BTB Implementation in C

Kevin Evans
11571810

May 3, 2022

Introduction

The Branch Target Buffer (BTB) is a prediction technique used to decrease the branch penalty in pipelined computer processors. This method stores a cache table containing an index, branch program address/counter (PC), a target PC, and a 2-bit prediction. When a branch occurs, it uses the BTB to lookup the mapped target. If the target is in the table, it uses the prediction value to determine whether to jump

With a five-stage pipeline and without a BTB implementation, a branch can create a stall lasting for one or more clock cycles. It is important to utilize the BTB to ideally reduce the stalls to zero. In a non-ideal program, the BTB can introduce stalls: when there is a BTB miss, and a branch is taken, it creates 1 stall. In the case of a BTB hit resulting in a misprediction, the prediction must be scrapped creating 1 stall.

Program design

This program implements a branch target buffer (BTB) in C. The program accepts a text file trace of a previously-ran program, containing the program addresses as it progresses. The program reads the file line-by-line and creates a BTB containing the current PC with the next PC. As the trace file progresses, the BTB is used and several statistics are calculated, shown in the next section below. This program implements two state machines, shown below in Figure 1. The state machine determines the "next" state of the prediction attribute of each entry in the BTB table.

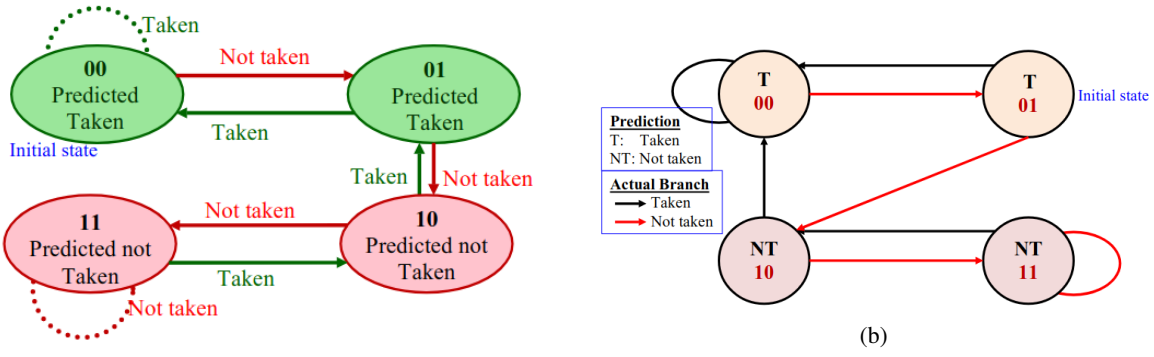

Figure 1: The two state machines used in this program, when determining the *next* state of each BTB entry.

Parameters set and observed

Two benchmarks were used, `Li_int` and `Dodoc_FP`. These files contain traces of the execution of two programs, each with roughly 700K+ entries. The BTB is generated and we collect the hit/miss rate, number of correct/wrong entries, number of branches taken, BTB entry collisions, number of wrong addresses. We additionally calculate the number of additional stalls caused by the BTB misses and mispredictions,

$$\begin{aligned}\text{approx. stalls} = \text{IC} \times \text{miss rate} \times \text{branch NT rate} \times 1 \text{ stall} \\ + \text{IC} \times \text{hit rate} \times \text{misprediction rate} \times 1 \text{ stall}.\end{aligned} \tag{1}$$

Simulation results

For both state machines and both benchmarks, the results are summarized in Table 1 and 2. These results are plotted in Fig. 2. Using (1), the approximate stalls were calculated: ≈ 4000 for `Li_int`, ≈ 2000 for `Dodoc_FP`. Each of these is minuscule compared to the total number of branches causing a stall, 132694 and 124254 respectively.

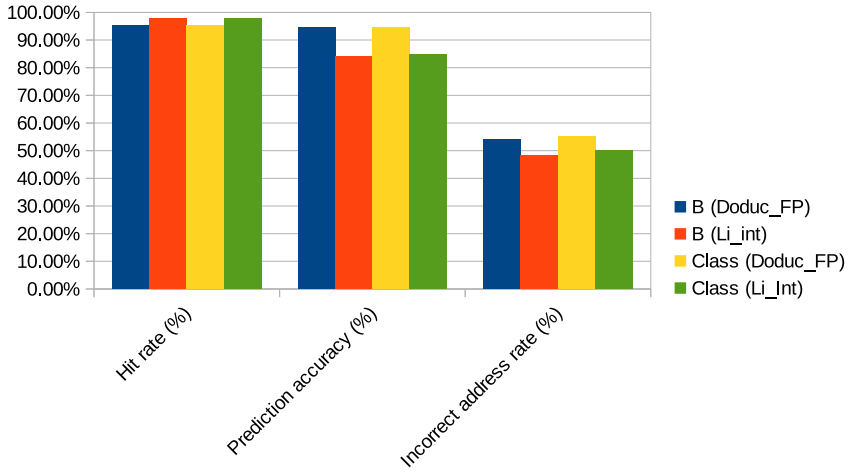

Figure 2: A comparison of the BTB results of the two state machines and two benchmarks.

Quantitative assessment of results

Both state machines for both benchmarks performed well, achieving a prediction accuracy of roughly 84% for both state machines in `Li_int`; and a hit rate of 94% for both machines in `Dodoc_FP`. For these two benchmarks, using the BTB reduces the stalls and improves branch performance by 32 to 64 times. This was calculated using

$$\%\text{ stall reduction} = \frac{\text{approx. stalls with BTB}}{\text{stalls without BTB}} = \frac{\text{approx. stalls with BTB}}{\text{total number of branches} \times 1 \text{ stall}}. \tag{2}$$

The significant reduction of stalls shows the importance of the 2-bit BTB on performance.

Concluding remarks

This 2-bit BTB can greatly improve performance. The implementation of the 2-bit BTB in C achieved an accuracy rate exceeding 80% and reduced the stalls by 32 to 64 times, for both state machines in two benchmarks. It would be advised to use branch prediction using a BTB on a pipelined processor, instead of not implementing a BTB. Other experiments could use a 1-bit or 3-bit predictor to potentially improve prediction accuracy.

Table 1: The results of two state machines using `Li_int` trace.

	Class State Machine	State Machine B
IC	716529	716529
Hit	129557	129557
Miss	3137	3137
Right	109691	108999
Wrong	19866	20558
Taken	111571	111571
Collision	2535	2535
Wrong addresses	9954	9954
Approx. stalls	4080	4205
Total branches	132694	132694
Hit rate	97.64%	97.64%
Prediction accuracy rate	84.67%	84.13%
Incorrect address rate	50.11%	48.42%
Stall reduction	32.51×	31.55×

Table 2: The results of two state machines using `Doduc_FP` trace.

	Class State Machine	State Machine B
IC	724090	724090
Hit	118421	118421
Miss	5833	5833
Right	111935	111786
Wrong	6486	6635
Taken	109320	109320
Collision	5114	5114
Wrong addresses	3576	3576
Approx. stalls	1941	1965
Total branches	124254	124254
Hit rate	95.31%	95.31%
Prediction accuracy	94.52%	94.40%
Incorrect address rate	55.13%	53.90%
Stall reduction	64.00×	63.21×

Appendix A: BTB simulation code

Please see the accompanying file `main.c`.