Computational and Storage Power Optimizations for the O-GEHL Branch Predictor

Kaveh Aasaraai
aasaraai@ece.uvic.ca

Amirali Baniasadi
amirali@ece.uvic.ca

Ehsan Atoofian
eatoofia@ece.uvic.ca

Electrical and Computer Engineering Department
University of Victoria
3800 Finnerty Rd.
Victoria, BC, Canada

ABSTRACT

In recent years, highly accurate branch predictors have been proposed primarily for high performance processors. Unfortunately such predictors are extremely energy consuming and in some cases not practical as they come with excessive prediction latency. One example of such predictors is the O-GEHL predictor. To achieve high accuracy, O-GEHL relies on large tables and extensive computations and requires high energy and long prediction delay.

In this work we propose power optimization techniques that aim at reducing both computational complexity and storage size for the O-GEHL predictor. We show that by eliminating unnecessary data from computations, we can reduce both predictor's energy consumption and delay. Moreover, we apply information theory findings to remove redundant storage, without any significant accuracy penalty. We reduce the dynamic and static power dissipated in the computational parts of the predictor by up to 74% and 65% respectively. Meantime we improve performance by up to 12% as we make faster prediction possible.

Categories and Subject Descriptors

C.1.0 [**Computer Systems Organization**]: Processor Architectures—*General*

General Terms

Design Performance

Keywords

Power-Aware Microarchitectures, O-GEHL, Branch Prediction

1. INTRODUCTION

Perceptron based predictors are highly accurate. The high accuracy is the result of exploiting long history lengths [9] and is achieved at the expense of high complexity.

The Optimized GEometric History Length (or simply O-GEHL) predictor is an example of a perceptron like predictor. O-GEHL relies on exploiting behavior correlations among branch instructions. To collect and store as much information as possible, the O-GEHL branch predictor uses multiple tables equipped with wide counters. The predictor uses the collected data and performs many steps before making the prediction. These steps include reading several counters from the tables and performing several computations (e.g., additions and comparisons) on the collected data.

In this work we revisit the O-GEHL predictor and show that while the conventional scheme provides high prediction accuracy, it is not efficient from the energy point of view. We are motivated by the following observations. First, our study shows that not all the computations performed by O-GEHL are necessary. This is particularly true for computations performed on counter lower bits. As we show later, not all counter bits always impact the prediction outcome. Therefore excluding less important bits from the computations, while reducing energy consumption, may not impact accuracy. Second, we have observed that the tables used by O-GEHL store redundant data. We show that the stored data can be represented using less storage if this redundancy is taken into account.

We rely on the above observations and introduce two power optimization techniques. Our techniques aim at reducing the power dissipated by the computation and storage resources. We reduce power for computation resources by eliminating unnecessary and redundant counter bits from computations and by accessing and using fewer bits at the prediction time. We reduce power for storage resources by representing the required data using less bits. We achieve this by having multiple counters sharing their lower bits. We show that by intelligent bit sharing it is possible to reduce predictor size while maintaining its accuracy. It should be noted that since our optimizations are not performed dynamically, they come with no latency or power overhead at runtime.

By applying our techniques we not only reduce power but also improve processor performance. This is due to the fact

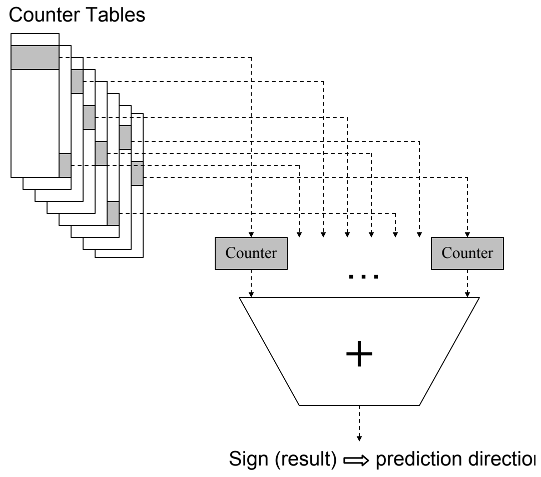

Figure 1: The O-GEHL branch predictor using multiple tables and different indexes. Sum of all the counters is used to make the prediction.

that by eliminating unnecessary computations we reduce prediction latency, resulting in a faster yet highly accurate prediction. We reduce the dynamic and static power dissipation associated with predictor computations by 74% and 65% respectively while improving performance up to 12%.

The rest of the paper is organized as follows. In Section 2 we discuss O-GEHL background. In Section 3 we discuss the motivation. In Sections 4 and 5 we introduce our optimizations. In Section 6 we explain our simulations methodology and report results. In Section 7 we discuss related work. In Section 8 we offer concluding remarks.

2. BACKGROUND

The O-GEHL branch predictor [14] uses multiple tables to store counters for each branch instruction. O-GEHL indexes tables relying on different hashing functions which use different history lengths. This is mainly done to exploit different history patterns and correlations while reducing the destructive aliasing effect. The history lengths used form a geometric series making use of both long and short history lengths possible.

As presented in Figure 1, the O-GEHL predictor takes the following steps to make a prediction. First, the predictor loads each counter from a different table using different indexing functions. Second, an adder tree computes the sum of all the counters. Third, the predictor makes the prediction based on the sum's sign.

At the update time, the predictor updates the counters using the actual outcome of the branch instruction. In case of a taken branch, the predictor increments all the associated counters. The predictor decrements all associated counters if the branch's outcome is not taken.

The correlation among branch instructions is collected using the constructive aliasing implicit in the indexing functions. A counter being used by two or more different branch instructions is practically storing correlation data between those branches effectively improving accuracy.

Counter Values

Decimal	1	-3	2	-1	-8	5	4	-2
Binary	0001	1101	0010	1111	1000	0101	0100	1110

Result (All Bits) = 1 + -3 + 2 + -1 + -8 + 5 + 4 + -2 = -2 → Not Taken
Result (High Bits) = 0 + -1 + 0 + -1 + -2 + 1 + 1 + -1 = -3 → Not Taken

Figure 2: The first calculation uses all counter bits and predicts the branch outcome as "Not taken". The second one uses only higher bits (underlined) and results in the same direction.

3. MOTIVATION

As presented in Figure 1, the O-GEHL predictor uses multiple counters per branch instruction. For every direction prediction, the predictor computes the sum of all the indexed counters.

The complexity of the computations involved in the summation process makes it a slow and energy hungry one. This process requires an adder tree, with a size and complexity proportional to the counters' widths. The wider the counters are, the more complex the summation process will be. Note that the 64Kbits O-GEHL predictor proposed for CBP-1 [13] uses a combination of 4- and 5-bit counters to achieve the best accuracy.

Our study shows that the conventional O-GEHL scheme is not efficient from the energy point of view. In particular we have made the following observations.

a) Our study shows that not all counter bits are equally important in making accurate predictions. In particular, higher order bits are more likely to impact the final outcome compared to lower order bits. In Figure 2 we present an example to provide better understanding.

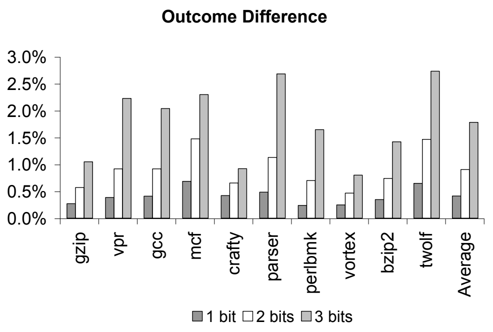

Figure 3: How often removing the lower n bits from the computations results in a different outcome compared to the scenario where all bits are considered. Bars from left to right report for scenarios where one, two or three lower bits are excluded.

To investigate this further, in Figure 3 we report how excluding the lower n bits of each counter impacts prediction outcome. As reported, on average, 0.4%, 0.9%, and 1.8% of time eliminating the lower one, two or three bits results in a different outcome respectively. This difference is worst (2.7%) when the lower three bits are excluded for *twolf* (See Section 6 for methodology).

We conclude from Figure 3 that eliminating lower order bits (referred to as LOBs) of the counters from the prediction process and using only higher order bits (referred to as HOBs) would not significantly affect the predictor's accuracy. We use this observation and reduce predictor's latency and power dissipation.

b) In Figure 4 we report how often the three lower order bits of the counters used by O-GEHL have the maximum or minimum possible values. As reported on average more than 60% of the time the counter value is biased. This suggests that using three bits per counter may be too much as not all counter values are equally frequent. This bias motivates us to reduce storage size by representing lower order bits using fewer resources (more on this in Section 5).

Based on the above observations, in this work we propose two optimization methods for the O-GEHL branch predictor. First, we reduce the computation complexity associated with making predictions. Second, we reduce the predictor's table sizes as we represent the same information using less storage.

4. LOB ELIMINATION

Considering the data presented in Section 3, we suggest eliminating the LOBs of the counters from the lookup and summation process. During predictor update, however, we increment/decrement counters without excluding any of the lower bits.

In Figure 5 we present our scheme. In this scheme, the adder tree does not load or use all counter bits. Instead, the adder tree bypasses the LOBs of the counters, and performs the accumulation process only on the HOBs. Eliminating LOBs reduces power but can impact accuracy and performance.

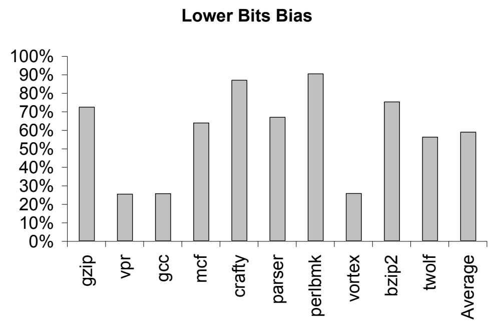

Figure 4: Bars show the percentage of time three lower bits of the counters are biased. On average 60% of time counters are biased compared to 25% on a uniform distribution.

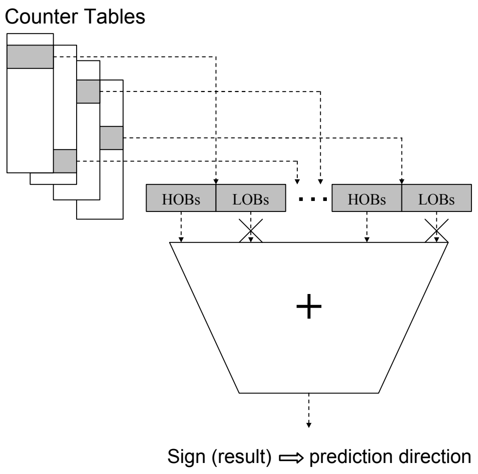

Figure 5: The optimized adder tree bypasses LOBs of the counters and performs the addition only on the HOBs. Eliminating LOBs results in a smaller and faster adder tree.

a) Accuracy and Performance: The number of input bits decides the adder tree size used in the O-GEHL predictor. Excluding LOBs from the computation reduces adder tree's input size resulting in a shorter computation time. This makes faster prediction possible but can potentially harm accuracy.

A previous study on branch prediction delay shows that further prediction accuracy improvement may not be worthwhile if it results in a slower prediction scheme[8]. Reportedly a relatively accurate single-cycle latency predictor outperforms a 100% accurate predictor with two cycles latency[8]. This observation motivates us to study whether the prediction speedup obtained by eliminating LOBs is worth the accuracy cost. In Section 6 we investigate this tradeoff and show that for the benchmarks used in this work the performance improvements achieved by faster prediction outweigh the cost associated with the extra mispredictions.

b) Power: We reduce both the dynamic and the static power dissipated by the predictor. As we reduce adder tree's size, fewer gates are used in its structure. Consequently, it dissipates less static power. We also reduce dynamic power as fewer switching activities occur.

We eliminate LOBs from the computations necessary at the prediction time. Therefore reading all bitlines is no longer necessary. One straightforward mechanism to make this possible is to decouple LOBs and HOBs. Accordingly, we store LOBs and HOBs in two separate set of tables. Figure 6 shows this in more detail.

At the prediction time, the predictor accesses only the set of tables storing the HOBs, saving the energy consumed for accessing LOBs in the conventional O-GEHL predictor. Note that while we save the energy spent on wordline, bitline and sense amplifiers, we do not reduce the decoder energy consumption as we do not reduce the number of table entries for this optimization.

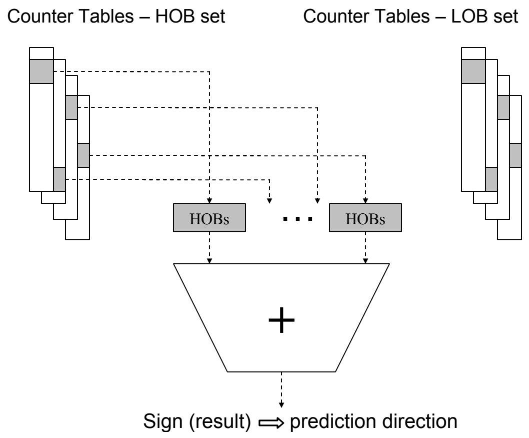

Figure 6: Predictor tables are divided into two sets, HOB set and LOB set. Only the HOB set is accessed at the prediction time in order to reduce power dissipation.

5. ENTROPY-AWARE STORAGE

In this Section we propose a scheme to reduce predictor tables' sizes. We measure the information conveyed by the counters and compute their entropy and the necessary number of bits required to carry this information. Knowing the exact entropy of these bits, we can avoid dedicating unnecessary storage.

5.1 Entropy

The formal definition of the information a message carries depends on the probability of that message. The information carried by each message can be represented as:

$$I = -\log_2 P$$

where P is the message occurrence probability. Shannon's theorem states that number of bits needed to represent a messages set is equal to its entropy. Entropy is defined as:

$$H = \sum_{i \in M} P_i I_i$$

where M is the message set. If we have a b-bit counter, we have 2^b possible messages. Therefore we can compute its entropy as:

$$H = \sum_{i=0}^{2^b-1} P_i I_i = \sum_{i=0}^{2^b-1} -P_i \log_2 P_i$$

where P_i is the probability of the counter having a value of i. In the case of having equal probabilities for all counter values, message set entropy is equal to b bits, which is the number of bits we have already dedicated. However, if any message has a higher probability, message set's entropy decreases to a value less than 2^b, meaning that it is possible to represent this message set with less than b bits. Knowing the amount of information carried by each message in the set, we can derive the set's entropy and determine the actual number of bits required for representation.

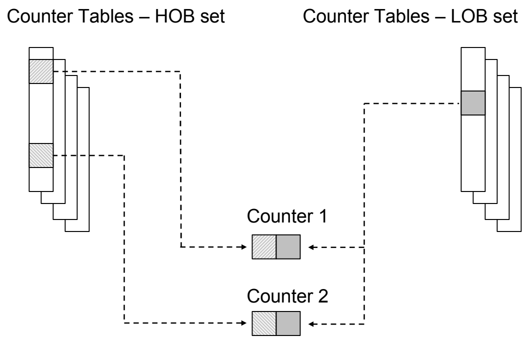

Figure 7: Two counters share their LOBs. The predictor indexes the same LOB entry for counters using different HOBs.

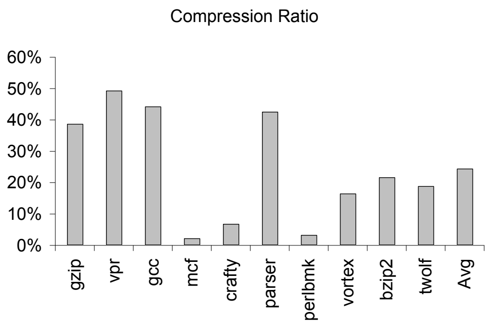

Figure 8: Compression ratio for data carried by three LOBs. On average, the data can be compressed to one-fourth of its size.

Note that the entropy of the message set may not always be an integer number, requiring several entries sharing a single bit. To this end, we exploit sharing groups in which counters share their bits. Under this situation, the group size determines the real number of bits being dedicated to each counter. A previous study has used a similar approach to share the hysteresis bits of the Gshare predictor [10]. In Figure 7 we show an example clarifying this further.

To reduce storage size we exploit the bias of the LOBs presented in Section 3, and change the information representation. In order to measure LOBs' entropy, we record counters values at prediction time. We compress the stored information using gzip program with "–best" flag. Similar to previous study we use the compression ratio to estimate the entropy of the message set [10].

Figure 8 shows the compression ratio for the data carried by three lower bits of the O-GEHL counters. As reported on average less than 25% of current reserved bits suffice to carry the information. In order to fascilitate using fewer bits, we have multiple entries share their LOBs. We form counter groups of up to size four, effectively avoiding using

Table 1: Processor Microarchitectural Configurations

Fetch/Decode/Commit	6
BTB	512
L1 I-Cache	32 KB, 32B blk, 2 way
L1 D-Cache	32 KB, 32B blk, 4 way
L2 Unified-Cache	512 KB, 64B blk, 2 way
L2 Hit Latency	6
L2 Miss Latency	100
Predictor Budget	64Kbits

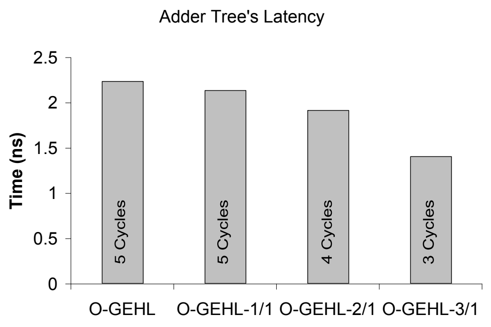

Figure 9: Time / Cycle required to compute the sum of counters.

the redundant storage. However, and in order to maintain accuracy, conservatively, we use a group size of two where every two counters share their LOBs. Note that having a power of two group size simplifies implementation issues. As a result, we reduce the table size by half.

6. METHODOLOGY AND RESULTS

For our simulations, we modify the SimpleScalar tool set [4] to include the conventional O-GEHL branch predictor and our proposed optimizations. We use Simpoint [15] to identify representative 500 million instructions regions of the benchmarks. We use a subset of SPEC2K-INT benchmarks for our simulations.

Table 1 reports the baseline processor used in our study. For predictor configuration, we use the 64Kb budget O-GEHL predictor proposed in CBP-I [13].

For predictor energy and timing reports, we use Synopsys Design Compiler synthesis tool assuming the 0.18 um technology. We use the high effort optimization option of the Design Compiler, and optimize the circuit for delay. We simulated both the conventional and the optimized O-GEHL predictors. We also use CACTI [16] to compute the predictor table access time and power dissipation.

For our simulations, we assume the processor has a 2GHz frequency.

In the interest of space we use the O-GEHL-n/s notation to refer to different low-complexity predictors suggested in this study. We use O-GEHL-n/s to refer to an O-GEHL predictor in which the lower n bits of the counters are excluded

Table 2: Table Access Time / Power Dissipation

Predictor	Access Time (ns)		Energy (pJ)	
	Predict	Update	Predict	Update
O-GEHL	1.15	1.15	167.04	167.04
O-GEHL-1/1	1.15	1.15	149.02	167.55
O-GEHL-2/1	1.10	1.10	130.68	167.52
O-GEHL-3/1	1.08	1.10	112.34	167.52

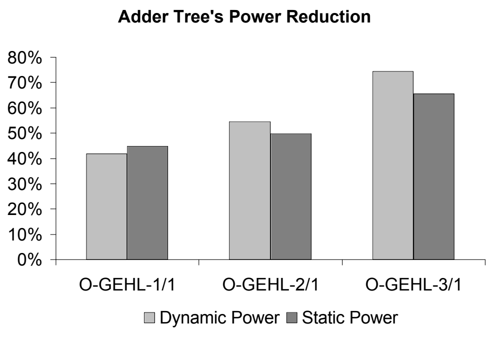

Figure 10: Energy reduction of the adder tree compared to conventional O-GEHL. Results are shown for O-GEHL-1/1, O-GEHL-2/1 and O-GEHL-3/1.

from the summation process and the sharing group size is equal to s. For example O-GEHL-3/2 refers to a low-power O-GEHL where the lower three bits of each counter are excluded from the summation and two table entries share their lower three bits. Also O-GEHL-3/1 refers to a low-power O-GEHL where the lower three bits of each counter are excluded from the summation but there is no lower bit sharing, i.e., group size is one.

6.1 Timing

Figure 9 reports time (in nanoseconds) and the number of cycles required to compute the predictor computation result. We report results for the original O-GEHL predictor and three different optimized ones. As reported, the original predictor takes 5 clock cycles to compute the result. By eliminating one bit from the computation process no clock cycle is saved. While our study shows that by removing two or three bits one and two clock cycles can be saved respectively, in this study we take a conservative approach and assume that only one clock cycle is saved after eliminating two or three lower bits.

Table 2 reports predictor's table access time for four different O-GEHL predictors. Since the optimized predictor accesses only the LOB tables at the prediction time, access time is slightly lower.

6.2 Power Dissipation

Figure 10 reports the reduction in both leakage and dynamic energy consumption for the predictor's adder tree. Results are obtained by gate level synthesis of the circuit. Eliminating one bit saves up to 41% of the dynamic energy

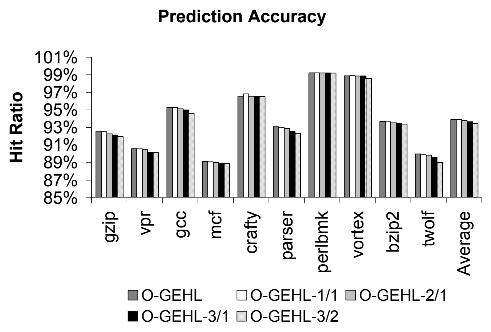

Figure 11: Prediction accuracy for the conventional O-GEHL predictor and four optimized versions, O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2. The accuracy loss is negligible.

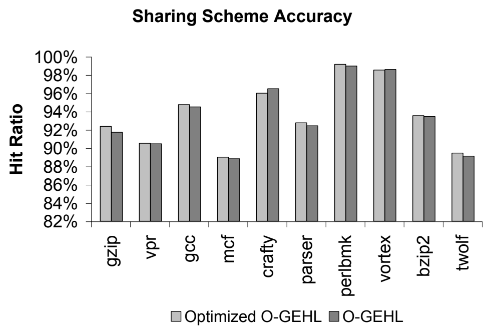

Figure 12: Accuracy for an optimized O-GEHL predictor (sharing groups of size two for three LOBs) and a conventional O-GEHL using the same real-estate budget.

and 44% of the static energy consumption. Energy reduction is higher when the number of eliminated bits increases. This is the result of exploiting smaller adders.

Table 2 reports the power dissipated while accessing HOB and LOB tables. At the prediction time, only LOB tables are accessed whereas at the update time, both sets are accessed. Since the same decoder is used for each pair of tables, accessing two tables at the update time does not impose any noticeable power overhead.

6.3 Prediction Accuracy

As we use fewer bits for making prediction, we can potentially harm accuracy. To investigate this further in Figure 11 we compare prediction accuracy for five different predictors: The original O-GEHL predictor, O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2. On average prediction accuracy is decreased by 0.06%, 0.1%, 0.3% and 0.5% for O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2 respectively. Maximum accuracy loss is 1.1% for *twolf* under O-GEHL-3/2.

Sharing the LOBs would result in an overall smaller storage. To make better evaluation of our scheme possible, we also compare the accuracy of an optimized O-GEHL predictor (sharing groups of size two for three low order bits) to a conventional O-GEHL using the same real-estate budget (but not sharing the LOBs). As reported in Figure 12, the optimized O-GEHL predictor achieves higher accuracy across all applications except for *crafty.*

6.4 Performance

Figure 13 reports processor's overall performance compared to a processor using the original O-GEHL predictor. We report for four different processors using O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2 branch predictors. As reported on average we improve performance by 4.7%, 4.6%, 4.3% and 3.9% for O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2 respectively. Although the optimized O-GEHL predictors achieve slightly lower accuracy compared to the original one, the overall processor

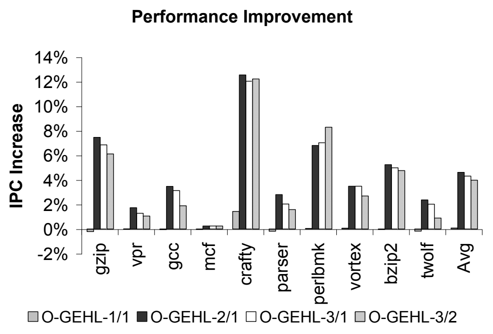

Figure 13: Performance improvement compared to a processor using the conventional O-GEHL predictor. Results are shown for processors using O-GEHL-1/1, O-GEHL-2/1, O-GEHL-3/1 and O-GEHL-3/2 predictors.

performance is higher. As explained earlier, this is the result of achieving faster prediction by eliminating LOBs.

7. RELATED WORK

Loh et al. [10] used the hysteresis bit bias to reduce table size in branch predictors using 2-bit saturating counters. They used data compression techniques and showed that hysteresis bit's entropy in 2-bit saturating counters is less than 1-bit.

Loh and Jimenez [11] introduced two optimization techniques for perceptron. They proposed a modulo path-history mechanism to decouple the branch outcome history length from the path length. They also suggested bias-based filtering exploiting the fact that neural predictors can easily track strongly biased branches whose frequencies are high.

Aasaraai and Baniasadi [1] proposed weight disabling to reduce branch prediction delay in the perceptron branch predictor. They dynamically reduce adder tree's delay by identifying and eliminating noneffective weights from computations.

Parikh et al. explored how branch prediction impacts processor power dissipation. They introduced banking to reduce the active portion of the predictor. They also introduced prediction probe detector (PPD) to identify when a cache line has no branches so that a lookup in the predictor buffer/BTB can be avoided [12].

Baniasadi and Moshovos introduced Branch Predictor Prediction (BPP) [2]. They stored information regarding the sub-predictors accessed by the most recent branch instructions executed and avoided accessing underlying structures. They also introduced Selective Predictor Access (SEPAS) [3] which selectively accessed a small filter to avoid unnecessary lookups or updates to the branch predictor.

Huang et al. used profiling to reduce branch predictor's power dissipation [7]. They disabled tables that do not improve accuracy and reduced BTB size for applications with low number of static branches.

Chang et al. [5] suggested identifying easily predictable branches and avoiding the pattern history table usage for such branches to reduce aliasing.

Eden and Mudge [6] introduced YAGS to reduce aliasing in the pattern history table.

Jimenez et al. [8] suggested using an overriding branch predictor to reduce delay. They used two predictions, one faster and one slower and less accurate consecutively.

Our work is different from all the above studies as it eliminates unnecessary and redundant computations for the O-GEHL predictor to reduce power. Unlike many previously suggested techniques, our optimizations do not come with any timing or energy overhead as we do not perform any extra computation or use any additional storage.

8. CONCLUSIONS

We studied the O-GEHL predictor and showed that by identifying and eliminating unnecessary computations, both static and dynamic power dissipation can be reduced considerably. We also showed that avoiding such computations reduces prediction latency, which results in better performance.

Moreover, we suggested exploiting sharing groups in which counters share their bits to reduce the necessary storage. Using sharing groups, we reduce predictor's table sizes and therefore power dissipation.

9. ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program and Canada Foundation for Innovation, New Opportunities Fund.

10. REFERENCES

[1] K. Aasaraai and A. Baniasadi. Low-power perceptron branch predictor. *Journal of Low Power Electronics*, 2(3):333:341, 2006.

[2] A. Baniasadi and A. Moshovos. Branch predictor prediction: A power-aware branch predictor for high-performance processors. In *Proceedings of 20th International Conference on Computer Design (ICCD 2002)*, pages 458–461, 2002.

[3] A. Baniasadi and A. Moshovos. Sepas: A highly accurate energy-efficient branch predictor. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pages 38–43, 2004.

[4] D. Burger and T. M. Austin. The simplescalar tool set version 2.0. Technical report, Technical Report 1342, Computer Sciences Department, University of Wisconsin, June 1997.

[5] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)*, pages 48–57, 1996.

[6] A. N. Eden and T. Mudge. The yags branch prediction scheme. In *Proceedings of 31st International Symposium on Microarchitecture*, 1998.

[7] M. C. Huang, D. Chaver, L. Pinuel, M. Prieto, and F. Tirado. Customizing the branch predictor to reduce complexity and energy consumption. In *Proceedings of 33rd International Symposium on Microarchitecture*, pages 12–25, 2003.

[8] D. A. Jimenez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd International Symposium on Microarchitecture (MICRO-33)*, pages 66–77, 2000.

[9] D. A. Jimenez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, pages 369–397, 2002.

[10] G. H. Loh, D. S. Henry, and A. Krishnamurthy. Exploiting bias in the hysteresis bit of 2-bit saturating counters in branch predictors. *Journal of Instruction Level Parallelism (JILP)*, 5:1–32, 2003.

[11] G. H. Loh and D. A. Jimenez. Reducing the power and complexity of path-based neural branch prediction. In *Proceedings of the 5th Workshop on Complexity Effective Design (WCED)*, pages 1–8, 2005.

[12] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 233–, 2002.

[13] A. Seznec. The o-gehl branch predictor. The 1st JILP Championship Branch Prediction Competition (CBP-1), in conjunction with MICRO-37, 2004.

[14] A. Seznec. Analysis of the o-geometric history length branch predictor. In *32nd Annual International Symposium on Computer Architecture*, 2005.

[15] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[16] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. Technical report, HP Laboratories Palo Alto, 2006.