Implementation of TAgged GEometric Branch Prediction on ChampSim

Sreedhar Reddy. P
Texas A&M University

Anirudh Raju. S
Texas A&M University

Madhava. Y
Texas A&M University

Abstract

The Tagged Geometric Branch Predictor (TAGE) provides state-of-the-art predictive accuracy for conditional branches, with equivalent storage budget outperforming all other predictors showcased at the Championship Branch Prediction in December 2004. It is one of the most commonly used branch predictor(BP), employing Geometric History Length similar to the O-GEHL predictor where history lengths are increased in Geometric series. This approach enhances prediction accuracy, as different branches often require varying history lengths. Additionally, it utilizes partially tagged components, similar to a PPM-like predictor. TAGE selects the most suitable partially tagged entry using a confidence counter known as usefulness, effectively mitigating aliasing issues.

1 Introduction

Conditional instructions usually cause the pipeline to stall in a traditional 5-stage pipeline architecture. This leads to wait for several cycles until the instruction's result is determined. This stalling will result in depreciation of the system performance. To reduce the depreciation and improve overall performance, Branch prediction techniques are employed. By anticipating the conditional instruction's execution path, these methods decrease the frequency and length of pipeline stalls and increase processor throughput. Branch predictors are critical components in a processor that guess the outcome of a branch instruction—specifically, whether a conditional jump will be taken or not taken. Based on this prediction, subsequent instructions are fetched and processed preemptively. If the prediction is correct, processing proceeds seamlessly. However, an incorrect prediction necessitates flushing these prefetched instructions from the pipeline, resulting in wasted computational resources. Therefore, enhancing the accuracy of branch predictions is crucial to minimize these inefficiencies and optimize processor performance. This is motivation for us to take the branch prediction project.

In this project, we implemented TAgged GEometric (TAGE) Branch prediction algorithm developed by André Seznec and Pierre Michaud. Notably, this algorithm has proven its effectiveness by winning two branch prediction championships.The implementation is done on the ChampSim simulator an open-source trace based simulator maintained at Texas A&M University [1]. The further sections consists of the related work of the branch prediction, implementation, algorithm of the TAGE predictor and finally we evaluated the TAGE by taking 8, 12, 16 tagged components and compared the efficient TAGE predictor with other prediction algorithms to put forth the effectiveness of TAGE.

2 Background and related works

There are two types of Branch prediction such as static branch prediction and dynamic prediction. Static Branch prediction decisions are made during compilation period where as Dynamic Branch prediction take decisions during the execution time. The fundamental principle behind branch prediction is the realization that results of branch instructions tend to follow repetitive patterns. By identifying and utilising these patterns, branch predictors can effectively anticipate the result of future branches. with this idea, there are multiple branch prediction alogirthms such as bimodal predictors, Gshare, perceptron based prediction etc. Bimodal prediction uses a table, where each entry consists of a signed counter which provides prediction based on its recent history. There are two-level adaptive branch predictors that utilise global history as well as local history. Gshare prediction technique hashes the global history and branch address which is received from program counter. This hash is used as index of the prediction table. Hashing is used to reduce the conflicts during the access of the table which improves the accuracy of the prediction. Perceptron based prediction techniques uses neural networks to observe the pattern and make predictions based on the patterns.

The fundamental concept behind the predictors like TAGE is using predictors which use different branch histories. The methods which make TAGE different from other predictors that use different history lengths are [3]

1. TAGE uses partial tag calculated from hashing of global history and program counter. This avoids multiple branch situations to point to the same entry of the table.
2. TAGE is able to more effectively balance warmup time versus history length which is because of increased precision of entry selection. To be more precise, TAGE does not have a high warmup time on branches with few branch scenarios that require a long history to be successfully predicted, in contrast to previous predictors.
3. TAGE prediction technique can make use of longer history lengths of branch because of above two features.

3 Methodology

The geometric series of history length is proposed from the OGEHL predictor [5], while TAGE is directly derived from the PPM-like predictor. Tables with comparatively short history length and indexes use the greatest amount of storage on a TAGE predictor, yet the predictor could still be able to capture correlation with very old branches. The prediction is given by either the default predictor or a tag match on a tagged predictor component. In the event of several hits in the tagged predictors, the forecast is provided by the predictor with the longest history length.

3.1 Implementation

The TAGE predictor is implemented with base prediction component T_0 and a group of tagged prediction components T_i which is illustrated in Figure 1.In general T_0 can be gshare or bimodal predictor etc. In our case we have considered T_0 as bimodal predictor with 2-bit counter. In T_i, i is an index used for the tagged components and these components are listed based on distinct history lengths that are derived from the below equation:

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

Each row of the tagged predictor table consists of three components such as useful counter *u*, signed counter *ctr* and a tag which is illustrated in Figure 2. The *ctr* depicts the prediction and *u* is used during the replacement. we considered *u* as 2-bit counter and *ctr* as 3-bit counter.

4 Steps of Prediction

Before going to discuss the algorithm, we will define some terms. The term "*provider component*" defines the table which has longest history length and matching tag. The term "*altpred*" is defined as the prediction that would have happened when there is a miss prediction by *provider component*. In an event where there is no hit on tagged predictors, the *altpred* is considered as default prediction.

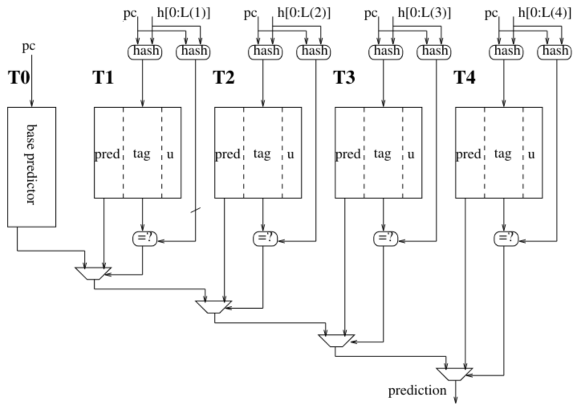
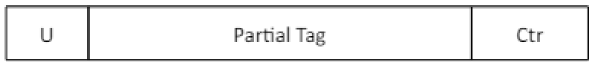


Figure 1: A 5-component TAGE predictor



U	Partial Tag	Ctr

Figure 2: A row of the tagged predictor table

4.1 Initialization

This phase of prediction is to initialise all the registers, prediction tables. It starts with initialization of Global history register (GHR) and Path history register (PHR). Next the base prediction table T_0 i.e., bimodal prediction table will be initialized with weakly taken prediction. Then initialise the tables of tagged predictors and Circular Shift Register which is used for GHR folding. TAGE does not utilize the complete GHR for tags within the predictor tables. To conserve storage, each GHR of a specific geometric length assigned to a tagged table is compressed into fewer bits. These compressed bits are then hashed together with selected bits from the incoming Program Counter (PC).The PC combines with the compressed histories to generate both the index and the tag for each TAGE table. The complete analysis of doing GHR folding can be referred from [4]. Once the tables are initialized, then there are two components of the prediction which is Branch Prediction stage and other stage where the table entries are updated based upon the result of the prediction.

4.2 Prediction

The prediction of branch whether it is taken or not taken is done in this stage. First the Index for the bimodal predictor table is calculated by taking the modulo of "PC" with the number of bimodal entries. Then calculate the tag of every tagged predictor by performing XOR operations. In our case, to calculate the tag of table we did PC $\oplus$ CSR1 $\oplus$ (CSR2 « 1)

[2]. These tags are masked to fit the bit width specified in Tag Size. Similarly, the indices for the TAGE tables are computed. Once the indices and tags are calculated, loop through tables to find the tables where the stored tag is matched with the calculated tag. In the event of several hits, the table with the longest history length is selected. In general The final prediction is done as Algorithm 1:

Algorithm 1 Prediction Algorithm

```
if $table$ is found then
    if $alternateTable$ is not found then
        $altpred \leftarrow bimodPred$
    else if $alternateTable$ is found then
        $altpred \leftarrow Taken|NotTaken$    ▷ based on counter
    end if
else if $table$ is not found then
    $pred \leftarrow bimodPred$
end if
```

In [6], It was mentioned that the prediction's level of trust is very low (typically less than 60%) when the provider component is tagged and the prediction is poor. The alternative prediction in this case is frequently more accurate than the provider component prediction. So, for upto **USE_ALT_ON_NA** times, if *altpred* is weakly taken or not taken or useful counter is zero, we use *altpred*. Otherwise we use prediction of the table with the longest history length

4.3 Update Results

This stage of the process is to update the tables and registers based on the result of the execution of branch instruction.

4.3.1 Update Counters

1. It starts with the updating the *ctr*. If there is tag hit on any tagged predictors (i.e., both provider component and also component that provided the *altpred*) and the outcome is taken, increase the *ctr* until it reaches maximum. If the outcome is not taken, decrease the *ctr*.

2. If there is no tag hit, increase or decrease the 2-bit counter of bimodal prediction table based upon the outcome.

3. Also we update the counter that monitors the accuracy of the *provider component* prediction. So, if the *provider component* entry is unused and prediction is weak, update the counter based upon *altpred*. If *altpred* is correct and not matching with *provider component*, increase the counter. Otherwise decrease the counter.

4.3.2 Allocate new entries

This process is done only if prediction is wrong.

1. First, find the component where u is zero.

2. If we do no find any component, decrease u in the entry of the every table which has index where the calculated tag should be present.

3. Otherwise, we randomly choose a table which has zero u and update the entry with calculated tag, update the *ctr* with weakly taken or not taken based on the outcome.

4.3.3 update *u*

If the prediction provided by the *provider component* is correct, increase u, otherwise decrease u. From time to time, the entire MSB column of the u is reset to zero, Then next cycle the entire LSB column is reset to zero. In our case, we have periodicity of 2^{20} branches. After all these steps, we do GHR folding and update the path history register.

5 Results and Discussion

The methodology discussed is implemented in the simulator "ChampSim" to find the effectiveness of the TAGE with bimodal predictor as base predictor. In order to evaluate the effectiveness, we have done following experiments:

1. Increase the number of tagged components and find the best number that provide high performance with the given memory budget.

2. Compare the efficient TAGE implementation with other branch prediction algorithms.

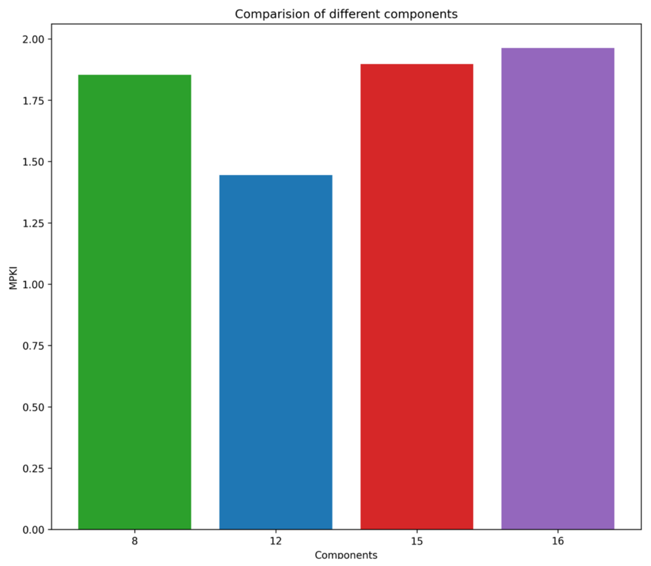


Figure 3: Comparison of MPKI for TAGE with different number of tagged components

Characteristic	**Base**	**T1-2**	**T3-4**	**T5**	**T6**	**T7**	**T8-9**	**T10**	**T11**	**T12**
Predictor bits	2	3	3	3	3	3	3	3	3	3
Useful bit	-	2	2	2	2	2	2	2	2	2
Tag Width	-	7	8	9	10	11	12	13	14	15
No of Entries in K	8	1	2	2	2	1	1	1	0.5	0.5
Total Storage in K Bits	16	12	26	28	30	16	17	18	9.5	10

Table 1: *12 tagged component TAGE predictor Characteristics*

5.1 Experiment - 1

In our project, we conducted different simulation trails using configurations with 8, 12, 15, and 16 components, each with varying tag sizes, table sizes, and history lengths to optimize the configuration while adhering to a storage constraint of 256 Kbits.

For the **8-component** configuration with $\alpha = 2$, we utilized the following parameters:

- Geo History Length: $\{4, 9, 17, 33, 65, 129, 257, 513\}$
- Tage Table Size: $\{10, 10, 11, 11, 11, 11, 10, 10\}$
- Tage Tag Size: $\{9, 9, 12, 12, 14, 16, 18, 20\}$

For the **12-component** configuration, we utilized the following parameters:

- Geo Hist Len: $\{4, 6, 10, 16, 25, 40, 64, 101, 160, 254, 403, 640\}$
- Tage Table Size: $\{10, 10, 11, 11, 11, 11, 10, 10, 10, 10, 9, 9\}$
- Tage Tag Size: $\{7, 7, 8, 8, 9, 10, 11, 12, 12, 13, 14, 15\}$

For the **15-component** configuration, we utilized the following parameters:

- Geo Hist Len: $\{6, 10, 18, 25, 35, 55, 69, 105, 155, 230, 354, 479, 642, 1012, 1347\}$
- Tage Table Size: $\{10, 10, 10, 11, 10, 10, 10, 10, 10, 9, 9, 9, 8, 7, 7\}$
- Tage Tag Size: $\{7, 9, 9, 9, 10, 11, 11, 12, 12, 12, 13, 14, 15, 15, 15\}$

For the **16-component** configuration, we utilized the following parameters:

- Geo Hist Len: $\{2, 3, 8, 12, 17, 33, 35, 67, 97, 138, 195, 330, 517, 1193, 1741, 1930\}$
- Tage Table Size: $\{10, 10, 10, 11, 10, 10, 10, 10, 10, 9, 9, 9, 8, 7, 7, 7\}$
- Tage Tag Size: $\{7, 9, 9, 9, 10, 11, 11, 12, 12, 12, 13, 14, 15, 15, 15, 15\}$

It has been observed that the Arithmetic mean of MPKI for 12-component TAGE predictor is less than other number components for the taken memory budget which is depicted in Figure 3. So, we have taken 12- tagged component TAGE results to be compared with other branch prediction techniques. we have done one more trail with more budget size i.e., 450Kb with 16 tagged component TAGE and results of this trail is mentioned in more plots section in last page.

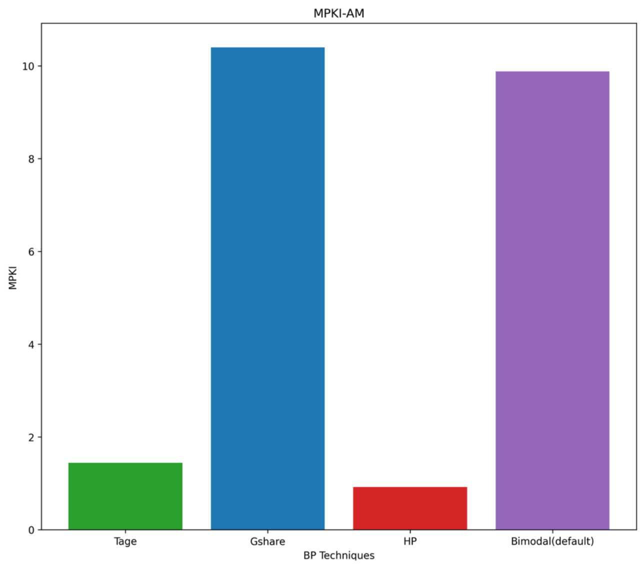


Figure 4: MPKI-AM of different branch prediction techniques

BP Technique	**Avg MPKI Value**
Gshare	10.40
TAGE	1.44
Bimodal	9.88
Hashed Perceptron	0.919

Table 2: *Average MPKI of different Branch Prediction technique*

5.2 Experiment - 2

In this experiment, we compare the 12- tagged component TAGE with other branch prediction methods to evaluate the

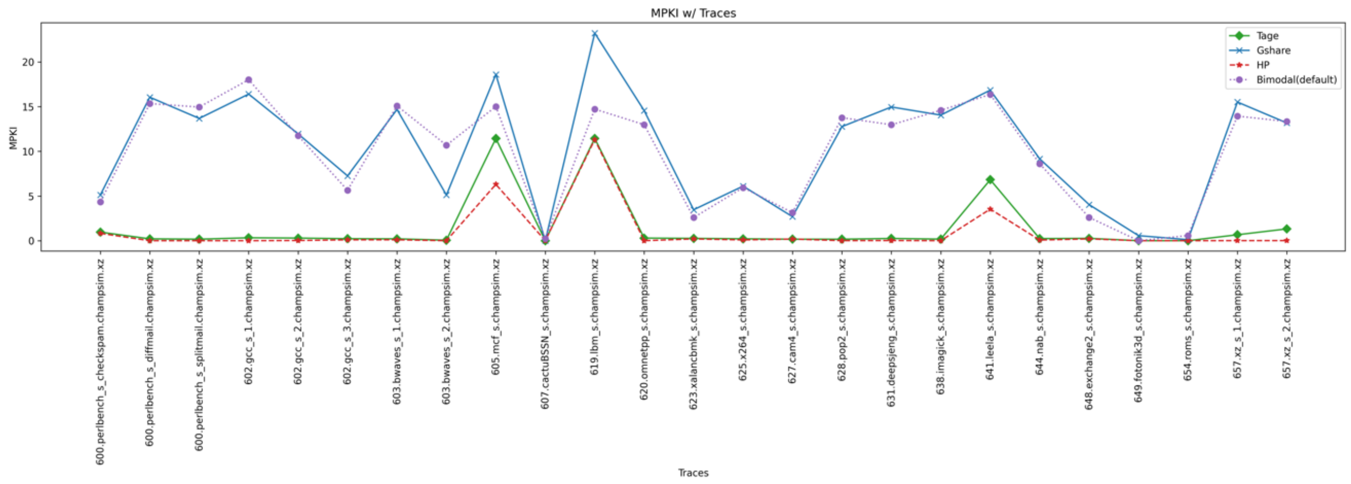


Figure 5: MPKI w/ Traces

effectiveness of TAGE. We have used default configured bimodal, Gshare, hashed perceptron implementations.

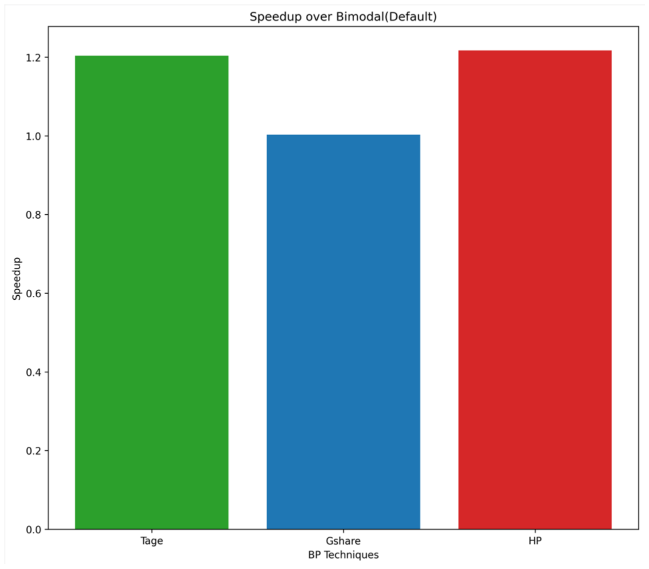


Figure 6: Speedup over Bimodal branch prediction

we had run the simulations and compared the results of 12-TAGE with the results of other branch prediction techniques. The Average MPKI values of each branch prediction technique is mentioned in Table 2 and depicted in Figure 4. It has been observed that TAGE achieves a significantly lower average MPKI compared to Gshare and Bimodal prediction techniques, while closely approaching the performance of the Hashed Perceptron for most traces, indicating comparable performance across a wide range of scenarios. It can be further improved by optimizing the Geometric history lengths, Tag size and Table sizes. Figure 6 illustrates the variation in MPKI across different traces. It reveals that, except for three traces *605.mcf_s.champsim.xz, 641.leela_s.champsim.xz, 657.xz_s_2.champsim.xz*, the results are closely aligned with those of Hashed Perceptron. Furthermore, there is a noticeable difference among traces when compared with both Bimodal and Gshare predictors.

The speedup achieved by *TAGE* over *Bimodal (default)* is 1.20, surpassing Gshare's 1.002, and closely trailing the performance of the Hashed perceptron, which achieved a speedup of 1.21.. It is evident from the Average MPKI graphs that TAGE Branch prediction outperforms the Branch prediction techniques such as Gshare, bimodal and almost resembles the Hashed perceptron. The average MPKI for *Bimodal (default)* is *9.88*, whereas for *TAGE* it is *1.45*, which is a significant difference and is almost near to that of *Hashed perceptron*, whose value is *0.92*. It can be further improved by optimizing the Geometric history lengths, Tag size and Table sizes.

6 Conclusion

In this project, we implemented and optimized TAGE by adjusting the geometric history length using various α values in the equation, as well as by exploring different tag sizes and table sizes to minimize the MPKI (Misses Per Kilo Instructions) for improved performance. Compared to default predictor, with TAGE we've achieved 85.25% reduction in misprediction. Additionally, by including the Loop predictor alongside the base predictor could further enhance performance. The insights gained from this project underscore the trade-off between accuracy and the demands on storage budget and complexity. Furthermore, the inclusion of a meta-predictor emerges as a crucial component in the hybrid predictor, highlighting the need for additional refinement. To enhance the project, it is essential to develop an efficient meta-predictor capable of selecting the optimal predictor among various Branch Predictions, including loop predictors, as well as more efficient ones such as perceptron or neural-based predictors.

References

[1] Gober, N., Chacon, G., Wang, L., Gratz, P. V., Jimenez, D. A., Teran, E., Pugsley, S., and Kim, J. The Championship Simulator: Architectural Simulation for Education and Competition.

[2] Michaud, P. A ppm-like, tag-based branch predictor. *The Journal of Instruction-Level Parallelism* (2005), pp.10.

[3] Miftakhutdinov, R. Why tage is the best (website).

[4] Schlais, D. J., and Lipasti, M. H. Badgr: A practical ghr implementation for tage branch predictors. *2016 IEEE 34th International Conference on Computer Design (ICCD)* (2016), 536–543.

[5] Seznec, A. The o-gehl branch predictor.

[6] Seznec, A. A new case for the tage branch predictor. *MICRO 2011 : The 44th Annual IEEE/ACM International Symposium on Microarchitecture* (2011).

7 More Plots

Please refer to the next page

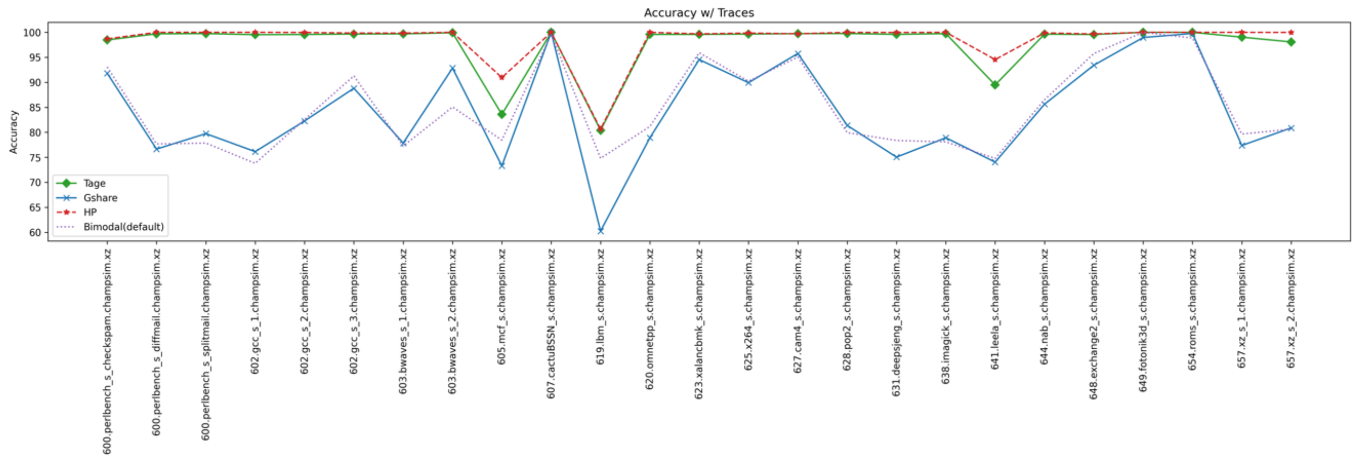


Figure 7: Branch Prediction Accuracy of each branch prediction technique for the given trace

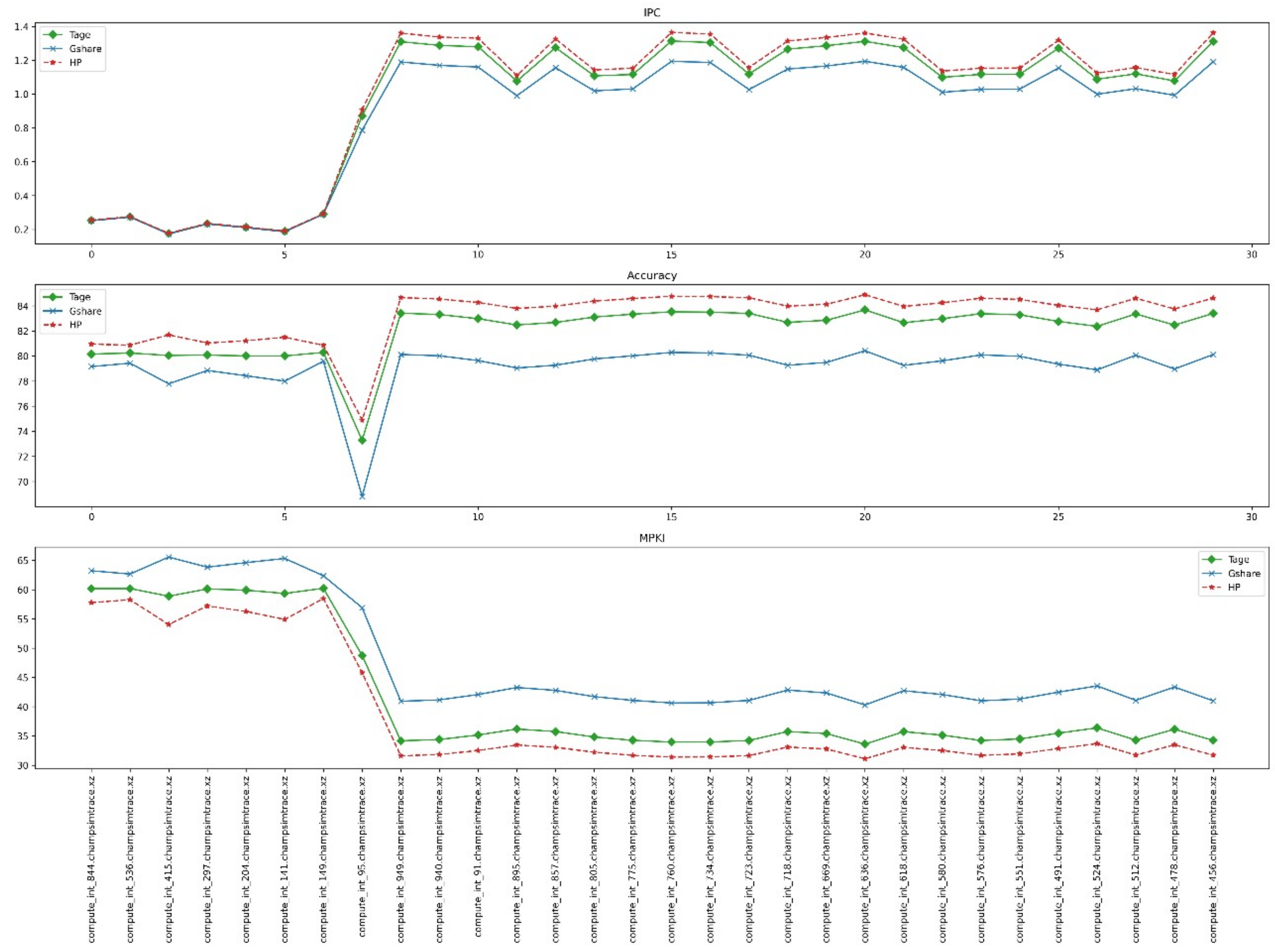


Figure 8: 16 tagged component TAGE with 450Kb memory budget effectiveness