Branch Predictor Prediction: A Power-Aware Branch Predictor for High-Performance Processors

Amirali Baniasadi
Electrical and Computer Engineering
Northwestern University

Andreas Moshovos
Electrical and Computer Engineering
University of Toronto

ABSTRACT

We introduce Branch Predictor Prediction (BPP) as a power-aware branch prediction technique for high performance processors. Our predictor reduces branch prediction power dissipation by selectively turning on and off two of the three tables used in the combined branch predictor. BPP relies on a small buffer that stores the addresses and the sub-predictors used by the most recent branches executed. Later we refer to this buffer to decide if any of the sub-predictors and the selector could be gated without harming performance. In this work we study power and performance trade-offs for a subset of SPEC 2k benchmarks. We show that on the average and for an 8-way processor, BPP can reduce branch prediction power dissipation by 28% and 14% compared to non-banked and banked 32k predictors respectively. This comes with a negligible impact on performance (1% max). We show that BPP always reduces power even for smaller predictors and that it offers better overall power and performance compared to simpler predictors.

1. INTRODUCTION

Dynamic power dissipation has emerged as a first class consideration in modern, high-performance processor design. In this work, we focus on power-efficient and highly-accurate branch predictors. Reportedly, for a typical high-performance processor, branch predictors dissipate a considerable amount (*i.e.*, about 10%)of overall power [1].

Overall power dissipation heavily depends on branch prediction accuracy making developing power-efficient branch predictors a challenging task. In isolation, it would seem that using smaller predictors would result in reduced power dissipation. However, this is not true since branch predictor accuracy indirectly impacts the amount of work and hence the amount of power that is consumed by the rest of the processor [2]. Accordingly, in this work we focus on predictors that are both *power-efficient* and *highly-accurate*.

We propose *branch predictor prediction* or *BPP* as a power-efficient extension to the commonly used combined predictors. Combined predictors use three underlying *sub-predictors* that are all accessed for every branch. Out of the three prediction hints, one is used to select among the other two. BPP exploits the *temporal* and *sub-predictor locality* characteristics of typical branch streams to allow us to gate two out of the three sub-predictors for many branches. Specifically, we have observed that: 1) often, short sequences of branches tend to appear repeatedly, and 2) they tend to use the same sub-predictors. In BPP, a small buffer is introduced in the fetch stage. BPP entries record the sub-predictors used by recent branches. Each BPP entry is tagged by the PC of a recently seen branch and records the sub-predictors used by the two branches that followed it in the dynamic execution stream. By associating sub-predictor hints with a preceding branch, BPP avoids increasing prediction latency. The hints become available at least one cycle in advance of when the actual prediction needs to take place.

We show that our predictor can reduce branch predictor power by 14% over a 32K-entry, conventional, banked combined branch predictor (taking into consideration the BTB and the return address stack that our BPP does not optimize). Moreover, we show that a processor with BPP is *always* more power efficient than one that uses any of the sub-predictors alone. Finally, we show that BPP reduces power even for smaller branch predictors (range 16K-entries down to 1K-entries).

The rest of the paper is organized as follows. In section 2 we explain BPP. In section 3 we present our experimental evaluation. We report performance, and power savings. We report relative power reduction for both the predictor and the entire processor. We compare our model to three different alternatives. In section 4, we study how BPP reacts to changes in predictor size and BPP configuration. In section 5 we review related work. Finally, in section 6, we summarize our findings.

2. BRANCH PREDICTOR PREDICTION

Branch prediction is essential to sustaining high performance in modern high-end processors. The combined predictor is used in many high-performance designs.

Combined predictors use three underlying sub-predictors. Two of the sub-predictors produce predictions for branches. They are typically tuned for different branch behaviors. By using a selector predictor, combined predictors offer the best of both underlying sub-predictors.

While using three tables makes better prediction possible, it also increases power dissipation. Our analysis shows that for a 32K-entry predictor, the three sub-predictors consume close to 65% of the total predictor power (the rest is consumed in the BTB and the return address stack).

BPP relies on typical program behavior, to gate two out of the three underlying sub-predictors for most branches. In particular, BPP exploits the following two phenomena: (1) Branch instructions show strong temporal locality. We measured the temporal locality in the dynamic branch stream by studying how often a branch is within the last *n* branches fetched, where *n* varies from 8 to 64. On average and for a subset of SPEC'2k benchmarks, almost half of the branches appear within eight branches apart and about 83% of the branches appear within the last 64 branches fetched. (2) Often branches tend to use the same sub-predictor. Our study shows that on the average, a branch uses the same sub-predictor with a 91% probability for a subset of SPEC'2K benchmarks.

Based on the above we suggest BPP. At the core of BPP we use a FIFO buffer called the BPP-buffer. BPP-buffer is a small power-efficient structure that stores information regarding the most recent branches encountered. The stored information is used to gate the selector and one of the two sub-predictors. This *n*-entry FIFO buffer records the *n* most recently fetched branches in sequence. Every buffer entry includes an address field and a two bit *sub-predictor hint* field. We use the latter to record the last sub-predictor used by the specific branch. Combinations 00 and 01 indicate that the gshare and bimodal predictors have been the last predictors used to speculate the branch outcome respectively. We assign combination 11 to indicate that the branch was miss-predicted last time speculated. We do not gate the sub-predictors if the miss-predicted branch reappears since we have no confidence on the predictor used last time.

The front-end fetches up to two branches every cycle for a total of eight instructions. Once all instructions are fetched we check the BPP buffer to see if the last branch fetched is among those recorded in the buffer. If no match is found, all sub-predictors will be probed during the next cycle. If a match is found, we look at the sub-predictor hints of the next two buffer elements. Since branch sequences tend to repeat, we implicitly predict (assume) that these access hints are the right ones for the next two branches that may be fetched during the next cycle. We gate two of the three tables if the next two (guessed by the BPP-buffer) dynamic branches have used (and therefore are predicted to use) the same sub-predictor and they have not been miss-predicted.

We access the BPP-buffer in parallel with the branch predictor. This is true for both branch lookup and update. We allocate BPP entries at fetch in order. We update the BPP-buffer speculatively as soon as a branch calculates its direction. We do not flush the BPP entries that follow a miss-predicted branch. We instead mark the miss-predicted branch. This allows us to salvage sub-predictor locality down miss-predicted paths. For example, due to control independence some of the sub-predictor hints may be valid even though an earlier branch was miss-predicted. We lookup the BPP buffer to decide what sub-predictor to gate (if any) for the next cycle. While BPP uses branch temporal locality it is different from a branch predictor cache of shorts since it associates sub-predictor hints with preceding branches.

Provided that sufficient branch and sub-predictor locality exists, BPP has the potential for reducing branch prediction power dissipation. However, it introduces extra power overhead and can, in principle, increase overall power dissipation if the necessary behavior is not there. We take into account this overhead in our study and show that for the programs we studied, BPP is robust.

3. METHODOLOGY AND RESULTS

In this section, we present our analysis of the BPP technique. We report performance results in section 3.1 We report power measurements in section 3.2.

We used programs from the SPEC2K suite compiled for the MIPS-like architecture used by the Simplescalar v3.0 simulation tool set. We used GNU's gcc compiler (flags: -O2 –funroll-loops –finline-functions). Table 1 reports the branch prediction accuracy per benchmark. To obtain reasonable simulation times we simulated 200M of the instructions after skipping the initialization. We detail the base processor model in table 2.

For most of our experiments we used a combined predictor with 32K-entries per sub-predictor. In section 4, we demonstrate that BPP is still effective even with smaller predictors. We use the 32K-entry predictor after investigating predictors of different sizes and studying performance and accuracy. Our study shows that further increase in predictor size rarely and slightly contributes to performance while increases predictor power dissipation dramatically. Our study shows that while using smaller predictors saves predictor power, it also results in higher overall processor power dissipation.

We used WATTCH [3] for power estimation. We modeled an aggressive 2GHz superscalar microarchitecture manufactured under a 0.1 micron technology.

Program	Ab.	BP Acc.	Program	Ab.	BP Acc.
ammp	amm	99%	***mesa***	mes	99%
bzip	bzp	97%	***parser***	prs	93%
compress	cmp	92%	***vortex***	vor	98%
equake	equ	98%	***vpr***	vpr	92%
gcc	gcc	93%	***wolf***	wlf	92%
mcf	mcf	91%			

Table 1: Benchmarks and control flow prediction accuracy (direction and target).

Branch Predictor	32K GShare+32K bi-modal w/ 32K selector
Scheduler	128 entries, RUU-like
Fetch Unit	Up to 8 instr./cycle. Max 2 branches/cycle 64-Entry Fetch Buffer
Load/Store Queue	128 entries, 4 loads or stores per cycle Perfect disambiguation
OOO Core	any 8 instructions / cycle
Func. Unit Latencies	same as MIPS R10000
L1 - Instruction /Data Caches	64K, 4-way SA, 32-byte blocks, 3 cycle hit latency
Unified L2	256K, 4-way SA, 64-byte blocks, 16-cycle hit latency
Main Memory	Infinite, 100 cycles

Table 2: Base processor configuration.

3.1. Performance

BPP can negatively impact accuracy and hence performance. Therefore, we first investigate how BPP impacts performance. To determine whether our technique is indeed worthwhile, it is important to compare it with a number of obvious alternatives. Accordingly, we compare with a number of alternative machines that use different branch predictor organizations. We compare with three alternative branch predictors: (1) **Conventional Combined Predictor (CMB)** (2) **Bimodal-Predictor (BMD)** (3) **Gshare-Predictor (GSH)**. Comparing to CMB shows whether our power savings are worth the possible performance loss, BMD and GSH comparisons explore the possible benefits and costs of the combined predictor when compared against its two sub-predictors.

In figure 1 we report performance for the processor that uses a 32-entry BPP (in section 4 we vary BPP size) compared to 3 different base cases. Bars from left to right report *relative performance* compared to CMB, BMD and GSH. On average, performance slowdown is 0.3% compared to CMB. In the worst case of *mcf*, it is only 1%. As it can be seen in table 1, mcf exhibits the worst branch behavior. However, even then, the performance loss is only 1%, making BPP an attractive alternative to a conventional combined predictor (CMB). Comparing to BMD and GSH, on average, BPP offers higher performance. The performance difference is amplified for some of the integer benchmarks (e.g., *gcc*), while they are negligible for most of the floating point benchmarks (e.g., *mes*).

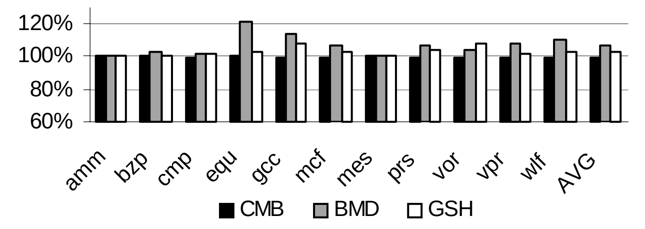


Figure 1: Performance (higher is better). Bars from left to right compare BPP to the CMB, BMD and GSH.

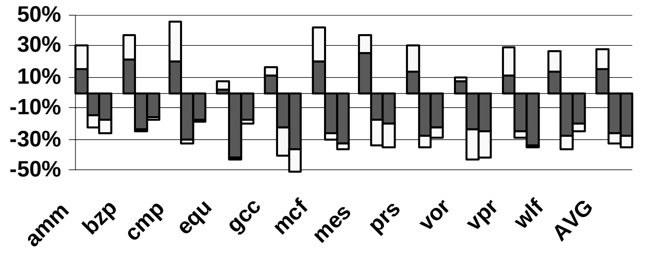
3.2. Power

As an indicator of power reduction we studied how often BPP gates two out of the three sub-predictors per benchmark for SPEC'2k benchmarks. Our measurements show that on the average, BPP gates sub-predictors 54% of time reaching a maximum of 96% and a minimum of 34%.

In the rest of the section we first report predictor power dissipation. while exploiting a less complex branch predictor reduces predictor's power dissipation, it may increase the total power dissipation[1]. Therefore, we also report total power dissipation. In our experiments we take into account the power overhead associated with the BPP buffer. Moreover, we also study how BPP interacts with banked predictors (suggested by previous work[1]). Banking is particularly important for large predictors. We used four banks based on CACTI's analysis. We make the following assumptions to *pessimistically* account for power overheads and static power: First, we assume gated structures still consume 10% of their maximum power dissipation. Second, in banked predictors we assume that all banks are always precharged (*i.e.*, banks need to be ready in case they are accessed by the next branch). It would be possible to use BPP to selectively precharge only the sub-predictor banks that we will use.

Figure 2 reports relative predictor and overall power reduction for non-banked and banked predictors. For each benchmark, bars from left to right report power reduction compared to CMB, BMD and GSH. The entire bar shows savings achieved for non-banked predictors. The dark portion of each bar shows savings for banked predictors. Figure 2(a) reports predictor power reduction. On average, non-banked BPP reduces predictor power dissipation 28% compared to a non-banked CMB. For banked predictors, power savings are lower but still considerable. A banked BPP reduces branch predictor power 14% compared to a banked CMB. For both banked and non-banked predictors, as expected, BPP dissipates more power than BMD and GSH.

In figure 2(b) we report overall power reduction for processors using banked and non-banked predictors. For non-banked BPP, compared to non-banked CMB, BMD and GSH average power reductions are 1.7%, 5.6% and 1.8%. For a banked BPP and compared to a banked CMB, BMD and GSH average power reductions are 0.6%, 5.2% and 1.4%. This suggests that BPP reduces both the predictor's and the total processor power dissipation compared to CMB. Moreover it shows that using a combined predictor both increases performance and saves power when compared to using only one of its sub-predictors.



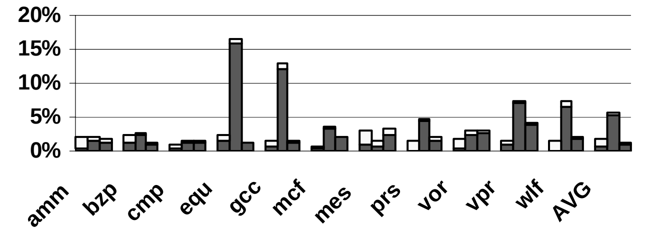


Figure 2: (a) Branch prediction power reduction (b) Overall power reduction. Bars from left to right compare to CMB, BMD and GSH. Entire bar reports for non-banked and dark portion of the bar reports for banked predictors.

4. SENSITIVITY ANALYSIS

In this section, we study how BPP reacts to smaller predictor sizes and different BPP buffer sizes.

First we report BPP sensitivity to predictor size. We studied relative predictor power dissipation compared to a non-banked combined predictor with a similar predictor size. We picked non-banked predictors since banking is reportedly less effective for smaller predictors[1]. Our studies show that savings are less but still obtainable for smaller predictors. On average, BPP reduces power by 4%, 8%, 13%, 16% and 21% for predictor sizes 1k, 2k, 4k, 8k and 16k.

We also studied how BPP reacts to various BPP buffer sizes. Our studies show that performance stays within 0.5% for buffer sizes 64, 32, 16 and 8. On average, and for a banked predictor, predictor power reduction is 11.3%, 14%, 10.4% and 8.7% for buffer sizes 64, 32, 16 and 8 respectively. Therefore, savings are maximum for a 32-entry buffer. Apparently, a 32-entry buffer is large enough to store the required information. While 8- and 16-entry buffers appear to be too small, a 64-entry buffer imposes unnecessary power overhead.

5. RELATED WORK

Previous work has introduced banking and the usage of *prediction probe detector (PPD)* as two techniques that reduce branch predictor power dissipation without harming accuracy[1]. Banking reduces the active portion of the predictor. In this paper we studied how BPP interacts with banking and demonstrated that BPP can reduce power even when the underlying predictors are banked. PPD aims at eliminating unnecessary BTB and predictor accesses. While BPP does not target reducing the BTB power dissipation it could be used on top of PPD to further reduce power dissipation by gating the sub-predictors when there are predictor lookups required.

6. CONCLUSION

We presented BPP, a technique for reducing power while maintaining the accuracy advantage of combined branch predictors.We affirmed that it is possible to significantly reduce power by exploiting a) the temporal locality amongst branch instructions, and b) the high predictability in their usage of sub-predictors. On average, BPP reduces predictor power dissipation 28%, compared to a non-banked and 14% compared to a banked conventional combined predictor. This comes with a negligible performance degradation. We have also shown that while BPP is more effective for larger predictors, it still reduces power even for smaller branch predictors. More importantly, we have shown that when one considers the overall processor power dissipation, BPP-enhanced processors always dissipate less power when compared to ones that use either just the conventional combined predictor, or just one of its underlying sub-predictors.

Because of the considerable power savings and the relatively small cost, BPP is an attractive power-aware enhancement for modern, high-performance processors.

REFERENCES

[1] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, amd M.R. Stan. *Power Issues Related to Branch Prediction. In Proc.* Intl. Symposium on High-Performance Computer Architecture, February 2002.

[2] A. Baniasadi, A. Moshovos, *Instruction Flow-Based Front-end Throttling for Power-Aware High-Performance Processors. In Proc.* ISLPED'01, August 2001

[3] D. Brooks, V. Tiwari M. Martonosi "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", *Proc of the 27th Int'l Symp. on Computer Architecture*, 2000