Choice Predictor for Free

Mongkol Ekpanyapong, Pinar Korkmaz, and Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
{pop, korkmazp, leehs}@ece.gatech.edu

Abstract

Reducing energy consumption has become the first priority in designing microprocessors for all market segments including embedded, mobile, and high performance processors. The trend of state-of-the-art branch predictor designs such as a hybrid predictor continues to feature more and larger prediction tables, thereby exacerbating the energy consumption. In this paper, we present two novel profile-guided static prediction techniques— Static Correlation Choice (SCC) prediction and Static Choice (SC) prediction for alleviating the energy consumption without compromising performance. Using our techniques, the hardware choice predictor of a hybrid predictor can be completely eliminated from the processor and replaced with our off-line profiling schemes. Our simulation results show an average 40% power reduction compared to several hybrid predictors. In addition, an average 27% die area can be saved in the branch predictor hardware for other performance features.

I. INTRODUCTION

Advances in microelectronics technology and design tools for the past decade enable microprocessor designers to incorporate more complex features to achieve high speed computing. Many architectural techniques have been proposed and implemented to enhance the instruction level parallelism (ILP). However, there are many bottlenecks that obstruct a processor from achieving a high degree of ILP. Branch misprediction disrupting instruction supply poses one of the major ILP limitations. Whenever a branch misprediction occurs in superscalar and/or superpipelined machines, it results in pipeline flushing and refilling and a large number of instructions is discarded, thereby reducing effective ILP dramatically. As a result, microprocessor architects and researchers continue to contrive more complicated branch predictors aiming at reducing branch misprediction rates.

Branch prediction mechanisms can be classified into two categories: static branch prediction and dynamic branch prediction. Static branch prediction techniques [1, 5, 14]make prediction at compile-time. Such prediction schemes, mainly based on instruction types or profiling information, work well for easy-to-predict branches such as while or for-loop branches. Since the static branch prediction completely relies on information available at compile-time, it does not take runtime dynamic branch behavior into account. On the other hand, dynamic branch prediction techniques [10, 13] employ dedicated hardware to track dynamic branch behavior during execution. The hybrid branch predictor [10], one flavor of the dynamic branch predictors, increases the prediction rate by combining the advantages demonstrated by different branch predictors. In the implementation of a hybrid branch predictor, a *choice predictor* is used to determine which branch predictor should be favored every time after a particular branch instruction is fetched. Introducing a choice predictor, however, results in larger die area and additional power dissipation. Furthermore, updating other branch predictors that are not involved in a prediction draws unnecessary power consumption if the prediction can be done at compile-time. Given the program profile information, a static choice prediction can be made by identifying the suitable branch predictor for each branch instruction. For example, for a steady branch history pattern such as 000000 or 1010101, the compiler will favor the local branch predictor. On the other hand, for a local branch history pattern of 01011011101 and global branch history pattern of 00**1**1100**1**11000**1**11001 (**boldface** numbers correspond to the branch history of this target branch) it will favor the global predictor over the local predictor, because the global pattern history shows a repetition of the sequence 001 where 1 corresponds to the target branch.

The organization of this paper is as follows. Section II describes related work. Section III is devoted to our schemes. Section IV presents our experimental framework. Experimental results are presented in Section V. Finally the last section presents our conclusions.

II. RELATED WORK

There exists an extensive amount of work on branch prediction due to its impact on high performance processor designs. Different branch prediction schemes have been proposed in order to reduce the misprediction rate. These techniques mostly exploit the local behavior of each individual branch as well as the global branch correlation to improve prediction accuracy, either at static compile-time or dynamic runtime.

Static techniques include two major schemes— profile-guided and program-based schemes. Profile-guided schemes collect branch statistics by executing and profiling the application in advance. The compiler then analyzes the application again using these statistics as a guide and regenerates an optimized binary code. Program-based schemes tackle branch prediction problems at either source code level, assembly level, or executable file level without any profiling information. One early study on using profile-guided branch prediction was done by Fisher and Freundenberger [5] , in which they observed that profile-guided methods work quite effectively for conditional branches as most of these branches probability are highly biased to one direction and this direction almost remains the same across different runs of the program. Ball and Larus [1] later studied a program-based branch prediction method by applying simple and combined heuristics to program analysis at static compilation time and then generating static branch predictions.

One important characteristics of branch prediction is that a branch can either exhibit self-correlation or can be correlated with other branches. In [14], Young and Smith proposed a static correlated branch prediction scheme, in which they use a technique called path profiling to find the correlated paths. After identifying all the correlated paths, the technique either duplicates or discriminates the paths depending on the type of correlation. Due to path duplication, their technique increases the code size while reducing misprediction rate measured by static per branch prediction.

In spite of all the hardware savings, static branch prediction cannot be applied to all the branches in a program since some branches can demonstrate very dynamic behavior and will not be strongly biased to one direction or another in their lifetime. Therefore, most of the branch prediction researchers focus on dynamic prediction mechanisms. Dynamic branch predictors make predictions based on runtime branch direction history. Yeh and Patt [13] introduced the concept of two-level adaptive prediction that maintains a first level N-bit branch history register (BHR) and its corresponding 2^N entry pattern history table (PHT) as a second level in the processor for making predictions. The BHR stores the outcomes of the N most recently committed branches used to index into the PHT in which each entry contains a 2-bit saturating up-down counter. They studied both local and global prediction schemes. Local prediction schemes keep the local history of individual branches while global prediction schemes store the global direction history of a number of branches equal to the history register size.

McFarling [10] pioneered the idea of hybrid branch prediction that uses a meta-predictor to select a prediction from two different branch predictors. The two branch predictors studied in his paper were bimodal and gshare branch predictors. Bimodal branch predictor is an array of 2-bit saturating up-down counters indexed by the low order address bits of the program counter (PC). The gshare predictor, which was also christened by McFarling in the same paper is a two-level predictor that exclusive-ORs the global branch history and the branch PC address to reduce destructive aliasing among different branches sharing the same global history pattern. The choice predictor, containing an array of saturating counters, is updated to reward the predictor generating correct prediction.

The Alpha 21264 processor implemented a hybrid branch predictor called *tournament branch predictor* [8] which features three predictors including a local predictor, a global predictor, and a choice predictor. The local predictor, one variation of a 2-level predictor, consists of a 1024-entry 10-bit local history table and its corresponding 1024x3 bits prediction table. The global predictor, also a 2-level predictor, provides a 12-bit global history register and its corresponding 4096x2 bits prediction table. The choice predictor, yet another 2-level predictor, uses the same global history register to index its own 4096x2 prediction table. The 2-bit saturating counter of the choice predictor is incremented for one predictor (e.g. global) and decremented for another (e.g. local) whenever one predictor makes the correct prediction and the other does not. The counter is updated to reward the one making correct prediction.

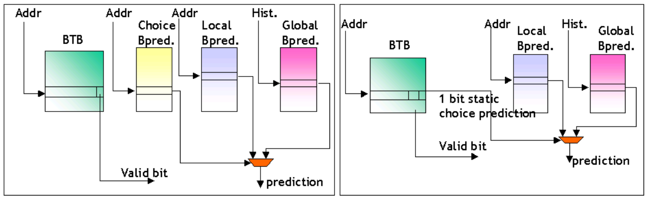

Figure 1: Branch prediction lookup schemes.

Chang et al. [3] studied branch classification. Their classification model groups branches based on profile data. They also proposed a hybrid branch predictor which takes the advantages of both static and dynamic predictors. Using the profile data, they perform static prediction for those branches that strongly bias to one direction in their lifetime. Their work is analogous to ours in the sense that we both employ static and dynamic branch prediction method. Comparison and simulation data will be presented and discussed in Section V.

Another recent work presented by Grunwald et al. in [6] also adopts static prediction for a hybrid predictor. Although a large experimental data were presented, there contains no details about their algorithms with respect to how they derive the choice prediction directions at static time. In addition, they compared their static prediction scheme with only McFarling hybrid prediction scheme, while we compare our technique against several other hybrid branch predictors and evaluate the impact to power and die area.

III. METHODOLOGY

Utilizing the local and global history correlation for branch prediction has become the major aspect of a high performance processor design. Given profiling is now a widely accepted technology for code optimization, in particular for static architectures such as EPIC, we propose a new methodology that utilizes profiling data from prior executions, classifies branches according to the types of correlation exhibited (e.g. local or global), and then decides which prediction result to use. During profile-guided recompilation, these decisions are embedded in the corresponding branch instructions as static choice predictions. For example, the branch hint completer provided in the Intel Itanium ISA [4] can be encoded with such information.

The basic branch prediction lookup scheme for a hybrid branch predictor with a hardware choice predictor and our scheme with static choice prediction are illustrated in Figure 1. In our scheme, the static choice prediction is inserted as an extra bit in each branch target buffer (BTB) entry. For each branch predicted, both the local and global predictors are accessed and the prediction implied by the static choice prediction bit in the indexed BTB entry is chosen. The critical path for this branch predictor is not lengthened with such a mechanism, hence no impact to clock speed. Using this bit to clock gate each branch predictor might lead to further power reduction, however, it is out of scope of this paper.

Most of the hybrid branch predictors with a dynamic choice predictor [8, 10] update all the branch prediction components for each branch access. This is because that, in a dynamic choice predictor, the choice predictor is updated dynamically depending on the prediction results of both branch predictors and for the further accesses to the same branch address there is uncertainty about which branch predictor will be used, hence updating both of them will result in more accuracy. In our model, we update only the branch predictor whose prediction is used, since every branch is already assigned to one of the predictors and updating only the assigned branch predictor is necessary. In our case, updating both branch predictors would not only consume more power but also increase the aliasing possibility.

In the following sections, we propose and evaluate two enabling techniques — *Static Correlation Choice (SCC)* prediction and *Static Choice (SC)* prediction from power and performance standpoints.

1 SCC model

In the SCC model, we profile and collect branch history information for each branch. We apply this technique to a hybrid branch predictor that consists of a local bimodal branch predictor [12] and a global two-level branch predictor [13]. The algorithm for the SCC model with the hybrid branch predictor is described in the following steps:

1. If a branch is biased to one direction either *taken* or *not taken* during its lifetime in execution, we favor its prediction made by the bimodal branch predictor. The bias metric is based on a pre-determined threshold value that represents the frequency of the direction of a branch (e.g. 90% in this study, this is based on our intuition that higher than 90% hit rate is acceptable).

2. To model the bimodal branch predictor, we count the total number of consecutive *taken*'s and consecutive *not taken*'s for each branch collected from profile execution. This count based on the local bimodal branch predictor is denoted by C_{LP}. For example, if the branch history of a particular branch is 111100000101010: the number of consecutive ones is 4-1 = 3 and number of consecutive zeros is 4, therefore, $C_{LP} = 3+4 = 7$.

3. To model the global branch predictor, we collect global history information for each branch on-the-fly during profile execution and compare it against all prior global histories collected for the same branch. If the last k bits of the new global history match the last k bits of any prior global history, then the new prediction is called to be within the same history group. There are 2^k possible groups in total. For each branch that is included in a group, we count the total number of consecutive *taken*'s and consecutive *not taken*'s. At the end of the profile run, we sum up the consecutive counts including *taken* and *not taken* for each history group and denote the value by C_{GP}. For example, assume we have four history groups (k=2) — 00, 01, 10 and 11 for a profile run. For a particular target branch after the profile execution, we have a branch history 101000001111 for the 00 group, 11111111110 for the 01 group, 1110 for the 10 group, and 1000000 for the 11 group. Then the summation for this global branch predictor, for this particular branch would be $C_{GP} = 7+9+2+5 = 22$. Note that the history does not include the direction of the current reference.

4. C_{LP} and C_{GP} values are collected after the profiling execution. The static choice prediction is made offline by comparing the values of C_{LP} and C_{GP}. The final choice, provided as a branch hint, as to which predictor to use for each branch is determined by favoring the larger value. In other words, if C_{LP} is greater than C_{GP}, the choice prediction uses the prediction made by the bimodal predictor otherwise the prediction of the global branch predictor is used.

SCC model basically targets McFarling's hybrid branch predictor. As aforementioned, McFarling's hybrid branch predictor consists of a bimodal local predictor and a gshare global predictor. The justification behind the calculation of C_{LP} (a metric for bimodal branch prediction) is that, for a bimodal predictor the more the branch result stays in state 00 (strongly not-taken) or 11 (strongly taken), the more stable the prediction will be. On the other hand, C_{GP} of a branch is the metric for the global branch prediction and its calculation is based on counting the number of occurrences of consecutive takens and not-takens (0's and 1's) for this branch for the possible number of different branch histories depending on the length of history. This is similar to the two-bit saturating counters which are chosen by the global history register in the gshare scheme.

2 SC model

In the SC model, static choice predictions completely rely on the result collected from the software-based choice predictor of an architecture simulator. During profiling simulation, we collect the information with respect to how many times the choice predictor is biased to the bimodal predictor versus the global branch predictor for each branch. The final static choice prediction is then based on the majority reported from the profiling simulation.

IV. EXPERIMENTAL FRAMEWORK

Our experimental framework is based on `sim-outorder`, a cycle-based architecture simulator from SimpleScalar toolkit version 3.0 [9]. We modified the simulator to (1) enable different hybrid branch predictors , (2) collect the profile information needed for the SCC and SC models, and (3) perform static choice branch prediction. Table 1 shows the parameters of our processor model. The SPEC CPU2000 integer benchmark suite [7] was used for our evaluation. All of the benchmark programs were compiled into Alpha AXP binaries with optimization level `-O3`. All the data presented in Section V was obtained through runs of

Table 1: Parameters of the processor model.

Execution Engine	Out-of-order
Fetch Width	8 instruction
Issue Width	8 instruction
ALU Units	4 units
Branch Target Buffer	4-way, 4096 sets
Register Update Unit	128 entries
Cache organization	4-way split I- and D-L1: 64 KB each 2 cycle hit latency 32 bytes line
	4-way L2(unified): 512 KB 16 cycle hit latency 64 bytes line
Memory latency	120 core cycles

one billion instructions. Since profiling is involved, the experiments were performed among *test*, *train* and *reference* profiling input sets whereas all the performance evaluation results come from *reference* input set. In other words, we collected different profiling results in order to analyze the impact of our proposed mechanisms based on different profiling input sets.

As our proposed technique gives us an opportunity to eliminate the choice predictor hardware, we also evaluate and quantify the power improvement due to the absence of a hardware choice predictor. The Wattch [2] toolkit, an architecture-level switching power model built on top of `sim-outorder`, was used for that purpose. We modified Wattch to enable clock-gating in different functional blocks of a branch predictor including the BTB, the local, global, and choice predictors, and return address stack.

V. EXPERIMENTAL RESULTS

This section presents our performance and power analysis. In the first experiment, we study the impact of our static models for choice prediction on performance, including branch prediction rate and speedup. The *train* input set in SPECint2000 benchmarks is used for collecting profile information, while the *reference* input set is used for performance evaluation. Results show that our prediction model performs as well as a hardware choice predictor and sometimes it even outperforms.

Figure 2 summarizes the branch prediction miss rates from different branch predictors for SPECint2000 benchmarks. For each benchmark program, experiments are conducted with a variety of branch prediction schmes. Among them are **gshare10**, **gshare11**, **gshare12**, **hybrid_g10**, **hybrid_g10+scc**, **hybrid_g10+sc**, **hybrid_g11+scc**, and **hybrid_g11+sc**. The **gshare10**, same as McFarling's gshare scheme [10], indexes a 1024-entry 2-bit counter array by exclusive-ORing the branch address and its corresponding 10-bit global history. Similarly, **gshare11** and **gshare12** perform the same algorithm by simply extending the sizes of their global history to 11 and 12 bits, thereby increasing their corresponding 2-bit counter arrays to 2048 and 4096 entries, respectively. The predictor, **hybrid_g10** uses a hybrid branch predictor approach similar to McFarling's combining branch predictor [10]. It consists of a bimodal predictor, a two-level predictor, and a choice (or meta) predictor each of them with a size of 1024x2 bits. The **hybrid_g10+sc** is the same as **hybrid_g10**, but replaces the hardware choice predictor with a profiling-based choice prediction mechanism using the SC model described in Section III. Likewise, **hybrid_g10+scc** uses the SCC model for choice predictions. Predictors **hybrid_g11+scc** and **hybrid_g11+sc** are extended versions of the **hybrid_g10+scc** and **hybrid_g10+sc** models, respectively, as they increase the size of the two-level branch predictor to 2048x2 bits.

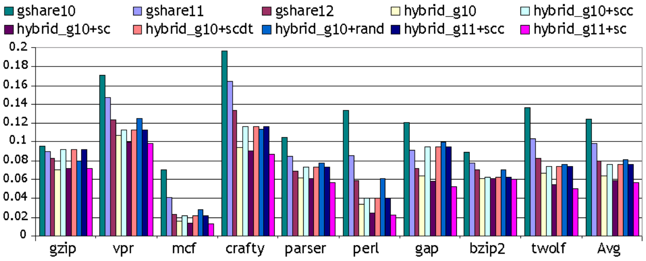

Figure 2: Miss rate with different branch predictors.

Moreover we also implement the prediction model proposed by Chang et al. [3] which we call SCDT model. In SCDT, profiling is used to classify branches into different groups based on dynamic taken rates and for each group the same branch predictor is used. If the dynamic taken rate of a branch is 0-5% or 95-100% then this branch is predicted using the bimodal predictor, otherwise it is predicted using gshare predictor. Comparing with SC, if there are a lot of branches that change their behavior dynamically, then SCC captures such behavior better than SCDT. For example, if the behaviour of a branch has k consecutive 0's and k consecutive 1's, a bimodal prediction is better, since it might reduce aliasing in gshare. By contrast SCDT will always use gshare. We also perform experiments using a random choice model which we call RAND model and it randomly selects a branch predictor statically. The **hybrid_g10+scdt** and **hybrid_g10+rand** results are based on the SCDT and RAND models respectively.

As shown in our simulation results in Figure 2, increasing the size of the global branch predictor alone does not perform as well as using a hybrid branch predictor. For example, the **gshare12** predictor consists of more prediction entries than the **hybrid_g10** branch predictor provides (area comparison is shown in Table 2), but none of the benchmarks shows the **gshare12** branch predictor outperforming the **hybrid_g10** branch predictor.

Also shown in Figure 2, instead of having a hardware choice predictor, we can achieve comparable prediction rates using a static off-line choice predictor. Our simulation results show that SCC does not perform as well as SC. This is because SC can accounts for aliasing in its model, and hence it is more accurate. The difference is less than 2% in branch miss prediction rate.

Comparing between SCC and SCDT, both schemes provide comparable results. This suggests that branches with varying behaviour, as explained earlier, rarely occur in

SPEC2000. Selecting branch predictors at random does not provide as good an average result as our SCC and SC.

We also show that instead of having a hardware hybrid choice predictor, we can employ a static choice prediction and increase the size of the global branch predictor. The **hybrid_g11+sc** model demonstrates the best results in branch prediction rate, and hence in performance for most of the benchmarks.

Figure 3 shows the normalized performance speedups due to various prediction schemes; the baseline in this figure is **gshare10**. The results show that speedup improves as prediction rates increase. We expect the increase will be more significant with deeper, and wider machine.

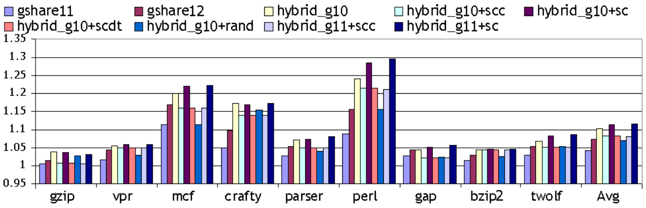

Figure 3: Normalized speedup with different branch predictors.

Previously, we explained that the purpose of our work is to decrease the area and power consumption of a branch predictor, while not compromising the performance of the processor. To this end, we use Wattch [2] to collect the power statistics of the branch predictor and the other units of the processor. In Wattch, the calculation of the power consumed due to switching is access based. That is, for every block (such as BTB, branch predictor, i-cache and d-cache) the switching power consumed per access is calculated, and this number is multiplied by the total number of accesses to that block. We assume a clock gating scheme in which a block that is not in use consumes an amount of static power is equal to 10% of its switching power. We also want to mention that we examined the effect of our branch prediction schemes on the power consumption of the branch direction predictor, and we claim improvements on the power consumption of the branch direction predictor.

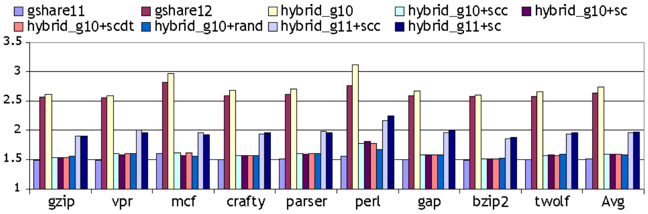

Figure 4: Normalized power consumption of different branch predictors.

Figure 4 shows the normalized power consumption values for different branch predictors, relative to the power consumption of **gshare10**. From this Figure and Figure 3, we can tell that for nearly all the benchmarks, **hybrid_g10+sc** yields the best processor performance for little branch prediction power. We can use Figures 3 and 4 as guides in a design space exploration framework, where the power budget of the branch predictor is limited, and a specific performance constraint has to be satisfied. For example, the results in Figure 4 show that the remove of the choice predictor in **hybrid_g10** can reduce the power consumption to a level comparable to that of **gshare11**. Similarly Figure 3 shows that **hybrid_g10+sc** outperforms **gshare11**, for all the benchmarks. Hence we can deduce that using **hybrid_g10+sc** is more advantageous in terms of both the power dissipation and performance.

Next, we study the impact of profiling on the training input set of our SC and SCC training. We aim to show how our models SCC and SC are affected as a result of various training data. We use three different input sets for profiling: *test*, *train*, and *reference*. The results show little impact on the branch prediction outcomes. The results are detailed in Figure 5 where the baseline is again **gshare10**. Figure 5 shows that SCC is less sensitive to profile information than SC. This is because SC incorporates aliasing information in its model. Let us consider the Control Flow Graph (CFG), which is shown in Figure 6. Assume that branches a and c point to the same location in global branch predictor and also are predicted accurately by a global branch predictor if there is no destructive aliasing. If branches a and c destructively interfere with each other, this results in profiling say that loop A-C is called more frequently than loop B-C hence static choice predictor will assign both branches a and c to local branch predictor. However on the running input set, if loop C-A runs more often than loop B-A then assigning both a and c to local branch predictor can reduce branch prediction accuracy. Figure 5 also shows that if profile information has the same behavior as the real input set, static choice predictor can outperform hardware choice predictor in most benchmarks.

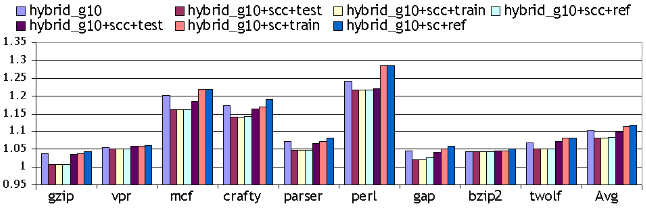

Figure 5: Normalized speedup on different profiling input sets.

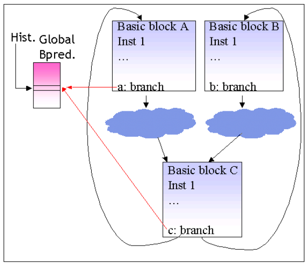

Figure 6: CFG example showing aliasing impact.

We then perform experiments using different hybrid branch predictors to show that SC and SCC are equally compatible with different kinds of hybrid branch predictors. In this set of experiments, **gshare10** is our chosen baseline. The results are shown in Figure 7. Note that **hybrid_PAg** is a hybrid branch predictor similar to the one used in Alpha 21264 processor. It consists of a two-level local predictor with a local history table size of 1024x10 bits, local predictor size of 1024x2 bit and with global and choice predictors of size 1024x2 bit. **hybrid_GAp** stands for a hybrid branch predictor with a 1024x2 bit bimodal predictor and four of 1024x2 bit counters instead of one such counter as in **hybrid_g10**.

Since SCC is not intended to target **hybrid_PAg**, i.e it cannot exploit full advantages from local branch predictor in **hybrid_PAg**, we exclude the result of the SCC on **hybrid_PAg**. For example, if we have local history pattern of 1010101010, C_{LP} is 0 and SC will not choose local branch predictor but local predictor in **hybrid_PAg** can predict this pattern accurately.

Results shown in Figure 7 also indicate that SC works well with **hybrid_PAg**.

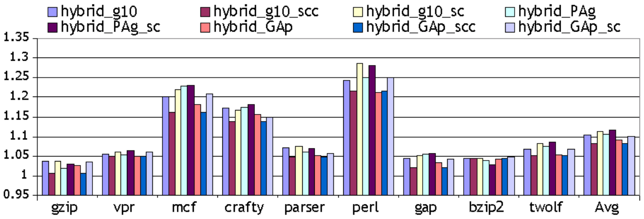

Figure 7: Normalized speedup on different hybrid branch predictors.

We now report the power consumption of different branch prediction strategy. Figure 8 shows the normalized power consumption for different hybrid predictors relative to **gshare10**. In this figure, we observe that for **hybrid_g10**, and **hybrid_GAp**, using SC and SCC methods bring an improvement of 42% on average. The average improvement for **hybrid_PAg** is around 37%. The power consumption in **hybrid_GAp** is not too high comparing with **hybrid_g10** since clock gating is applied to unused predictors.

These results allow the possibility of replacing the hardware choice predictor with our schemes, and reclaim in its corresponding die area. Assuming a static memory array area model, as described in [11], for the branch predictor, the area can be quantified as followings:

$$area_{static\_memory} = 0.6\ (size_w + 6)\ (line_b + 6)\ rbe \qquad (1)$$

where $size_w$ is the number of words, $line_b$ is the number of bits and rbe is an area unit of a register cell. The two +6 terms approximate the overhead for the decoder logic and sense amplifiers. Based on equation 1, we derived the normalized areas of different branch predictors relative to **gshare10** in Table 2. Note that the die area saved by using our profile-guided SCC and SC schemes for **hybrid_g10** predictor is 33.18%. The saving is less for other predictors because these predictors are comprised of more complicated local and global predictors which consume a lot of area. One interesting result in the table shows that the area of **hybrid_GAp+sc/scc** is smaller than the area of **hybrid_g10**. This is due to the fact that fewer decoders are needed for **hybrid_GAp+sc/scc** compared to **hybrid_g10**. The four 1024x2 bit tables in **hybrid_GAp** share the same decoder, hence we need only one 10x1024 decoder and one 2x4 decoder for **hybrid_GAp**, while **hybrid_g10** needs three separate 10x1024 decoders (one for each predictor).

Table 2: Normalized area of hybrid branch predictors.

Branch predictor	Normalized area
gshare11	1.9822
gshare12	4.806
hybrid_g10	2.973
hybrid_g10+scc/sc	1.986
hybrid_g11	3.955
hybrid_g11+scc/sc	2.968
hybrid_PAg	4.946
hybrid_PAg+sc	3.959
hybrid_GAp	3.713
hybrid_GAp+sc/scc	2.726

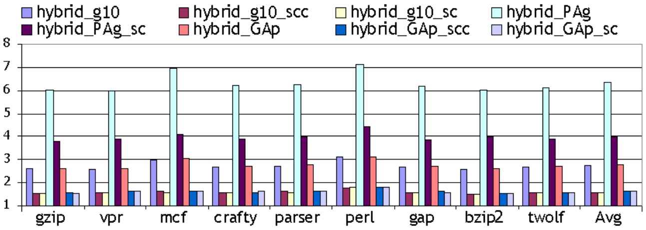

Figure 8: Normalized power consumption of different hybrid branch predictors.

VI. CONCLUSIONS

In this paper, we study two profile-guided techniques: *Static Correlation Choice* and *Static Choice*, for performing offline static choice predictions. Our work offers the possibility of eliminating the hardware choice predictor while achieving comparable performance results. In other words, the branch prediction rates attained by dynamic choice predictors can also be achieved using the two proposed models, thus resulting in similar performance. The studies we carried out using different input data further indicate that the SC and SCC techniques are largely insensitive to profiling data. By using our techniques, we can reduct the power dissipation of the branch predictor by 40% on average. Moreover, an average saving of 27% in die area can be saved.

VII. REFERENCES

[1] T. Ball and J. R. Larus. Branch Prediction for Free. In *PLDI-6*, 1993.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. *ISCA-27*, June 2000.

[3] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. N. Patt. Branch Classification: a New Mechanism for Improving Branch Predictor Performance. *International Journal of Parallel Programming*, Vol. 24, No. 2:133–158, 1999.

[4] Intel Corporation. IA-64 Application Developer's Architecture Guide. Intel Literature Centers, 1999.

[5] J. A. Fisher and S. M. Freudenberger. Predicting Conditional Branch Directions From Previous Runs of a Program. In *ASPLOS-5*, pages 85–95, 1992.

[6] D. Grunwald, D. Lindsay, and B. Zorn. Static Methods in Hybrid Branch Prediction. In *PACT'98*, 1998.

[7] John L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Micro*, July 2000.

[8] R. E. Kessler. The ALPHA 21264 Microprocessor. *IEEE Micro*, March/April 1999.

[9] SimpleScalar LLC. SimpleScalar Toolkit version 3.0. http://www.simplescalar.com.

[10] S. McFarling. Combining Branch Predictors. Technical Report TN-36, Compaq Western Research Lab, 1993.

[11] J. M. Mulder, N. T. Quach, and M. J. Flynn. An Area Model for On-Chip Memories and its Application. *IEEE JSSC*, Vol. 26 No. 2, February 1991.

[12] J. E. Smith. A Study of Branch Prediction Strategies. In *ISCA-8*, 1981.

[13] T.-Y. Yeh and Y. N. Patt. Two-Level Adaptive Training Branch Prediction. In *MICRO-24*, 1991.

[14] C. Young and M. D. Smith. Static Correlated Branch Prediction. *ACM TOPLAS*, 1999.