An Analysis of TAGE and L-TAGE Branch Predictors

Team Amdahl's Vandals

Outline

Introduction

Branch predictors:

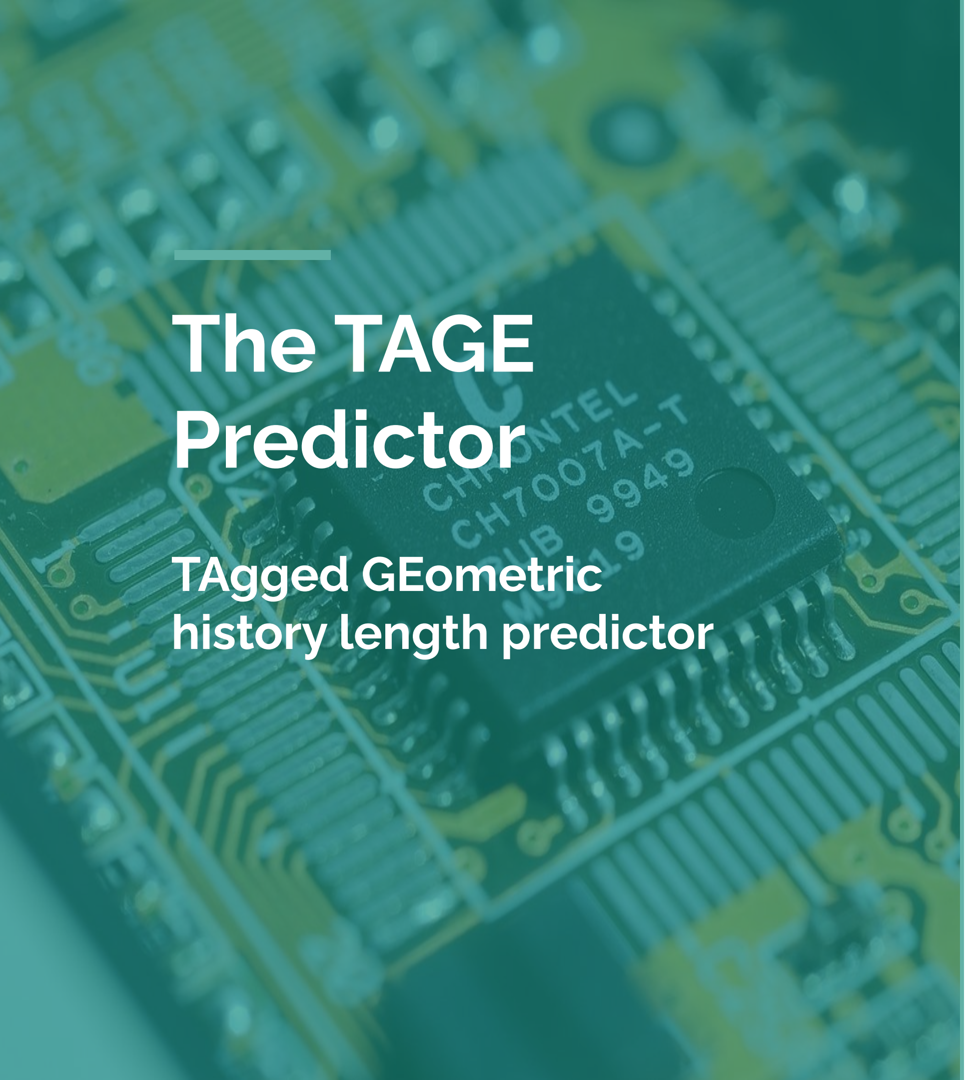
- Critical role in achieving effective performance
- 1% improvement in prediction accuracy can save millions of cycles in large programs

More efficient Branch Predictors?

- History repeats itself
- Multiple history lengths can capture correlations from both remote branch outcomes and recent history
- Compute final prediction using multiple isolated predictor components

More efficient Branch Predictors?

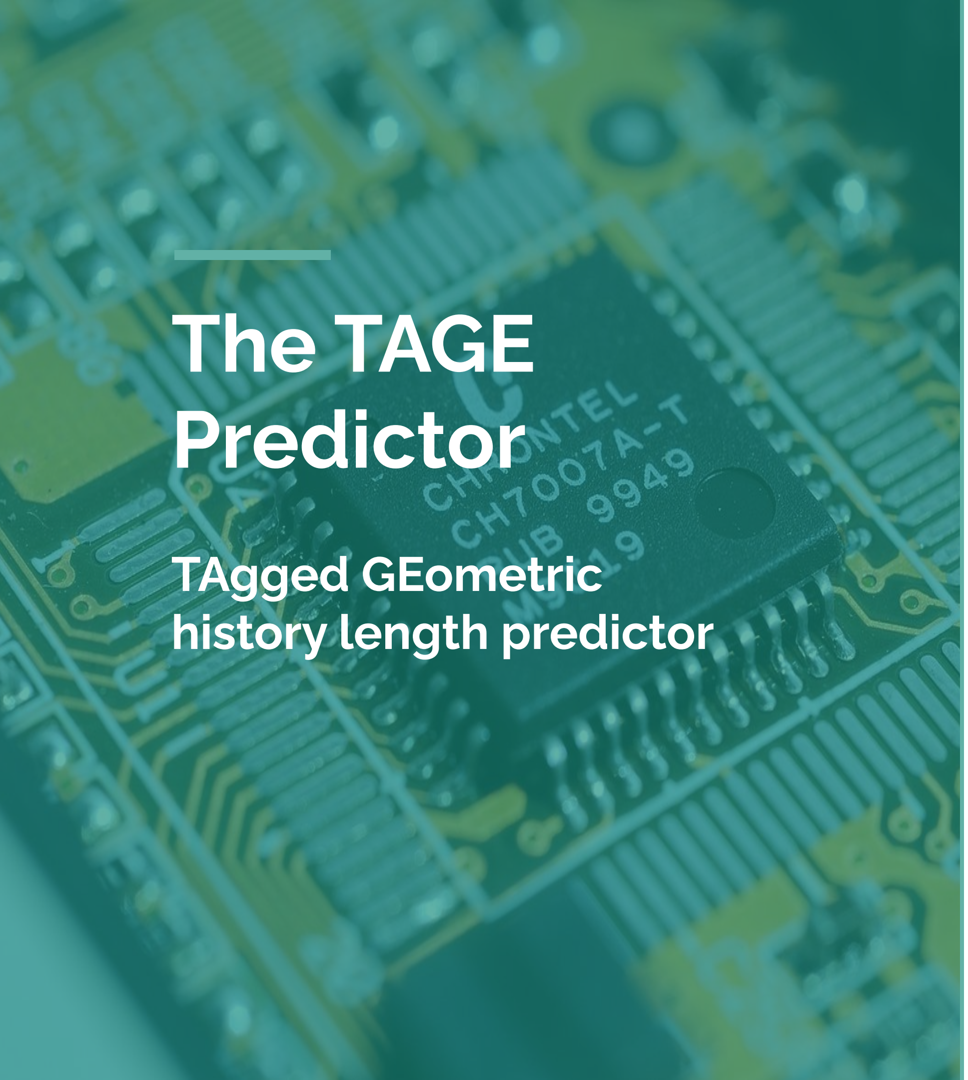
- **O-GEHL Predictor:** uses adder tree to combine predictions from multiple components
- Geometric series gives history lengths to be utilised
- Allows long history lengths to be exploited and recent branch outcome correlations to be captured



The TAGE Predictor

TAgged GEometric history length predictor

- Tagless bimodal base predictor + partially tagged components indexed using history lengths in GP
- Prediction by either base predictor or tag match on a tagged component
- Multiple hits?
  - If tag-matching table with longest history has strong prediction, use it
  - Else use tag-matching table with second-longest history (alternate prediction)



The TAGE Predictor

TAgged GEometric history length predictor

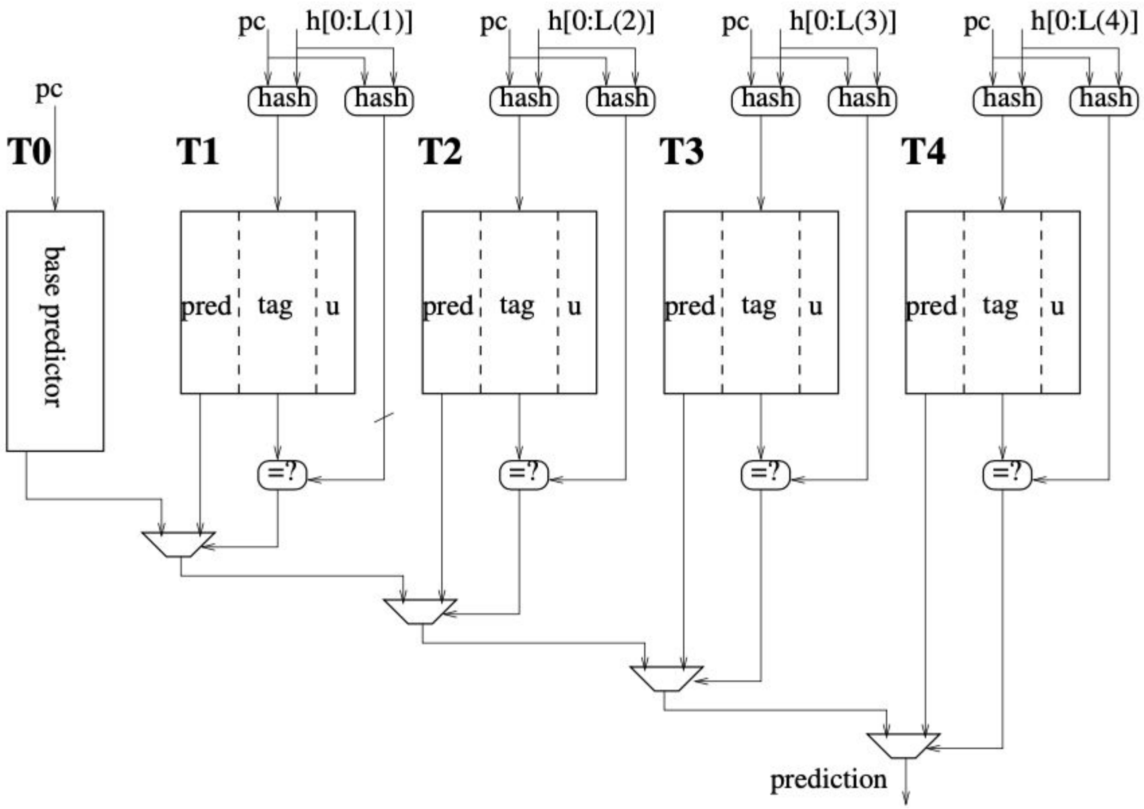
- New and efficient predictor update algorithm used
- Makes partial tagging more efficient than adder for final prediction computation
- Outperforms O-GEHL at equivalent storage budgets and predictor complexity

Principles and Explanation - 1k ft. view

- Geometric history lengths
- Update rules
- Rationale behind them

TAGE

A base predictor + several tagged predictor components indexed with increasing history lengths



Geometric History Length Prediction

- *M* distinct predictor tables
- Indexed with hash functions of branch address and global branch history
- Distinct history lengths used for computing index of distinct tables
- Base table T_0 indexed using branch address only
- For table T_i ($i = 1$ to $M\text{-}1$), history length:

$$L(i) = (int)(\alpha^{i-1} \times L(1) + 0.5)$$

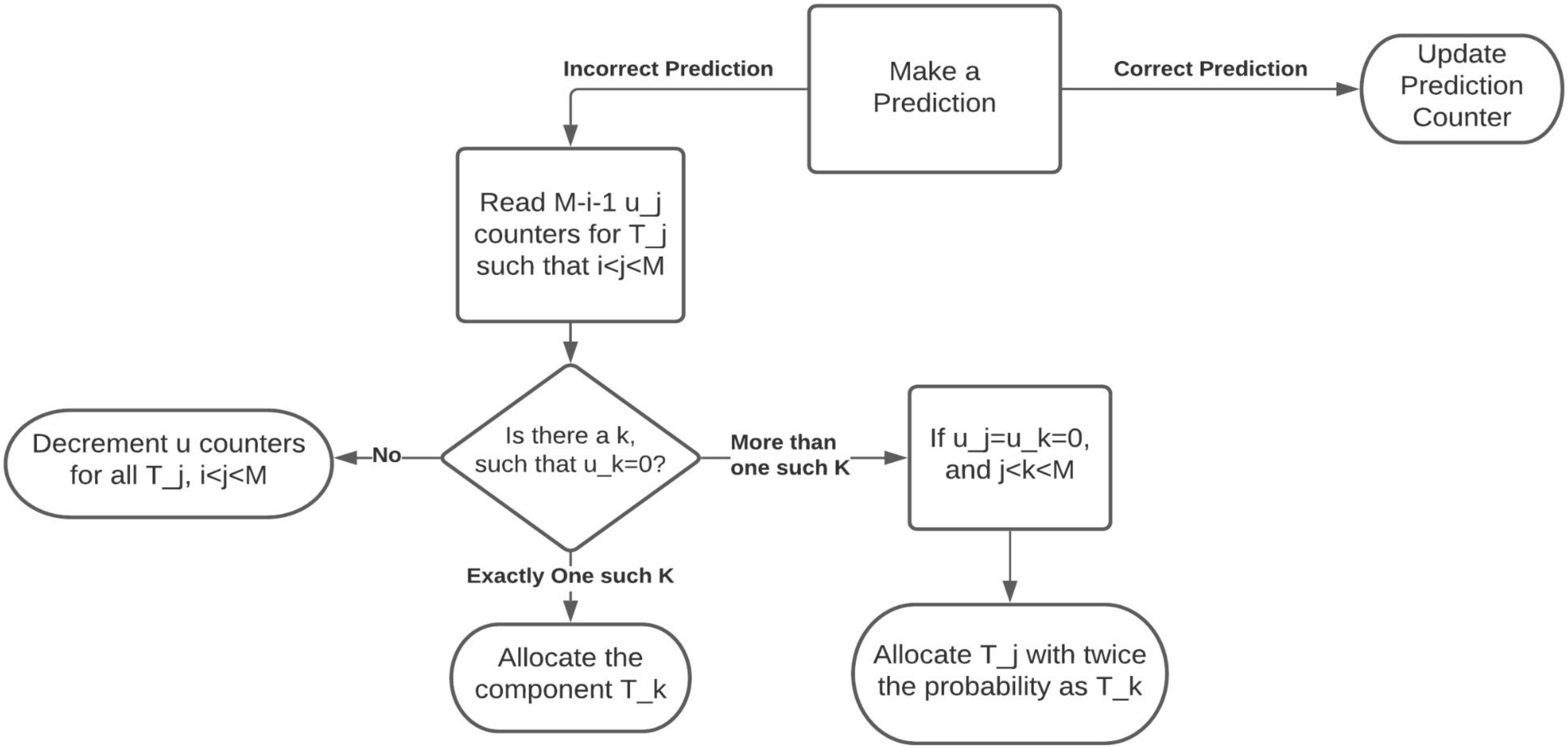
Update Policy

- Each tagged component entry has a useful counter *u*
- Updated when alternate prediction (second best) ≠ final prediction *pred*.

u incremented $\xleftarrow{\text{yes}}$ *pred* correct? $\xrightarrow{\text{no}}$ *u* decremented

- MSBs and LSBs of all *u* counters periodically reset to 0.

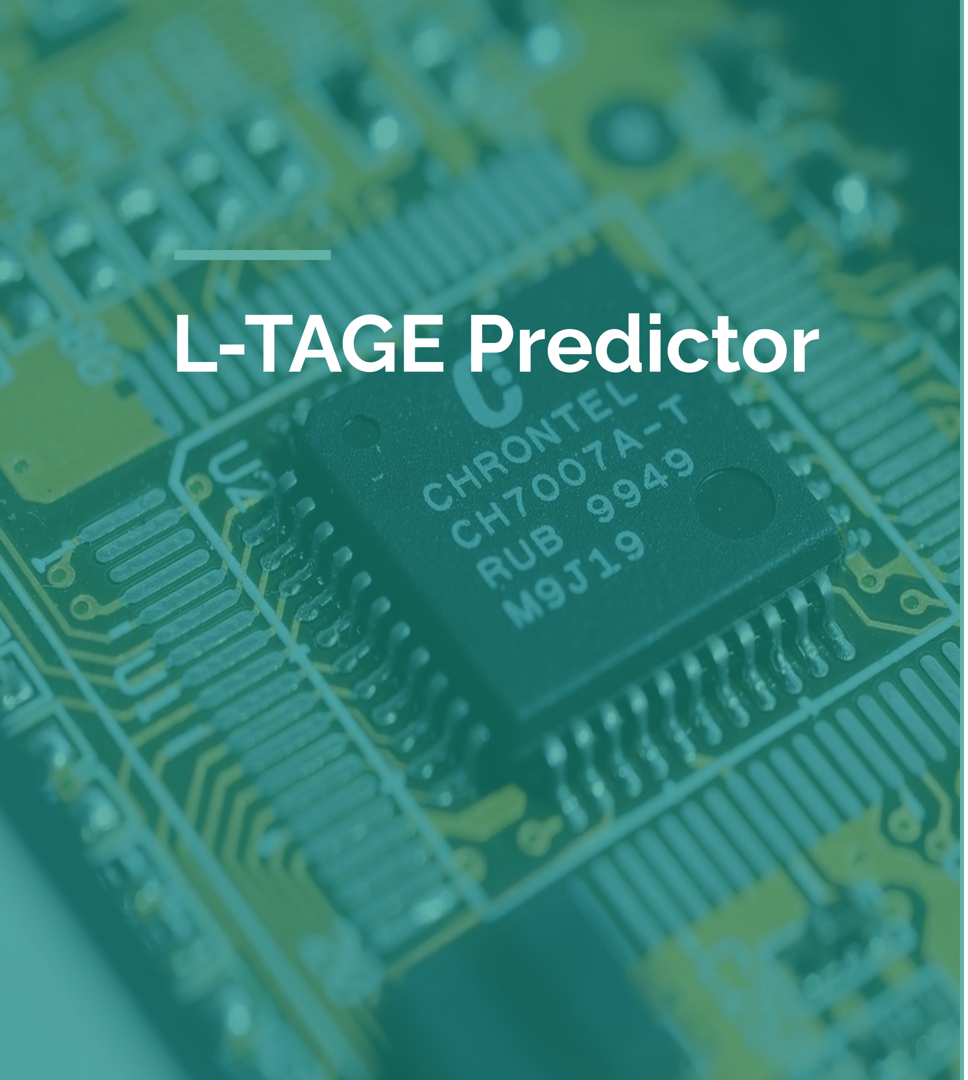
Update Policy



Why this update policy?

- ➔ Minimizes perturbation caused by single occurrence of a branch
- ➔ At most one tagged entry is allocated on a misprediction
- ➔ The useful counter *u* helps mimic a pseudo LRU policy

L-TAGE Predictor



- An X component TAGE predictor combined with a Y-entry loop predictor
- Loops are detected based on repeated access to the tables for branch prediction

Loop Predictor Component

- Identifies regular loops with constant number of iterations
- Provides global prediction when loop has been executed 3 times successively with same number of iterations
- Replacement policy is based on the age
- Used when confidence of loop prediction is high enough

Why loop prediction?

- ➔ Can capture regular loop behaviors, which TAGE cannot, using very limited storage
- ➔ Can predict the number of times a loop will iterate
- ➔ Allows several basic blocks to be prefetched accurately
- ➔ Reduces number of instruction cache and memory requests

So what did we do?

- We implemented TAGE and L-TAGE in ChampSim

  Find the code here: https://github.com/sudoRicheek/amdahls-vandals

  To see some sample results clone the repo and:
  - `Load the traces [server_001, server_002, server_003, server_004, server_009] into traces/`
  - `cd ChampSim/`
  - `./run_cases.sh`

The generated results can be found in `results_10M/`

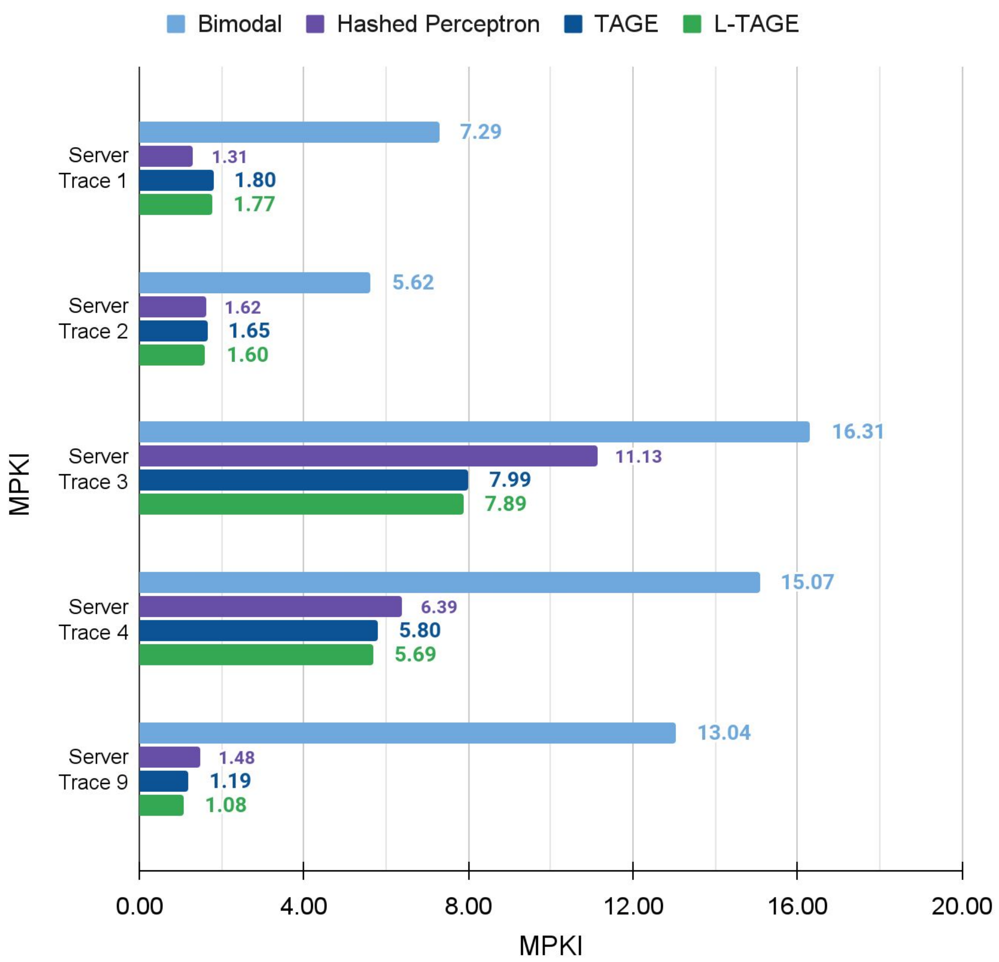
What did we observe?

- Comparing performance across predictors
- Varying number of components for TAGE and L-TAGE
- Varying history lengths
- Varying tag widths

Comparing Predictors

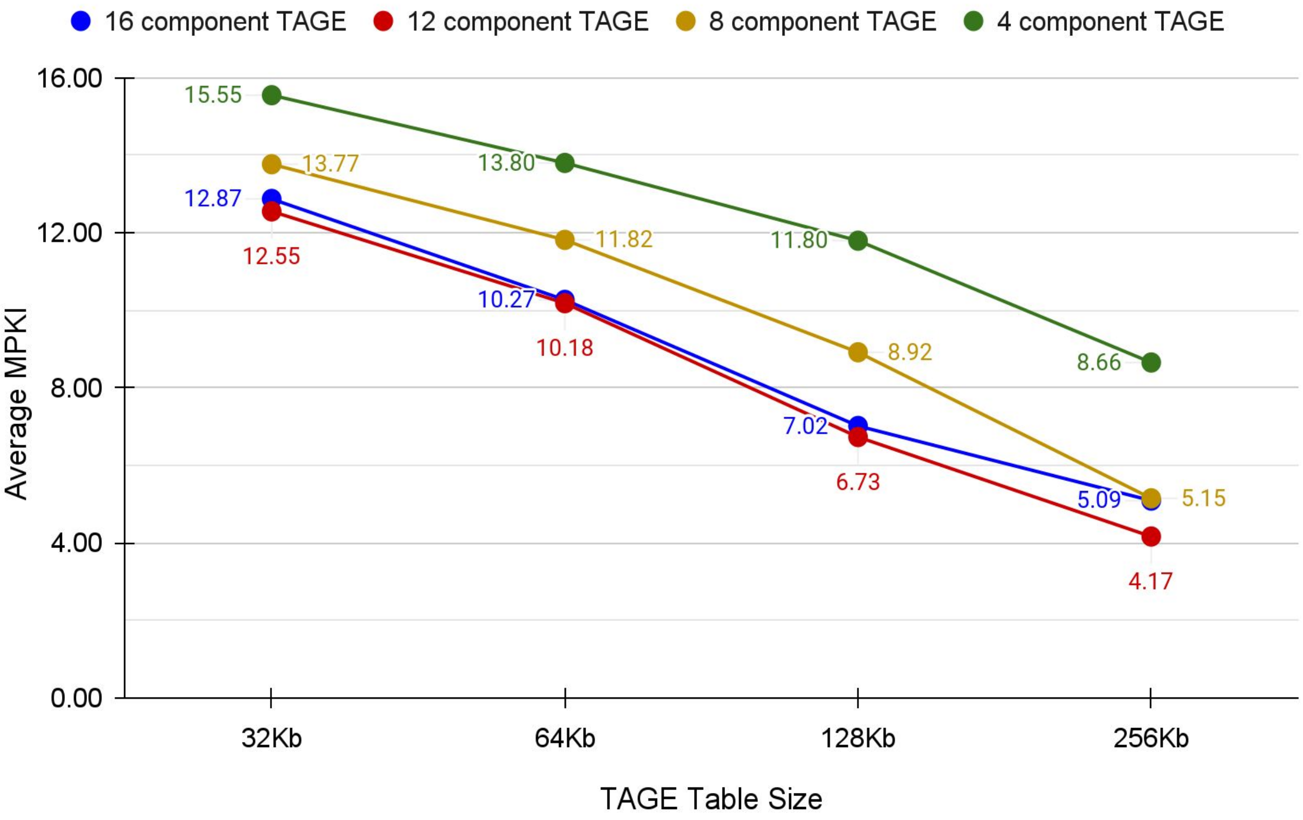
In all cases, our TAGE and L-TAGE perform better than Bimodal

In most cases, it also performs better than Hashed Perceptron



TAGE: Varying number of components

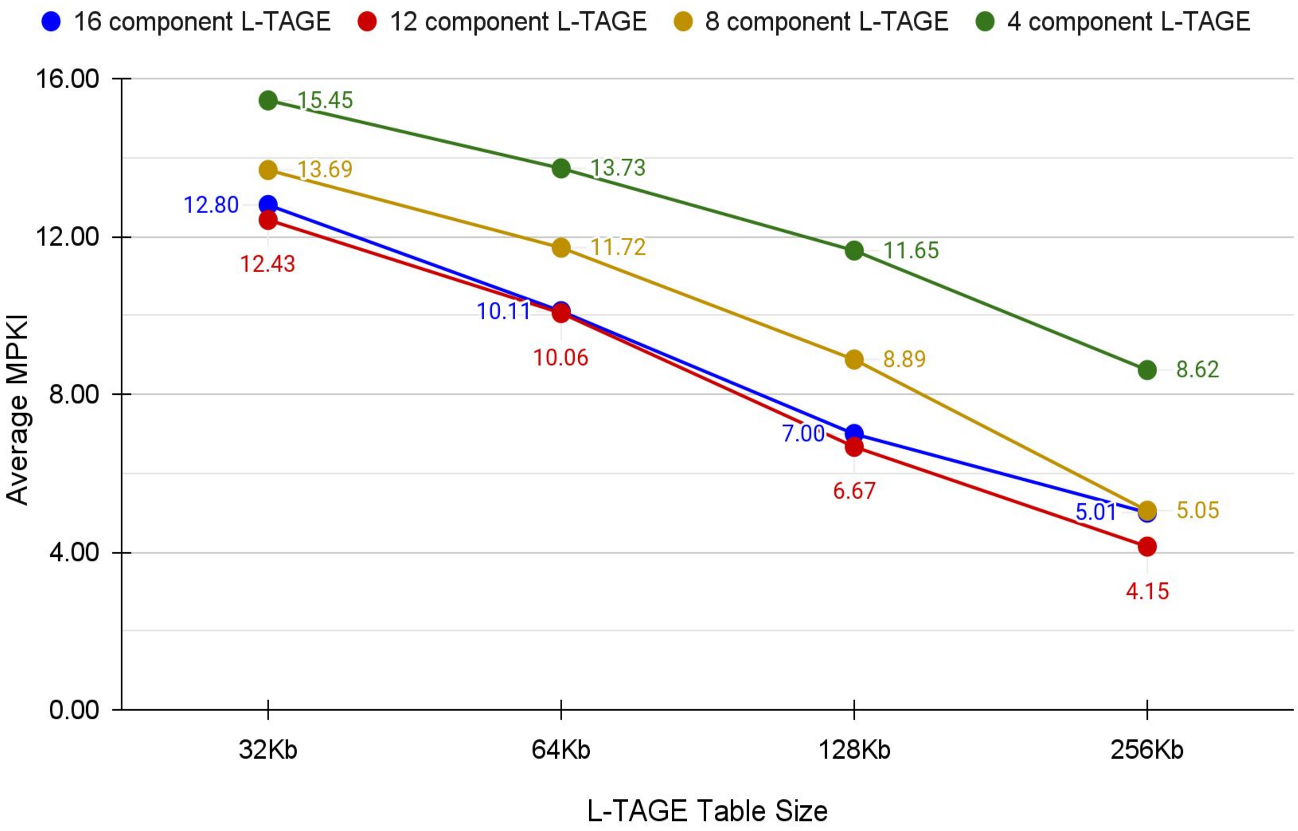
- On increasing Table Size, accuracy of all predictors increase
- On increasing components upto 12, there is significant decrease in MPKI
- On increasing components from 12 to 16, average MPKI begins to stagnate



MPKI average taken over 5 server caches

L-TAGE: Varying number of components

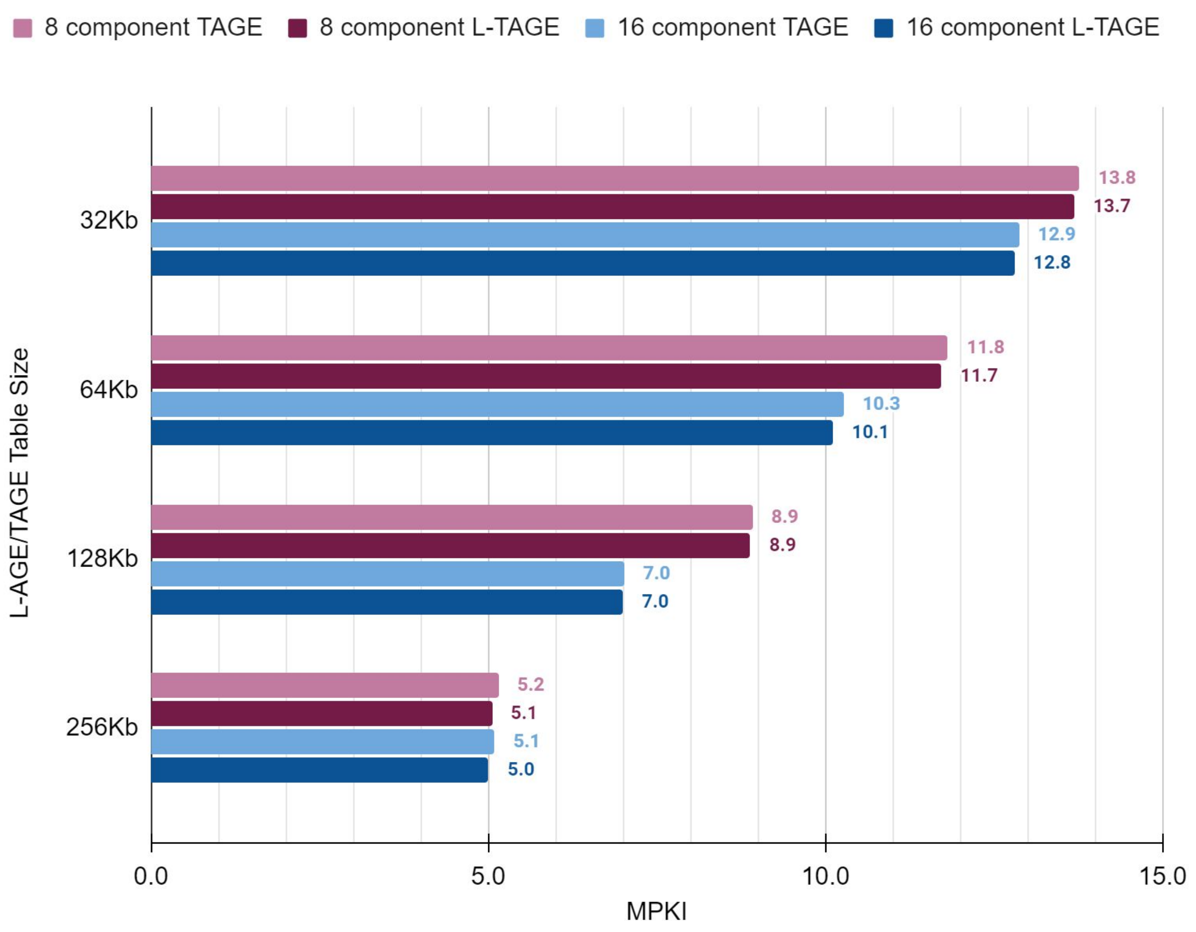
- Again, on increasing Table Size, accuracy of all predictors increase
- On increasing components upto 12, there is significant decrease in MPKI
- On increasing components from 12 to 16, MPKI begins to stagnate



MPKI average taken over 5 server caches

TAGE vs. L-TAGE

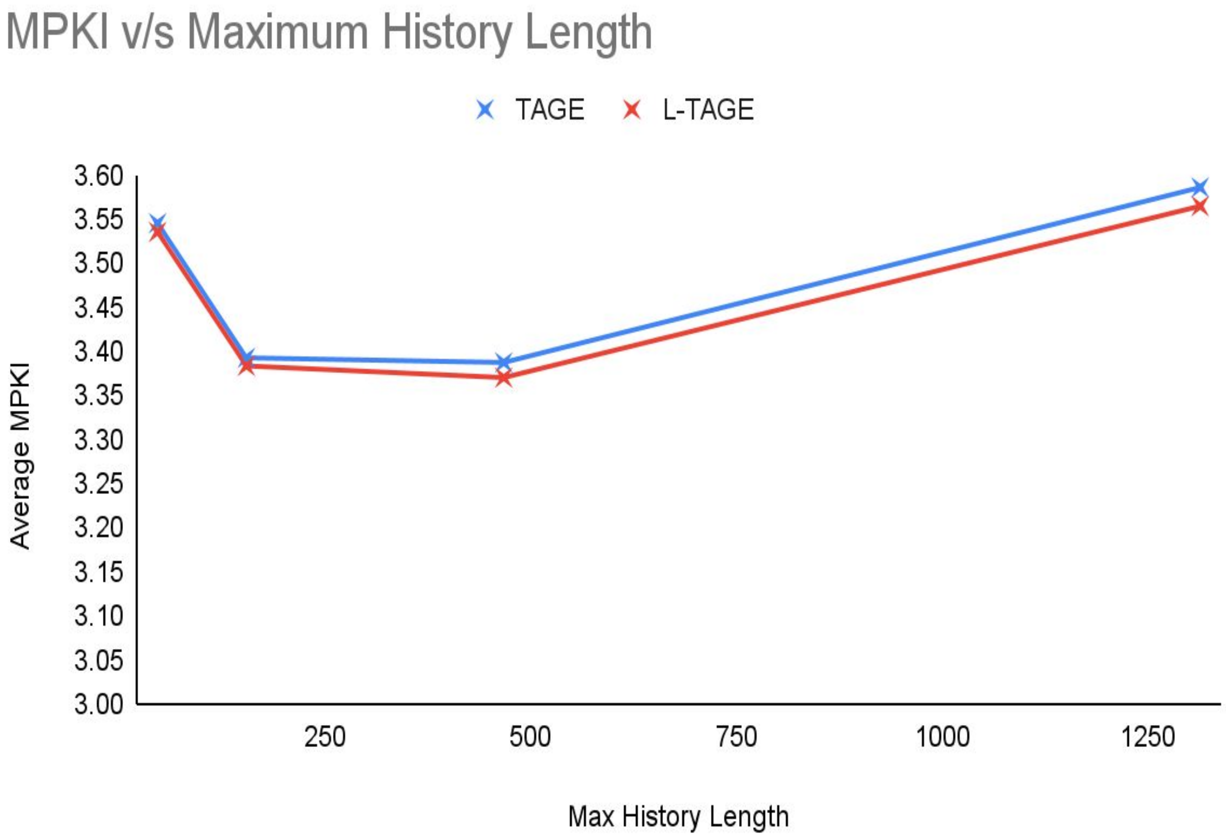
- We again see a clear trend of decreasing MPKI with increasing Table Size.
- A 16 table TAGE and L-TAGE performs much better than its 8 table counterpart.
- n-component L-TAGE performs better than n-component TAGE over all the tabulated Table sizes.



MPKI average taken over 5 server caches

Varying History Length

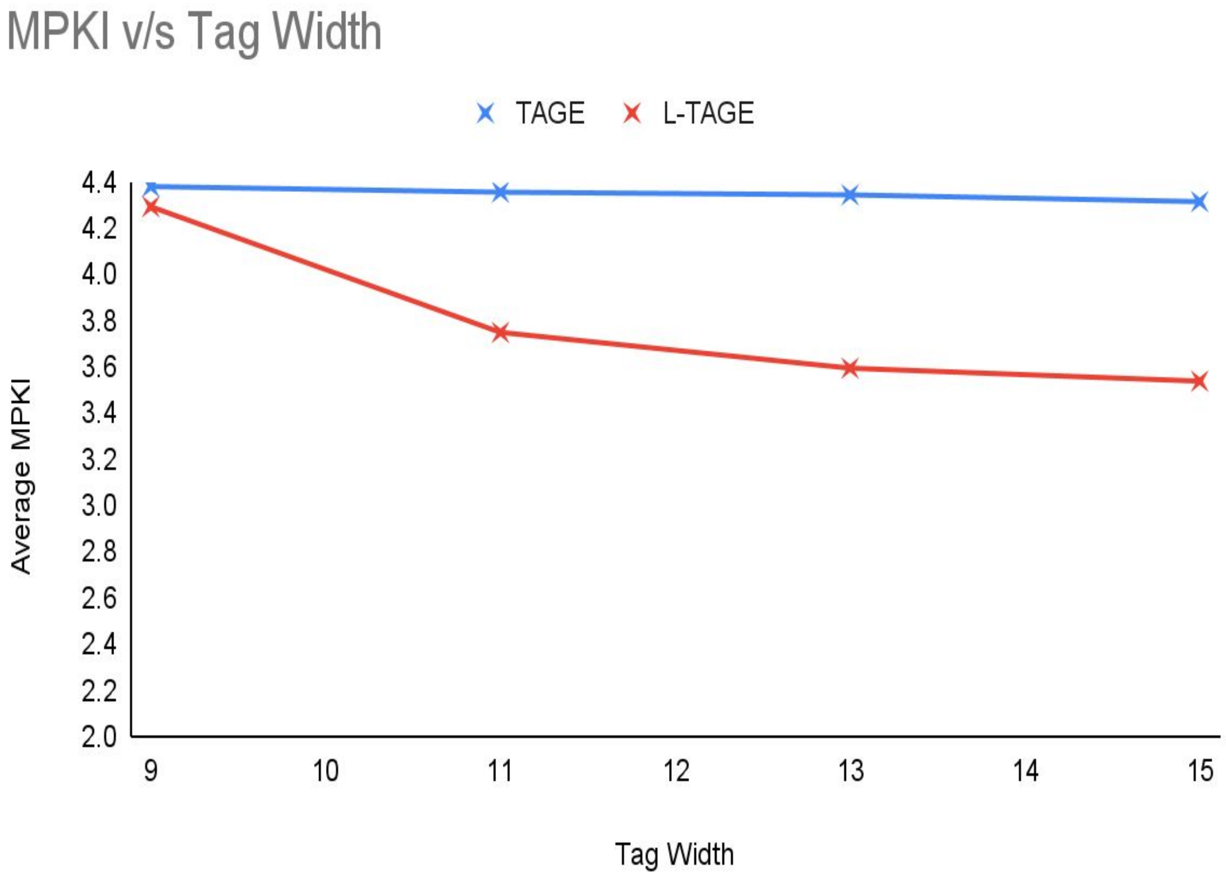
- With increase in history length, initially MPKI decreases (hence a larger history helps)
- As we increase further the maximum history length, MPKI stagnates and then increases marginally.
- TAGE and L-TAGE are robust to perturbations in history length.
- Marginal increase could be because server traces don't require very long history



MPKI average taken over 5 server caches

Varying Tag Width

- Using a larger tag width will need more storage for tables
- Too small tag widths will result in false tag matches, leading to mispredictions.
- Results show that an increase in tag width stagnates MPKI after a certain point



MPKI average taken over 5 server caches

What we learnt

Conclusion

- We first saw how TAGE greatly improved the efficacy of geometric length based branch predictors like O-GEHL
- We built an intuition to justify history lengths in GP, Update Rule, and the Loop Predictor in L-TAGE
- We finally present our code and show extensive results by studying each major component of L-TAGE and TAGE in detail.

Contributions

Roll No.	Name	Contribution
190260036	Richeek Das	Code, Inferences, Optimal Parameters
190050002	Aakriti	Graphs generation, Inferences, Results video
190050020	Ankit Kumar Misra	Code, TAGE theory in PPT, Results video
190050111	Shabnam Sahay	Theory, PPT formatting and editing, Theory video
190050119	Sumit Jain	TAGE Theory, Results, Graphs, Inferences

References

A case for (partially) tagged Geometric History Length Branch Prediction

The L-TAGE branch predictor

https://github.com/boubinjg/BranchPrediction

https://github.com/2Bor2C/tage

Championship Branch Prediction (CBP) 2016