1 Choice of Predictor

I picked a TAGE-like predictor since I heard it was used in many modern processors. TAGEs have also won the past few Championship Branch Prediction competitions. I implemented a newer version of TAGE called BATAGE from one of the authors of the TAGE paper (Pierre Michaud) for two main reasons. The first was that BATAGE purportedly offers slightly better performance at 32KB than TAGE. The second was that the paper was very clear about the changes it was proposing and seemed more straightforward to implement. The TAGE paper was a bit fuzzy on the exact details and seemed harder to implement.

1.1 Papers Referenced

- Michaud, Pierre. "An alternative tage-like conditional branch predictor." ACM Transactions on Architecture and Code Optimization (TACO) 15.3 (2018): 1-23.
- Seznec, André, and Pierre Michaud. "A case for (partially) TAgged GEometric history length branch prediction." The Journal of Instruction-Level Parallelism 8 (2006): 23.
- Michaud, Pierre. "A PPM-like, tag-based branch predictor." The Journal of Instruction-Level Parallelism 7 (2005): 10.
- Kessler, Richard E. "The alpha 21264 microprocessor." IEEE micro 19.2 (1999): 24-36.

2 Implementation Details

BATAGE uses the same general structure as TAGE. The main difference between BATAGE and TAGE is using separate up/down counters and removing the useful counter u. Conventional TAGE improves if you add a statistical corrector (TAGE-SC-L). BATAGE tries to remove the need for a corrector by storing separate counters for taken and not taken and then using Bayesian statistics to determine the confidence level of a prediction. When allocating new entries, BATAGE replaces low or medium confidence entries and decays high confidence entries instead of doing a psuedo-LRU scheme like in TAGE.

I made several modifications to the version of BATAGE presented in the paper in order improve performance and reduce complexity. BATAGE uses a controlled allocation throttling (CAT) mechanism reduce the number of allocations when the allocation is too "easy". In this situation the new allocation would just take up extra space and likely wouldn't be too useful. However, my implementation of BATAGE has a few flaws and adding CAT did not seem to help. This is likely due to the fact that I used 2 bit saturating up/down counters in the bimodal base predictor instead of 3 bit saturating counters. One consequence of this is that the base predictor does not have enough information to make high confidence predictions. This is probably limiting the performance of my predictor in the long run.

All predictors were sized to fully use up the available budget. The TAGE-like predictor has a 2048 entry bimodal base predictor where each entry is a 2 bit counter. It also has 4 additional tables each containing 512 14 bit entries (10 bit tag plus 2 two bit counters). This sums to a total of 32768 bits used by all tables. It also uses 175 bits of register data shared between the predictor and the updater.

3 Data and Results

Table 1: Misprediction Rates

	static	gshare	tournament	custom
fp_1	12.128	0.826	0.984	0.803
fp_2	42.350	1.355	0.406	1.142
int_1	44.136	12.199	9.605	8.745
int_2	5.508	0.401	0.326	0.294
mm_1	50.353	5.333	0.945	1.889
mm_2	37.045	9.094	6.705	6.345

Table 2: Resource Usage in Bits

	static	gshare	tournament	custom
table	0	32768	32768	32768
register	0	14	12	175