# Branchless Cycle Prediction for Embedded Processors

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

## ABSTRACT
Modern embedded processors access the Branch Target Buffer (BTB) every cycle to speculate branch target addresses. Such accesses, quite often, are unnecessary as there is no branch instruction among those fetched.

In this work we introduce **Branchless Cycle Prediction (BLCP)** to exploit this design inefficiency. BLCP uses a simple power efficient structure to predict cycles where there is no branch instruction among those fetched, at least one cycle in advance. We avoid accessing BTB during such cycles.

We show that, by using BLCP, it is possible to reduce BTB power dissipation by 32% while paying a negligible performance cost (average: 0.2%).

## Categories and Subject Descriptors
C.1.1 [**Processor Architectures**]: Single Data Stream Architectures

## General Terms
Design, Measurement, Performance.

## Keywords
Branch Target Buffer, Embedded Processors, Power-Aware Architectures, Low-Power Design.

## 1. INTRODUCTION
The goal of this work is to reduce branch target buffer (BTB) energy consumption without harming accuracy and hence overall performance. BTB is a major energy consuming structure in the processor and is often used by branch predictors for target address prediction.

To find the branch target address as soon as possible, at fetch, modern embedded processors (*e.g.*, Intel's XScale) [4]) access BTB every cycle and for all fetched instructions. While this results in early identification of the target address (for possibly branch instructions), it is inefficient as only a small percentage of these accesses result in predicting the target address. This is due to the fact that only a fraction of instructions are branch instructions. Nonetheless, in order to avoid extra delay, modern processors access BTB for all instructions. This aggressive approach, quite often, results in power dissipation without contributing to performance.

One possible approach to avoid extra BTB accesses for non-branch instructions is to use a pre-decoder to separate branch and non-branch instructions. The problem with this approach is that it can result in extra delay. As clock rates increase, wire delay will impact large structures such as the branch predictor [7]. Moreover, previous study shows that delay in the predictor significantly erodes performance. Therefore, any design choice resulting in increasing predictor delay, is not a good choice [8].

We introduce *Branchless Cycle Prediction* or *BLCP*, a power efficient technique that exploits instruction history to reduce predictor energy. We find occasions where no branch is among the instructions fetched. We avoid accessing the BTB during such occasions. To eliminate any timing overhead, we use fetch history to identify branchless cycles, at least one cycle in advance.

We use a subset of the MiBench benchmarks [3] and WATTCH [6] and show that with an average performance penalty of 0.2%, BLCP can reduce BTB energy consumption by 32%.

It is important to note that since BLCP provides information regarding instructions fetched in future cycles, it does not increase overall prediction latency.

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

## 2. MOTIVATION
Embedded processors often perform under resource constrains. It is due to such restrictions that designers use simple in-order architectures in embedded processors. As such, fetching a small number of instructions every cycle provides enough work to keep the pipeline busy [4].

Consistent with previous study [3], our study shows that embedded applications may have branch frequency anywhere from 5% to 30%. Considering the fact that modern embedded processors fetch very few instructions each cycle, chances are, quite often, there is no branch instruction among those fetched. We refer to such cycles as *branchless cycles* (or *BLC*). A *branch cycle* (or *BC*) is a fetch cycle where there is at least one branch instruction among those fetched.


*SAC'06*, April, 23-27, 2006, Dijon, France.

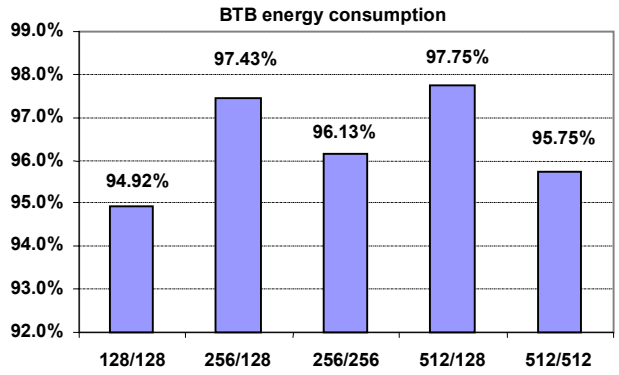



**Figure 1. BTB's energy consumption share in the branch predictor unit. Bars from left to right report for the following configurations (BTB size/bimodal predictor size): 128 entry BTB and bimodal predictor, a 256 entry BTB, 128-entry bimodal, 256-entry BTB and bimodal predictor, 512-entry BTB, 128-entry bimodal predictor and a 512-entry BTB and bimodal predictor.**

While currently exploited branch predictors are already consuming considerable energy [11], their consumption is expected to grow as embedded processors seek higher performance and exploit more resources. A considerable share of predictor energy is consumed by the BTB. In Figure 1 we report the percentage of predictor energy consumed by the BTB in a processor similar to Intel XScale as measured by Wattch. We report for different predictor configurations/sizes to cover both currently used predictors and those likely to be used in future. As presented the BTB is a major contributor to the overall predictor energy consumption.

Modern embedded processors use BTB to maintain steady instruction flow in the pipeline front-end [4]. The processor accesses the BTB to find the target address of the next instruction, possibly a branch. Unfortunately accessing BTB every cycle is not power efficient. While only taken branch instructions benefit from accessing the BTB, the BTB structure is accessed for every instruction. These extra accesses result in power dissipation without contributing to performance. The key motivating observation for this work is that by identifying occasions where accessing the BTB does not contribute to performance we can avoid these extra accesses. This will reduce power dissipation without harming performance.

**Table 1. The subset of MiBench benchmarks studied in this work and their BLC frequency.**

| Program | BLC | Program | BLC |
|---|---|---|---|
| *Blowfish Encode* | 87% | *Blowfish Decode* | 87% |
| *CRC* | 80% | *Dijkstra* | 82% |
| *D Jpeg* | 94% | *C Jpeg* | 85% |
| *GSM Toast* | 94% | *Lame* | 92% |
| *Rijndael Encode* | 94% | *SHA* | 93% |
| *String Search* | 78% | *Susan Smoothing* | 92% |

To investigate possible power reduction opportunities we report how often BLCs occur in each benchmark. Table 1 reports the percentage of BLCs for a representative subset of MiBench benchmarks [3] used in our study. On average, 88% of cycles are BLCs. In other words, 88% of the time, we could avoid accessing the BTB without losing performance.

Quite often a number of consecutive cycles may be branchless. We refer to these periods as branchless intervals. To provide better insight in Figure 2 we report how often different branchless intervals occur. As reported, about 90% branchless intervals take less than 10 cycles.

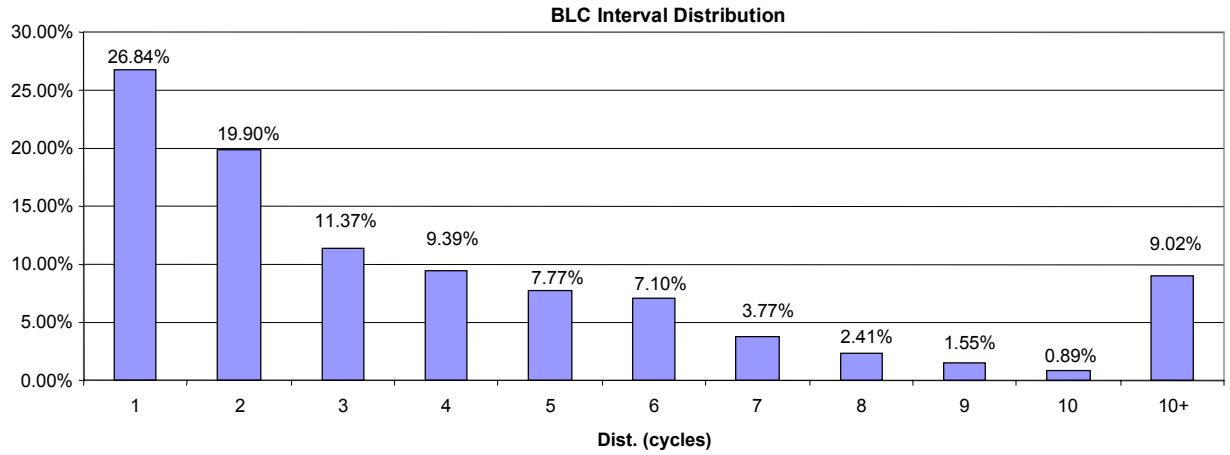



**Figure 2. BLC interval frequency.**

# 3. BRANCHLESS CYCLE PREDICTOR (BLCP)

We conclude from the previous section that there is an opportunity in the embedded space to reduce BTB energy consumption by avoiding unnecessary BTB accesses. In this section we propose a simple, highly accurate and power efficient predictor to identify such accesses. By identifying a BLC accurately and at least one cycle in advance, we avoid unnecessary BTB accesses. Depending on the number of instructions fetched during the predicted BLC, we avoid a number of unnecessary BTB accesses. Figure 3 shows our proposed BLC predictor architecture. We refer to our BLC predictor and the BLC-filter.

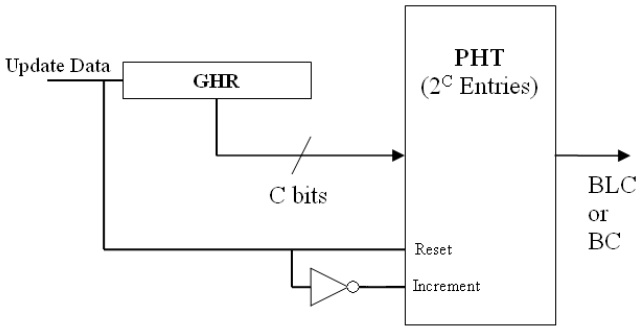



**Figure 3. BLC-Filter architecture.**

The BLC-filter is a history-based predictor which consists of two major parts: a small Global History Shift Register (GHR) and a Prediction History Table (PHT). The PHT size is decided by the GHR size. GHR is used to record the history of BCs and BLCs. 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.

BLC prediction is done every cycle. GHR records the most recent branch or branchless cycles. We represent every BLC in GHR with a zero and every BC with a one. The GHR value is used to access an entry in the PHT. Every PHT-entry has a saturating counter with the saturating value of *Sat*.

We probe the predictor every cycle. If the counter associated with the most recent GHR value is saturated we assume that the following cycle will be a BLC and avoid accessing the BTB (see Figure 4(a)).

We update BLC-filter every cycle and as soon as we know whether there has been a branch among the instructions fetched in the most recent fetch cycle. A BC results in updating the GHR with a one, where a BLC results in shifting in a zero in the GHR. We use the GHR to access the PHT entry to update. We increment the associated PHT counter if the latest group of fetched instructions does not include any branches. We reset the associated counter if there is at least one branch among those fetched.

**Table 2. Simulated processor configuration**

| Processor Core | |
|---|---|
| Instruction Window | RUU= 8; LSQ=8 |
| Issue Fetch | 1 Instruction per Cycle; |
| Issue Width | 2 Instruction per Cycle; |
| | 1 integer, 1 FP |
| Miss Pred. Penalty | 6 cycle |
| Fetch Buffer | 8 entries |
| Functional Units | 1 Int ALU, 1 Int mult/div |
| | 1 FP ALU, 1 FP mlt/div, 1 mem prt |
| **Memory Hierarchy** | |
| L1 D-cache Size | 32 KB, 32-way, 32B blocks, wr bk |
| L1 I-cache Size | 32 KB, 32-way, 32B blocks, wr bk |
| L1 latency | 1 cycle |
| L2 | N/A |
| Memory latency | 32 cycles |
| D-TLB/I-TLB Size | 128/128-entry, fully assoc., |
| | 30-cycle miss |
| **Branch Prediction** | |
| BTB | 128-entry, 1-way |
| Direction Predictor | bimodal predictor, 128 entries |
| Return-address-stack | N/A |

As presented in Figure 4(b), we update the BLC-filter every cycle. One way to maintain the filter as accurate as possible is to use decode-based information. Accordingly, at decode we check if there has been any branch instruction among those decoded. We update the predictor's entry associated with the history at the time the branch instruction was fetched. Finding the entry is done by shifting left the most recent history by 2 bits (our fetch latency is 2 cycles).

# 4. METHODOLOGY AND RESULTS

We used programs from the MiBench embedded benchmark suite compiled for the Intel Xscale-like architecture simulated by the Simplescalar v3.0 tool set [9]. Table 1 reports the subset of MiBench benchmarks we used in our study. We used GNU's gcc compiler. We simulated the complete benchmark or half a billion instructions, whichever comes earlier. We detail the base processor model in table 2.

(a) Lookup

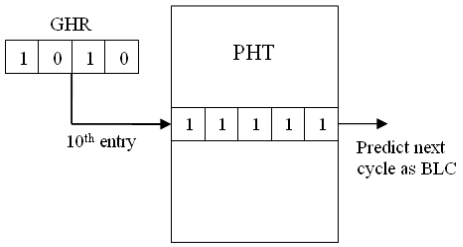
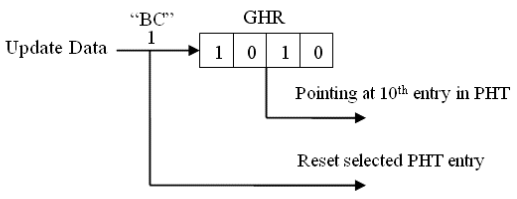



**Figure 4. (a) Branchless cycle predictor lookup.**

(b) Update




**(b) Branchless cycle predictor update.**

To evaluate our technique, we used a modified version of Wattch tool set [6]. We report both accuracy (*i.e.*, how often we accurately predict a BLC) and coverage (*i.e.,* what percentage of BLCs are accurately identified).

Provided that a sufficient number of BLCs are accurately identified, BLCP can potentially reduce BTB energy consumption. However, it introduces extra energy overhead and can increase overall energy if the necessary behavior is not there.

We used CACTI [10] to estimate the energy overhead associated with BLCP. In Figure 1 we report the relative energy consumed per access by an 8-entry BLCP filter using 6 bit counters and other structures used by the branch predictor in Table 3. Numbers reflect the energy consumed by each structure compared to the energy consumed by a branch predictor equipped with 128-entry bimodal predictor and a direct-mapped 128-entry BTB. As reported the overhead of the 8-entry BLCP filter is far less than the energy consumed by the BTB. Nonetheless, we take into account this overhead in our study.

**Table 3: Energy consumed per access by the branch predictor units and the BLCP-filter.**

| Modeled Unit | Size | Percentage |
|---|---|---|
| **BLCP** | 8 x 6 | 1.61% |
| BTB | 128 x 1-way | 94.92% |
| bimodal dir. Predictor | 128 entries 2- bits | 3.47% |

## 4.1 Accuracy and Coverage

In Figure 5 we report prediction accuracy and coverage for BLCP. In 5(a) we report average accuracy for different GHR sizes (*i.e.*, 1 to 6) and saturating counters (*i.e.*, 2 to 6-bit counters). As reported we predict BLC cycles with an accuracy varying from 91% to 99.6% for different BLCP-filter configurations.

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

In 5(b) we report what percentage of BLCs is identified. We identify between 25% and 81 % of BLCs for different BLCP configurations. We conclude from Figure 5 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 and coverage. While a small GHR size results in recording very little history, larger GHR sizes may result in mapping small repeating patterns to different BLCP-filter entries which may result in a longer BLCP learning period.

## 4.2 Energy and Performance

We report BTB energy reduction and processor performance loss in Figure 6. We use the same set of parameters used in Figure 5. BTB energy reduction varies between 21% and 75% for different BLCP-filter configurations. For a fixed GHR-size larger saturation threshold comes with lower performance loss but at the expense of reduced energy savings. For a fixed counter size, increasing the GHR-size almost always improves BTB energy reduction. Note that small number of saturating bits (*e.g.*, 2-bit counters) results in high performance slowdown. Larger GHR sizes also come with high overhead as they require larger PHT tables. We find a BLC-filter with a GHR-size of 3 which uses 6-bit saturating counters the most efficient one. Using this configuration, on average, we reduce BTB energy consumption by 32% with an average performance cost of 0.2%.

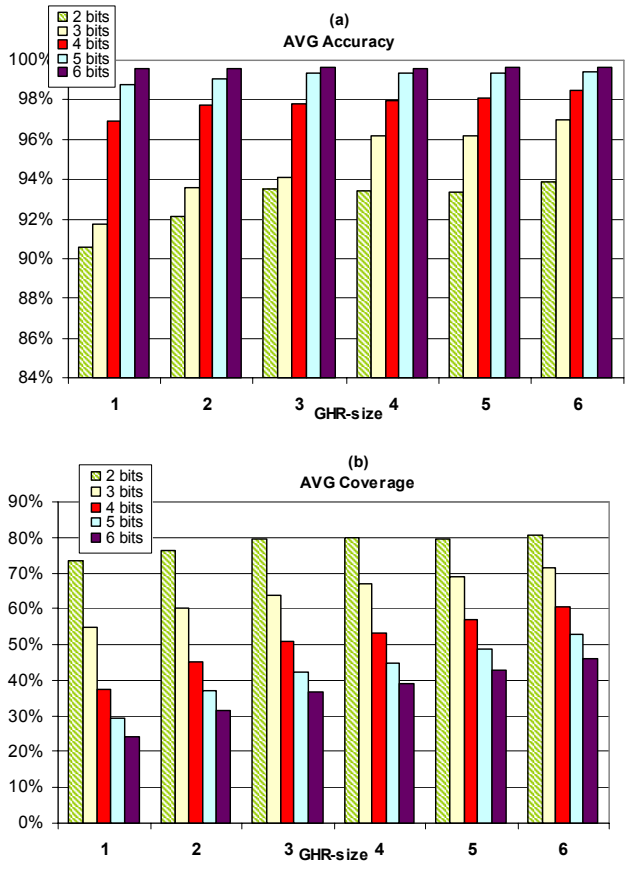



**Figure 5. Average accuracy and coverage achieved for different BLC-filter configurations and for the MiBench benchmarks studied here. For each GHR-size (x-axis) bars from left to right report for 2 to 6-bit saturating counters.**

Note that we observe small performance improvement for GHR-size of 1 and 6-bit saturating counters. Our study shows that some applications achieve better performance as the result of better prediction for indirect jump instructions.

# 5. RELATED WORK

Petrov and Orailoglu introduced static techniques to reduce BTB energy consumption in embedded processors [2]. BLCP is different as it uses run-time information.

Other techniques suggested to reduce BTB power dissipation include banking and prediction probe detector (PPD) [5]. Banking reduces the active portion of the predictor. Predictor Probe Detection (PPD) [5] 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-

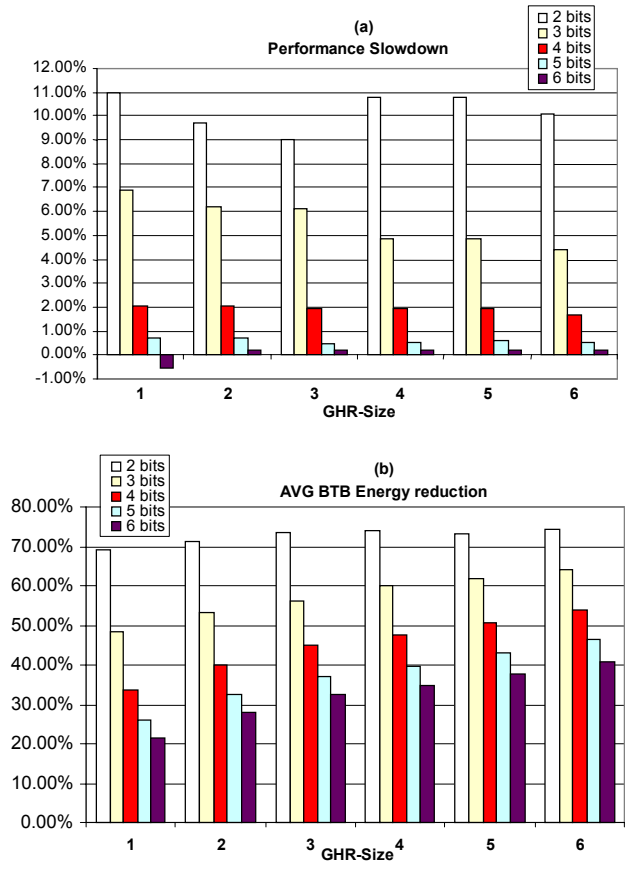



**Figure 6. Average performance slowdown and BTB energy reduction for different BLC-filter configurations and for the MiBench benchmarks studied here. For each GHR-size (x-axis) bars from left to right report for 2 to 6-bit saturating counters.**

flow instructions at all, so that the BTB lookup can be eliminated. Our predictor can be used on top of banking to further reduce power dissipation. Our predictor is different from PPD as it predicts application behavior in future cycles.

Chung and Park proposed early branch predictor accessing to predict BTB access [1]. There method required modifying the branch predictor. Our method is independent of the branch predictor implementation and uses past behavior to make predictions.

# 6. CONCLUSION

In this work we introduced BLCP, a power efficient method to identify and avoid unnecessary BTB accesses. BLCP relies on a small filter referred to as the BLC-filter to record and use instruction fetch history. BLCP uses the recorded information to identify cycles where there is no instruction among those fetched.

We studied how variations in the BLC-filter configuration impacts energy and performance. By using a small low-overhead structure we reduced BTB energy consumption considerably with negligible performance cost. BLCP 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 of Innovation, New Opportunities Fund.

# 8. REFERENCE

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

[2] Peter Petrov and Alex Orailoglu. "Low-Power Branch Target Buffer for Application-Specific Embedded Processors" *Euromicro Symposium on Digital System Design (DSD)*, pp. 158-165, September 2003.

[3] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001

[4] Sudarshan K. Srinivasan, Miroslav N. Velev, "Formal Verification of an Intel XScale Processor Model with Scoreboarding, Specialized Execution Pipelines, and Impress Data-Memory Exceptions", MEMOCODE 2003, pp. 65-74

[5] 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.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A frame work for architectural power analysis and optimizations," in *Proc. 27th Int. Symp. Computer Architecture*, 2000, pp. 83–94.

[7] V Agarwal, M.S. Hrishikesh, D. Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures", *ISCA* 27, May 2000

[8] 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 Int'l Symposium on Microarchitecture*, Nov 2000.

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

[10] 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.

[11] Michele Co, Dee A.B. Weikle, and Kevin Skadron, "A Break-Even Formulation for Evaluating Branch Predictor Energy Efficiency", *Workshop on Complexity-Effective Design (WCED) in conjunction with the 32nd Annual ACM/IEEE International Symposium on Computer Architecture,* 2005