SABA: a Zero Timing Overhead Power-Aware BTB for High-Performance Processors

Kaveh Jokar Deris Amirali Baniasadi
Electrical and Computer Engineering Department
University of Victoria, Victoria, Canada
{kaveh, amirali}@ece.uvic.ca

Abstract

Modern high-performance processors access the branch target buffer (BTB) every cycle to speculate branch target addresses. This aggressive approach improves performance as it results in early identification of target addresses. However, unfortunately, such accesses, quite often, are unnecessary as there is no control flow instruction among those fetched.

In this work we introduce Speculative BTB Access (SABA) to address this design inefficiency. SABA relies on a simple power efficient structure, referred to as the SABA-filter, to identify cycles where there is no control flow instruction among those fetched, at least one cycle in advance. By identifying such cycles and eliminating unnecessary BTB accesses we reduce BTB's power dissipation (and therefore power density).

SABA comes with zero timing overhead as it makes decisions regarding future cycles and therefore does not impact critical path delay. Our study shows that, by using SABA, it is possible to eliminate more than half of the unnecessary BTB accesses while paying a very low performance cost (average: 0.7%). We also study how variations in SABA-filter configuration, branch predictor configuration and processor execution bandwidth impact power savings and performance slowdown for a SABA-enhanced processor.

1 Introduction

Power optimization techniques often aim at optimizing processor units consuming large shares of total power dissipation (*e.g.,* issue logic [15]). However, in recent years there have been studies focusing on units with smaller share of total power but with higher power density. Such units often create hot spots and can cause permanent faults (*e.g.,* register renaming unit[16]).

This work introduces power optimization techniques for the branch target buffer (BTB). We target the BTB due to the following: First, conventional high-performance designs access the BTB aggressively and frequently. This requires using multi-ported structures and can result in high temperatures (possibly resulting in faults) and higher leakage [17, 18]. Second, BTB is an energy hungry structure and consumes a considerable share of the branch predictor unit's energy budget.

Exploiting the BTB improves performance by making early identification of target addresses (for control flow instructions) possible. To achieve this, at fetch, modern processors access BTB for all instructions to find the branch/jump target address as soon as possible. This aggressive approach helps performance, but it is inefficient from the energy point of view. This is due to the fact that control flow instructions (conditional or unconditional branches) account for less than 25% of the fetched instructions. Therefore, many BTB accesses consume energy and produce heat but do not contribute to performance.

We introduce ***S**peculative **B**TB **A**ccess* (***SABA***), as a power-aware method to identify and eliminate unnecessary BTB accesses. SABA exploits fetch cycles' history to detect cycles where BTB access does not contribute to performance. SABA reduces power dissipation (and therefore power density) by avoiding accessing the BTB during such occasions. To avoid any timing overhead, identifying cycles with no control flow instructions is done at least one cycle in advance. Therefore, SABA comes with zero timing overhead and does not increase front-end latency.

Using a set of the SPEC CPU2000 benchmarks and WATTCH power models [3] we find that with an average performance penalty of 0.7%, SABA eliminates more than half of the unnecessary BTB accesses.

The rest of this paper is organized as follows. In Section 2 we discuss our motivation. In Section 3 we introduce SABA and present the details. In Section 4 we discuss methodology and results. In Section 5 we review related work. Finally, in Section 6 we offer our concluding remarks.

2 Motivation

Modern processors fetch instructions aggressively. This is to keep the instruction queue full and let the pipeline execute as many instructions as possible in parallel. The number of fetched instructions varies during program runtime depending on program behavior and processor configuration.

A previous study shows that the average gap between control flow instructions can be as high as 12 instructions [2]. Nonetheless, in order to avoid extra delay, modern processors access BTB for all instructions. This aggressive approach, quite often, results in power dissipation and excessive heat without contributing to performance. This is due to the fact that many of these accesses are not necessary since there are no control flow instructions among the instructions fetched during the cycle. We refer to such cycles as *control-free cycles* or *CFCs*. Alternatively we refer to cycles where there is at least one control flow instruction among the instructions fetched as *control-dependent cycles* or *CDCs*. The energy consumed by unnecessary BTB accesses depends on how often CFCs occur.

Figure 1 reports CFC frequency for processors with different execution bandwidths and for the subset of SPEC CPU2000 used in this work. As presented, CFC frequency is high for different execution bandwidths.

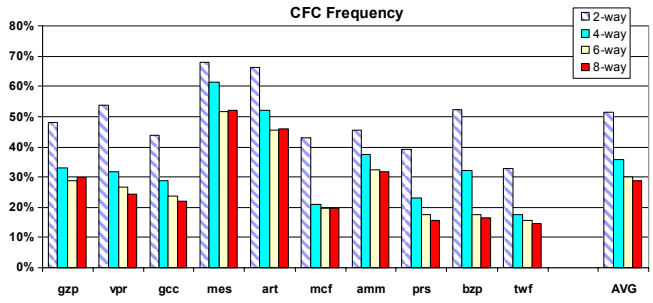

Fig. 1. *CFC frequency for different applications and for different execution bandwidths*

We conclude from Figure 1 that there is an opportunity to reduce BTB power dissipation by identifying CFCs and avoiding accessing BTB for them.

An alternative approach to avoid BTB accesses during CFCs is to use a pre-decoder to identify control flow instructions. This, unfortunately, can result in extra delay. Previous work shows that increasing predictor latency significantly erodes performance[1]. Therefore, any design choice resulting in increasing branch predictor delay, is probably not a good choice [4]. To address this problem, SABA avoids increasing prediction latency by making decisions regarding BTB usage in future cycles. SABA aims at identifying CFCs at least one cycle in advance and therefore does not impact critical path delay. Hence, it does not increase overall prediction latency.

3 SABA Architecture

SABA aims at predicting CFCs at least one cycle before they occur. Accurate CFC prediction enables us to prevent unnecessary BTB accesses. Figure 2 shows SABA's block diagram including the SABA-filter used to eliminate unnecessary BTB accesses.

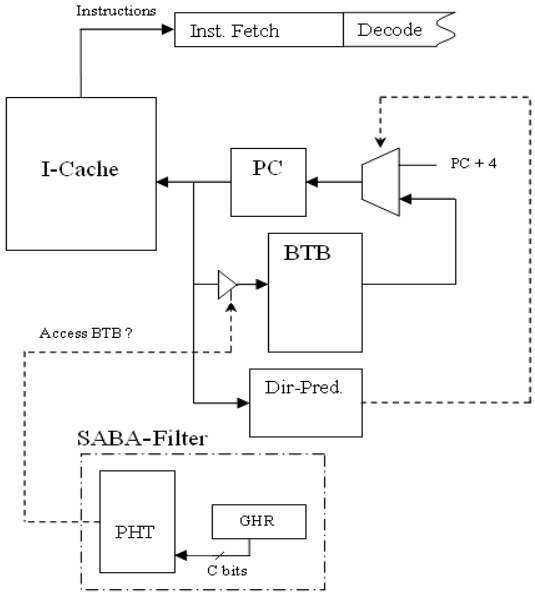

Fig. 2. *SABA architecture*

SABA consists of a small global history shift register (GHR). This register represents the history of CDCs and CFCs. Throughout this paper we refer to the length of this register as *GHR-size*. The bigger the GHR-size is, the more we know about past history.

We use zeros and ones to represent CFCs and CDCs in the GHR respectively. We use the GHR value to access an entry in the PHT (Prediction History Table). Every PHT-entry contains an *n-bit* saturating counter with a maximum value of *Sat*. We increment the counter when a CFC is detected. We decrement the counter if the fetch cycle is a CDC.

At fetch, we use the recent fetch history to probe the counters. We use the counter value to speculate whether the

[1] The trend is exemplified by the AMD processor family. The AMD-k6 used a highly accurate 8k-entry GAs branch predictor. The newer AMD Althon uses a less accurate but faster 2k-entry branch predictor [5]. Accordingly, AMD has sacrificed higher predictor accuracy (and therefore energy efficiency [2]) for faster clock rate.

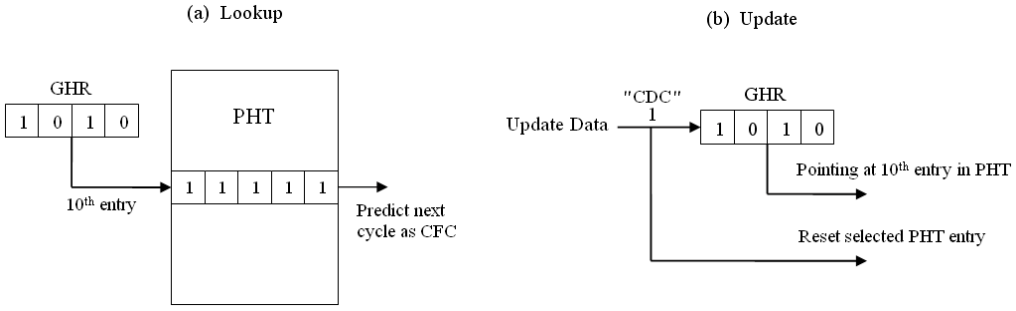

Fig. 3. *(a) Control free cycle predictor lookup.* *(b) Control free cycle predictor update.*

next cycle is a CFC. A saturated counter in PHT indicates that the next cycle is probably a CFC (see Figure 3(a)).

We update the SABA-filter every cycle and as soon as we know whether there has been a control flow instruction among those fetched (see Figure 3(b)). This is possible at decode and after instructions are decoded. One possible way to correct CFC mispredictions, upon finding a control flow instruction among the decoded instructions, is to update the SABA-filter entry associated with the time the instruction was fetched. This can be done by approximating this time by shifting the most recent global history by 3 bits (assuming that fetch latency is 3 cycles).

TABLE 1: Simulated processor configuration

Processor Core	
Instruction Window	RUU= 128; LSQ=32
Fetch Width	Upto 8 Instruction per cycle;
Issue Width	4 Instruction per cycle; 4 integer, 4 FP
Miss pred. Penalty	6 cycle
Fetch Buffer	64 entries
Functional Units	4 Int ALU, 4 Int mult/div 4 FP ALU, 4 FP mlt/div, 2 mem prt
Memory Hierarchy	
L1 D-cache Size	64 KB, 4-way, 32B blocks, wr bk
L1 I-cache Size	64 KB, 4-way, 32B blocks, wr bk
L1 latency	3 cycle
L2	Unified, 512KB, 4-way LRU 64B blocks, 16-cycle Latency, wr bk
Memory latency	80 cycles
D-TLB/I-TLB Size	128/64-entry, fully assoc., 30-cycle miss
Branch Prediction	
BTB	512-entry, 4-way
Direction Predictor	Combined predictor, 32K entries
Return-address-stack	64-entry

4 Methodology and Results

In this section, we present our methodology and evaluate SABA. We report how variations in SABA-filter configuration, branch predictor configuration, and processor execution bandwidth impact performance, BTB access frequency and power dissipation reduction. We used programs from the SPEC CPU2000 suite compiled for the MIPS-like architecture used by the Simplescalar v3.0 simulation tool set [6]. We detail the base processor model in table 1.

To evaluate our technique, we used a modified version of Wattch toolset [3]. We report both *accuracy* (*i.e.*, how often we accurately predict a CFC) and coverage (*i.e.,* what percentage of CFCs are accurately identified) for a wide range of filter sizes (from 2 to 64 PHT entry).

Table 2: Energy consumed per access for branch predictor units and the SABA-filter.

Modeled Unit	Size	Percentage
SABA	32 x 3	< 0.25%
BTB	512 x 4-way	60.2%
Comb. dir. Predictor	32k x 3 (LHT) 32k x 2 (GHT) 32k x 2 (sel.)	33.91%
RAS	64 (entry)	5.64%

Provided that a sufficient number of CFCs are accurately identified, SABA has the potential for reducing BTB energy consumption. However, it introduces extra energy overhead and can increase overall energy if the overhead exceeds savings. To estimate the relevant process parameters, including the power overhead associated with SABA, we used the process scaling methodology developed for CACTI [7] that is incorporated in WATTCH. We report relative energy consumed per access by a typical (*i.e.,* 32-entry) SABA filter and other structures used in the branch predictor in Table 2. We report energy consumption for each structure compared to the energy consumed by a branch predictor equipped with a 32k-entry hybrid predictor, a 4-way, 512-entry BTB and a 64-entry RAS. As reported the overhead of the typical SABA filter is far less than the energy consumed by the BTB. Nonetheless, we take into account this overhead in our study.

4.1 Accuracy and Coverage

In Figure 4 we report average prediction accuracy and coverage for SABA for the SPEC'2K applications studied

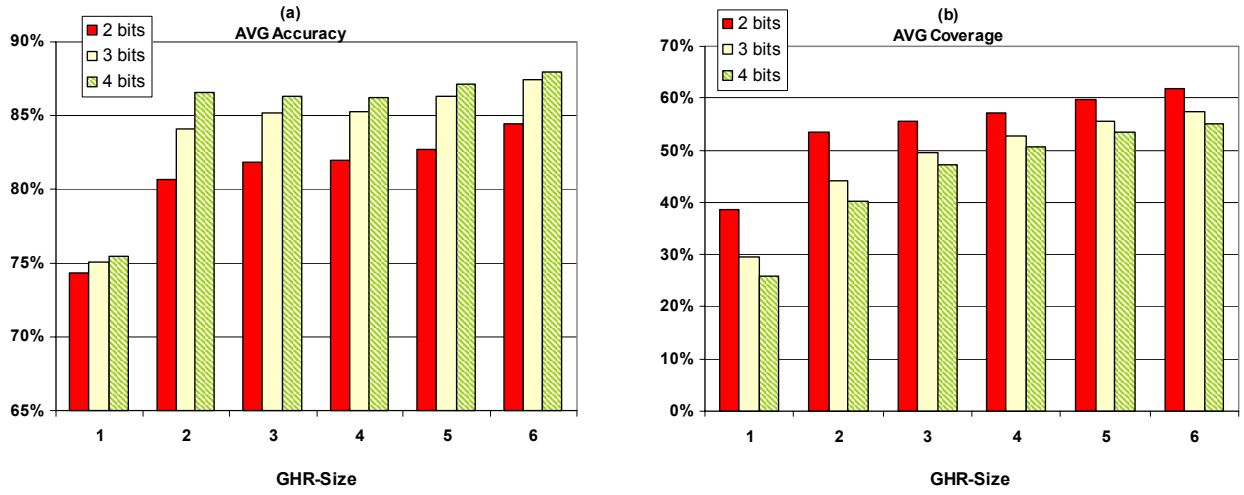

Fig. 4. *Average accuracy and coverage achieved for different SABA-filter configurations and for the SPEC2k benchmarks studied here. For each GHR-size (x-axis) bars from left to right report for 2-, 3- and 4-bit saturating counters.*

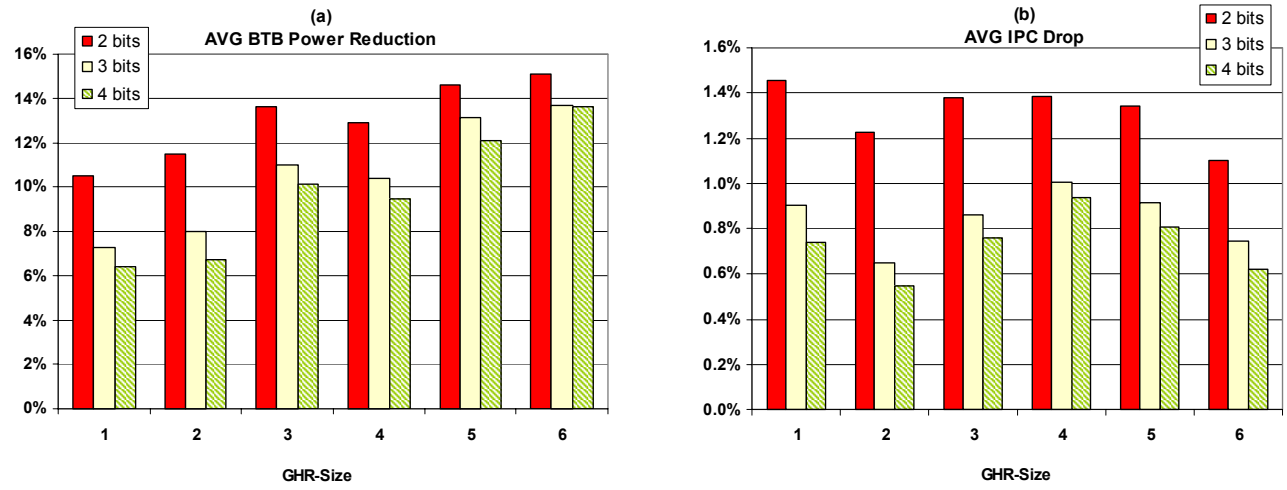

Fig. 5. *Average BTB power reduction and performance loss for different SABA-filter configurations and for the SPEC2k benchmarks studied here. For each GHR-size (x-axis) bars from left to right report for 2-, 3- and 4-bit saturating counters for different configuration of suggested predictor.*

in this work. In 4(a) we report average accuracy for different GHR-sizes (*i.e.*, 1 to 6) and saturating counters (*i.e.,* 2, 3 and 4-bit counters). As reported we predict CFC cycles with an accuracy varying from 74% to 88% for different SABA-filter configurations.

Note that mistaking a CFC for a CDC does not harm performance as it only results in unnecessary BTB access. However, mispredicting a CDC as a CFC, can result in late target address identification and comes with an extra cycle penalty in our study.

In 4(b) we report what percentage of CFCs is identified. We identify between 25% and 61% of CFCs for different SABA configurations.

We conclude from Figure 4 that with a fixed GHR-size, increasing saturation threshold will generally increase accuracy but reduce coverage. With a fixed counter size, increasing the GHR-size does not always improve accuracy or coverage. A small GHR-size results in recording very little history, whereas larger GHR-sizes may result in mapping small repeating patterns to different SABA-filter entries which may result in a longer SABA learning period.

4.2 Power and Performance

We report BTB power reduction and processor performance loss in Figure 5. We use the same set of parameters used in Figure 4. BTB power reduction varies between 6 and 15% for different SABA-filter configurations. Average performance loss is below 1.5% for all configurations. For a fixed GHR-size larger saturation threshold comes with lower performance loss but at the expense of reduced power savings. For a fixed counter size, increasing the GHR-size almost always improves BTB power reduction. The only exception is when we increase the size from three to four. We

conclude from Figure 5 that the best GHR-size or history length to consider in predicting application behavior is different from one application to another. This is consistent with other studies [14].

4.3 Sensitivity Analysis

In this section we report SABA's BTB power reduction and performance cost for different branch predictors and for processors with different execution bandwidths.

4.3.1 Branch Predictor Configuration

To investigate how SABA impacts power and performance for different branch predictor configurations, we study the following combined branch predictor [8] configurations: a) 2k-entry gshare, bimodal & selector BTB: 256-entry, 4-way b) 4k-entry gshare, bimodal & selector BTB: 512-entry, 4-way c) 8k-entry gshare, bimodal & selector BTB: 512-entry, 4-way and d) 32k-entry gshare, bimodal & selector BTB: 512-entry, 4-way.

The gshare predictor used in all predictors uses 6-bit history.

We assume a GHR-size of 6 bits and 3-bit saturating counters. This requires using a 64-entry PHT where each entry is a 3-bit counter (192 bits total). We select this configuration as it results in considerable power reduction while maintaining performance within acceptable limits.

In Figure 6, we report performance and BTB power reduction for the four configurations. As reported variations in predictor and the BTB size have very little impact on our results. *Mesa* and *art* show higher BTB power reduction compared to other benchmarks. This is consistent with the data reported in Figure 1 where both benchmarks show higher number of CFCs compared to other applications.

Note that in the case of *mesa* we witness a small (less than 0.1%) performance improvement. Our study shows *mesa* achieves better performance as the result of better prediction for indirect jump instructions. *Art*, on the other hand, shows the highest performance loss among all applications. This can be explained by the fact that SABA does not predict CFCs occurring in *art* as accurately as other applications.

4.3.2 Execution Bandwidth

In Figure 7, we report BTB power reduction for four different pipeline execution bandwidths. To provide insight we also report maximum power reduction possible as achieved by a perfect CFC predictor (Oracle). While the entire bar shows maximum power reduction possible, the lower part in each bar reports power reduction achieved by a 64-entry SABA-filter using 3-bit counters. In a narrow pipeline, *e.g.,* the 2-way processor, fewer instructions are fetched each cycle. With fewer fetched instructions, there is lower chance of having a control flow instruction among those fetched. Hence, CFCs are more frequent. Consequently, there is higher chance for BTB power reduction by using SABA. This explains why BTB power reduction is higher for the 2-way processor. As the processor bandwidth increases so does the number of fetched instructions. Therefore, CFCs become less frequent and so does the BTB power reduction achieved by SABA.

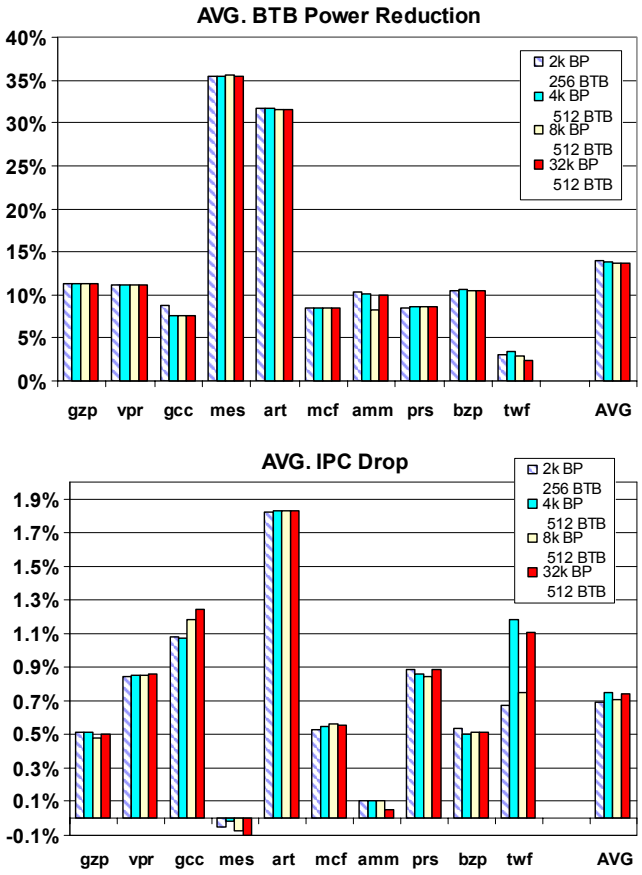

Fig. 6. *Average BTB power reduction and performance loss for different branch predictor and BTB size.*

We conclude from Figure 7 that SABA eliminates more than half of the unnecessary BTB accesses across all execution bandwidths. Note that this comes with negligible performance cost.

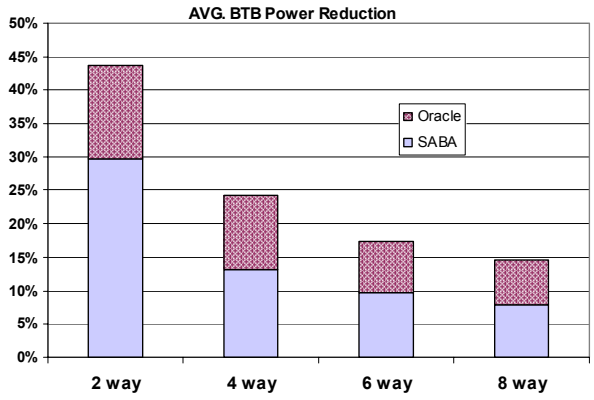

Fig. 7. *Maximum BTB power reduction (entire bar) and BTB power reduction achieved by SABA (lower bar) for processors with different execution bandwidths.*

5 Related Work

Previous work has introduced Banking and *prediction probe detector (PPD)* [2] to reduce predictor or BTB energy

consumption. Banking reduces the active portion of the predictor. Predictor Probe Detection (PPD) [2] reduces branch predictor energy consumption. PPD aims at reducing the power dissipated during predictor lookups. PPD identifies when a cache line has no conditional branches so that a lookup in the predictor buffer can be avoided. Also, it identifies when a cache line has no control-flow instruction at all, so that the BTB lookup can be eliminated. SABA can be used on top of banking to further reduce power dissipation. SABA is different from PPD as it does not impact timing complexities and makes decisions regarding BTB accesses occurring at future cycles.

Chung and Park proposed modifications to the branch predictor to perform early PHT access to predict BTB access [1]. Our work is different as it is independent of the branch predictor implementation and relies on fetch history to make predictions.

Baniasadi and Moshovos introduced Branch Predictor Prediction (BPP) [12] and Selective Predictor Access (SEPAS) [13] to reduce branch predictor energy consumption. BPP stores information regarding the sub-predictors accessed by the most recent branch instructions executed. This information is used to avoid accessing all three underlying structures. SEPAS selectively accesses a small filter to avoid unnecessary lookups or updates to the branch predictor. Our method is different from BPP and SEPAS as we focus on the BTB lookups.

Chaver *et al,* [10] used profiling to resize large BTB structures whenever reducing size does not impact the BTB miss rate. Hu *et al*, reduce leakage power in direction predictors [9]. They turn off cache lines if they have not been used in a long time. Monchiero *et al,* introduced a new *Hint* instruction to help identifying static branches and to reduce prediction power [11]. Our work is different as we use runtime information to avoid unnecessary BTB accesses.

6 Conclusion

In this work we presented SABA, a power-aware extension to high-performance processors, to identify and avoid unnecessary BTB accesses. SABA relies on a small filter referred to as the SABA-filter to record instruction fetch history. We used the recorded information to identify cycles where there is no control flow instruction among those fetched.

We studied how variations in SABA-filter configuration, branch predictor configuration and processor execution bandwidth impact power and performance. We eliminated more than half of the unnecessary BTB accesses with negligible performance cost. SABA does not impact predictor delay as it stops unnecessary accesses occurring in future cycles.

7 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.

References

[1] Sung Woo Chung, Sung-Bae Park, “A Low Power Branch Predictor to Selectively Access the BTB”, *Asia-Pacific Computer Systems Architecture Conference*, pp 374-384, September 2004.

[2] Dharmesh Parikh, Kevin Skadron, Yan Zhang, Mircea R. Stan, “Power-Aware Branch Prediction: Characterization and Design”, *IEEE Transaction on Computers*, Vol. 53,No. 2, pp. 168-186, February 2004.

[3] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A frame work for architectural power analysis and optimizations,” In *Proceedings 27th International Symposium on Computer Architecture*, 2000, pp. 83–94.

[4] Daniel A. Jimenez, Stephen W. Keckler, and Calvin Lin. “The Impact of Delay on The Design of Branch Predictors”, In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, Nov 2000.

[5] K. Diefendorff, “K7 Challenges”, Intel. Microprocessor Report, 12(14), Oct. 1998

[6] D. C. Burger and T. M. Austin. “The SimpleScalar tool set,version 2.0.”, *Computer Architecture News*, 25(3):13–25, Jun. 1997.

[7] S. Wilton and N. Jouppi. “An Enhanced Access and Cycle Time Model for On-chip Caches.” In *WRL Research Report 93/5, DEC Western Research Laboratory,* 1994.

[8] S. McFarling, “Combining Branch Predictors”, Technical Note TN-36, In *DEC Western Research Laboratory*, June 1993.

[9] Z. Hu, P. Juang, K. Skadron, D. Clark and M. Martonosi. “Applying Decay Strategies to Branch Predictors for Leakage Energy Saving”, In *Proceedings of IEEE International Conferences on Computers and Processors*, 2002.

[10] D. Chaver, L. Pinuel, M. Prieto, F. Tirado and M. C. Huang. “Branch Prediction on Demand: an Energy Efficient Solution”, In *ISPLED'03*, 2003.

[11] Matteo Monchiero, Gianluca Palermo, Mariagiovanna Sami, Cristina Silvano, Vittorio Zaccaria and Roberto Zafalon. “Low-Power Branch Prediction Techniques for VLIW Architectures: A Compiler-Hints Based Approach Integration”, *The VLSI Journal*, 38(3):515-524, January 2005.

[12] Amirali Baniasadi, Andreas Moshovos, “Branch Predictor Prediction: A Power-Aware Branch Predictor for High-Performance Processors”, In *Proceedings of ICCD 2002,* pp. 458-461

[13] Amirali Baniasadi, Andreas Moshovos, "SEPAS: A Highly Accurate and Energy Efficient Branch Predictor", *Proceedings of International Symposium on Low Power Electronics and Design*, 2004, pp. 38-43

[14] Juan, T., Sanjeevan, S., Navaro, J. J. "Dynamic History-Length Fitting: A third level of adaptivity for branch prediction", *Proc. of the 25th Ann. Int'l Symp. Computer Architecture*, Jun. 1998.

[15] R. Canal and A. Gonzalez. "A low-complexity issue logic", In *Proc. of 2000 International Conferences on Supercomputin*g, May 2000.

[16] Andreas Moshovos, "Checkpointing alternatives for high performance, power-aware processors", *Proceedings of International Symposium on Low Power Electronics and Design*, *2003*: pp. 318-321

[17] Philo Juang, Kevin Skadron, Margaret Martonosi, Zhigang Hu, Douglas W. Clark, Phil Diodato, Stefanos Kaxiras, "Implementing branch-predictor decay using quasi-static memory cells", *ACM Transactions on Architecture and Code Optimization*.pp. 180-219 (2004)

[18] Kevin Skadron, Tarek F. Abdelzaher, Mircea R. Stan: Control-Theoretic "Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management", *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02), 2-6 February 2002, Boston, Massachusettes, USA*: pp. 17-28