 1K	14	13	11	82.2%	8.854
	1M + 4K	17	15	14	79.4%	8.463
DIST-SERV-1	8k + 64	10	1	12	99.4%	11.613
	16K + 128	11	1	13	99.3%	9.529
	32K + 256	12	2	13	99.3%	7.417
	64K + 512	13	2	14	99.3%	5.943
	128K + 1K	14	2	15	99.2%	4.841
	1M + 4K	16	11	16	93.8%	2.813

TABLE VIII
PERCEPTRON PREDICTOR MINIMUM MISPREDICTION RATES PER BUDGET

Input File	Target Budget (bits)	Global History Bits	PC Bits	Percent Budget Used	Misprediction Rate
DIST-INT-1	8k + 64	9	7	93.1%	9.695
	16k + 128	19	7	93.1%	7.987
	32k + 256	20	8	97.7%	6.587
	64k + 512	33	8	79.1%	5.351
	128K + 1K	41	9	97.7%	3.142
	1M + 4K	64	11	75.9%	1.342
DIST-FP-1	8k + 64	20	6	97.9%	3.495
	16k + 128	20	7	97.8%	3.336
	32k + 256	36	7	86.1%	2.595
	64k + 512	64	7	75.7%	2.282
	128K + 1K	64	8	75.6%	2.051
	1M + 4K	64	10	37.9%	1.931
DIST-MM-1	8k + 64	9	7	93.1%	8.464
	16k + 128	18	7	88.5%	8.204
	32k + 256	20	8	97.7%	7.838
	64k + 512	20	9	97.7%	7.693
	128K + 1K	29	9	69.8%	7.523
	1M + 4K	30	12	72.4%	7.363
DIST-SERV-1	8k + 64	4	8	99.1%	12.600
	16k + 128	4	9	93.0%	10.149
	32k + 256	9	9	93.1%	7.435
	64k + 512	20	9	97.7%	4.937
	128K + 1K	20	10	97.7%	3.300
	1M + 4K	32	12	77.0%	2.017