ECE552, Lab Assignment 2: Dynamic Branch Prediction

Sebastian Wardzinski - 1004070676

Microbenchmark

Note that my mb.c file is highly commented and can be read as a stand-alone document. Here I will provide a summary of the tests and explain their validity.

The microbenchmark was compiled using no compilation flags (therefore "-O0"). I initially ran the file with no code inside, this gave me an idea of the overhead mispredictions I should expect (I got ~1760). I created 3 nested pairs of loops, the outer loop always having a large value that was not a multiple of the other outer loops. This makes it easy to see the results of each case:

1) The 1st set of loops had an inner loop that ran 6 times (and exited on the 7th). Since our 2level predictor has a history of 6, it was able to handle this with no problem. This also proves that the history registers are private, since otherwise the fact that the outer loop is taken would make it so that there is a streak of 7 takens, which would be too long of a pattern for our design.

2) The 2nd set of loops ran the inner loop 7 times. This pattern is too long for our predictor, causing it to fail as expected, causing one misprediction for every iteration of the outer loop.

3) The 3rd set of loops ran a while loop 6 times, with an if statement right after. Here the two branch statements are close enough to have aliasing in the BHT register. This if-statement was always taken, so it would combine with the while-loop 'taken' statements from the next iteration to make a streak of 7 takens. As expected, this caused a misprediction on every outer loop iteration.

Table 1: Branch Predictor mispredictions/misprediction rate per benchmark

BP Type	astar	bwaves	bzip2	gcc	gromacs	hmmer	mcf	soplex
2bitsat	3695830 24.639	1182969 7.886	1224967 8.166	3161868 21.079	1363248 9.088	2035080 13.567	3657986 24.387	1065988 7.107
2level	1785464 11.903	1071909 7.146	1297677 8.651	2223671 14.824	1122586 7.484	2230774 14.872	2024172 13.494	1022869 6.819
opened	792468 5.283	864093 5.761	1166877 7.779	820425 5.470	796676 5.311	1928972 12.860	1617015 10.780	794351 5.296

* The average misprediction rate for each branch predictor is 14.5, 10.6, and 7.32 respectively

Open-ended branch predictor

I used a hybrid branch predictor, which ran two branch predictors independently, then chose which predictor to trust based on their past performance at a specific branch instruction (with aliasing being possible). The two predictors used in my hybrid design were:

- Gshare: indexes a table of 2^{13} 2-bit sat counters by XORing the least-significant 13 bits of the address of the branch instruction with the 13 bit-long global branch history
- 2level: scaled up version of second branch predictor in assignment

This design consumed ~120KB of memory, composed from the following components:

- 32KB * 2 bits + 15 bits ⟶ the array of 2-bit saturating counters for gshare + history register
- 2^12 * 8 (32KB BHT) + 2^4 * 2^8 * 2 bits (8KB PHT) ⟶ for the 2level implementation
- 8KB * 2 bits ⟶ for the hybrid selector

I initially chose the gshare predictor because it has a simple yet elegant design which minimizes aliasing while using a memory efficient global history register. This netted me an average of ~7.6 mispredictions per 1k instructions. To bring me below the 7.5 threshold I needed a little boost; so I tried incorporating gshare into a hybrid branch predictor. When using 2bitsat as my second predictor I cut my misprediction rate to ~7.4; however, by using 2level it improved to ~7.3.

CACTI

2level: Two RAM modules were used (file *-1 is for BHT, *-2 is for PHT), where only the size was changed. The first module contains the 512 entries of the BHT table. Each history register is 6 bits, but we still need to allocate a full 1 byte line to them because of the constraints of the program, so we need 512 bytes in total. The second file contains the 8 PHT tables, each with 64 2-bit counters. In total we have 512 2-bit counters, and each 1 byte line can hold 4 of them, so we need 128 bytes. All their statistics have been summed since each module is accessed sequentially on each request.

openend: Here 4 RAM modules were used, for the BHT & PHT (*-1 and *-2), the gshare memory (*-3), and the selector memory (*-4). The size of each module is adjusted (see config file for calculation right above the "-size" parameter), but all the block sizes are still 1 byte since we either access 8-bit histories or 2-bit counters. The energy, power, and area statistics of all modules will be summed. However, only the max value of the access time and cycle time will be included as I will assume that the 3 separate components (2level, gshare, selector) can be accessed in parallel.

Table 2: Latency, Power, and Area Statistics for the Branch Predictors

Statistic	2level	opened	Increase (%)
Access time (ps)	307.4	375.0	22.0
Cycle time (ps)	212.7	289.2	36.0
Total dynamic read energy per access (pJ)	0.6363	5.123	705
Total leakage power of a bank (mW)	0.2486	5.598	2150
Cache height x width (mm^2)	0.001385	0.02506	1710