A Modular Branch Predictor Performance Analysis Framework for Fast Design Space Exploration

Ya Wang*, Hanwei Fan*, Sicheng Li†, Tingyuan Liang*, Wei Zhang*
*The Hong Kong University of Science and Technology
†Alibaba DAMO Academy
*{ywangmu,hfanah,tliang,wei.zhang}@ust.hk †sicheng.li@alibaba-inc.com

***Abstract*—As modern processor designs scale up and workloads become more complex, the selection of the branch predictor (BP) and the optimization of its internal parameters are increasingly critical in striking a balance between performance and resource usage. However, current fast performance evaluation models and micro-architectural Design Space Exploration (DSE) frameworks provide limited support for BP components, especially regarding internal parameter adjustments. In this work, we propose a modular BP performance analysis framework that provides fast performance feedback for different BP configurations. Our framework includes a pattern analyzer equipped with more accurate metrics for quantifying an application's predictability of branch behavior, a classification module for selecting the appropriate BP type, and an analytical model set that reflects the impact of internal parameter adjustments of various BPs on both performance and storage resource usage, thereby supporting DSE. Experimental results on three benchmarks confirm the framework's effectiveness, as our proposed model exhibits better correlation while reflecting more parameter changes than previous work. To the best of our knowledge, this is the first analytical model framework that supports comprehensive BP type and parameter adjustments.**

***Index Terms*—Branch Prediction, Analytical Models, DSE**

I. Introduction

As processor designs scale up, the micro-architectural design space correspondingly expands, leading to an increasingly complex verification process. Under such circumstances, efficient design space exploration frameworks and swift design evaluation models become indispensable for micro-architectural designs, particularly for custom designs tailored to specific workloads. Unfortunately, existing frameworks and models do not offer effective support for Branch Predictor(BP) which is a crucial frontend component that maintains high processor performance.

The BP is essential for guiding the pipeline front end, predicting subsequent instructions to maintain pipeline utilization and instruction throughput. Given the performance ramifications of mispredictions, which trigger pipeline flushes, and the significant storage demands of BP tables, the BP is vital for optimizing processor performance within limited memory constraints.

Despite its significance, current micro-architectural DSE frameworks[3, 6, 10, 11] fail to incorporate the internal hardware parameters of branch predictors into their design space. This omission overlooks the huge potential search space presented by current complex multi-table BPs. While specific DSE work for branch predictor[17] involves only one type of BP and resorts to time-consuming simulations for evaluation.

Current methods for expedited micro-architecture design evaluation predominantly utilize analytical models. These models employ formulas to quickly estimate performance based on application characteristics, offering a speed advantage over simulations. Nevertheless, they often fall short in accurately assessing the BP component, a significant source of error[15]. In these models, BP's effect is approximated by the product of the misprediction rate and the associated cycle penalty. While the penalty for mispredictions has been well-documented[5], accurately estimating the misprediction rate remains a substantial challenge. Certain models[10] derive this rate from behavior simulations, which are not only time-intensive but also necessitate repetition for each new design point, rendering them inefficient for extensive design spaces. Conversely, other models[13, 15, 16] predict the misprediction rate using micro-architecture independent metrics, neglecting the effects of BP parameter variations.

To address these problems, we propose a modular branch predictor performance analysis framework to provide fast performance evaluation for BP configurations and help to achieve a trade-off between misprediction rate and storage resource. Our framework enhances the processor analytical model to support BP internal parameter adjustments and expands the design space of micro-architecture, allowing us to undertake a more comprehensive exploration. In particular, we present several novel contributions:

- We propose a pattern analyzer to systematically analyze branch traces, offering a series of accurate and effective metrics designed to quantify the regularity and predictability of an application's branch behavior.
- We design a classification module to select the suitable BP type for applications, where several effective criteria are proposed to evaluate the compatibility between applications and different branch predictors
- We establish a well-correlated analytical model from a new perspective, which can comprehensively reflect the changes in BP parameters. To the best of our knowledge, this is the first attempt at establishing analytical models for the complex multi-table BP.
- We conduct experiments on three mainstream benchmark sets, demonstrating that our model can effectively support DSE for BPs.

979-8-3503-4859-0/DATE24/© 2024 EDAA

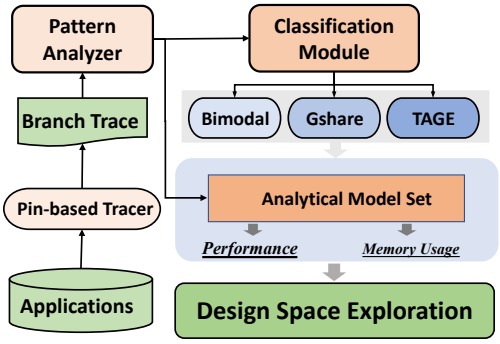

Fig. 1: Overview of the Branch Predictor Analysis Framework

II. Modular BP Analysis Framework

Our proposed framework is depicted in Fig. 1. Initially, we implemented a tracer based on the Intel Pin Tool[12] to extract the branch trace from binary executable files. Subsequently, the pattern analyzer processes this branch trace and gives accurate and effective metrics to evaluate complexity and predictability comprehensively. The next step involves the classification module, which uses these features to categorize applications and select the most suitable BP for each. This module quantifies the expertise of branch predictors and automates the analysis process. Moreover, we have developed analytical models for BPs that require hardware parameter modifications to evaluate performance and memory usage for targeted applications, enabling our framework to effectively support DSE under various constraints. As the input to our framework is fundamentally a branch trace, it allows for the free combination of different applications. This flexibility facilitates the design of BP in processors with various objectives, encompassing application-specific, domain-specific, and even more generic designs.

A. Pattern Analyzer

We implemented a pattern analyzer to comprehensively assess the predictability of the application's branch trace and provide accurate and efficient metrics for subsequent modules. Following a structured strategy from abstract to concrete, we first introduce **transition entropy**, a new entropy-based metric to quantify global trace regularity, which is regarded as a fundamental feature of branch predictability. Because the overall regularity is the most important factor affecting the difficulty of predicting applications. Then, to further identify factors causing branch miss prediction, our analyzer examines outliers in the outcome trace through **Matrix Profile** analysis and considers branch aliasing effects through **Hash Distribution** analysis. This structured, multi-scale approach, ranging from global to local and macro to micro, offers valuable insights for enhancing branch prediction performance.

1) Transition Entropy Analysis: The fundamental concept behind entropy analysis is to treat the branch trace as a signal sequence and apply the concept of information entropy to quantify the information density of the sequence. There are some existing entropy-based metrics such as branch entropy[16] and linear entropy[13], but branch entropy focuses only on the occurrence frequency of patterns and linear entropy focuses on taken rate corresponding to specific patterns, both of which fail to fully represent the regularity of the trace. Therefore, we propose a new and more accurate metric called transition entropy, which simultaneously considers the bias and switch of the outcome. For a binary sequence of fixed length, it can be expressed as follows:

$$TE(p, tr) = tr \cdot abs(p - 0.5) \tag{1}$$

Here, p represents the taken rate of the sequence, and tr denotes the transition rate. It is easy to calculate, and we find that the miss rate closely approximates this entropy, demonstrating a strong correlation. The intuition behind this is that a branch predictor normally favors outcomes that occur frequently and consistently. On the other hand, the original information entropy reflects the number of bits needed to represent a certain amount of information, which is conceptually more complex than branch prediction. Therefore, our metrics not only simplify the computation but also make it more suitable for Branch Prediction.

When assessing predictability, we collect the taken and transition rates of outcomes (jump or not) encountered under different lengths of history patterns using a sliding window approach. We calculate the transition entropy value for the branch trace as follows:

$$TE(h) = \frac{1}{L} \sum_{i}^{n} l_i \cdot TE(p_i, tr_i) \tag{2}$$

Where h is the history pattern length, L is the total trace length, and n represents the variety of all patterns of length l appearing in the trace. l_i, p_i, and tr_i correspond to the length of the outcome trace, the taken rate, and the transition rate for each pattern, respectively. For an application, we evaluate the global trace and the local traces corresponding to different PCs in parallel and provide the global entropy GTE(h) and local entropy LTE(h).

2) Matrix Profile Analysis: To capture compulsory misses in the trace, which can be caused by long loops or sudden interruptions and manifest as outliers deviating from the regular pattern in the trace, we conducted a hamming distance-based Matrix Profile analysis, which can be expressed as the Algorithm 1. The Hamming distance calculates the number of different bits between two equal-length binary sequences, effectively measuring their dissimilarity.

The Matrix Profile constructs a matrix profile vector by identifying the most similar subsequences of a specified length within the global trace. Analyzing the matrix profile vector can reveal critical characteristics of time series data. Just as the example in Fig.2, where the window length l is set to 128, representing the maximum value in our defined design space for the Global History Register (GHR), local minima in the matrix profile denote highly similar sub-sequences, which likely correspond to repetitive patterns in the data. In contrast, larger values may signify discords or changes in trend. Repetitive patterns are branches that can be easily predicted in branch prediction, while discords or abrupt shifts

are difficult to predict and can lead to prediction misses. To calculate the Compulsory Miss Rate (CMR), we sum up the intervals producing discords and then divide by the total branch trace length.

Algorithm 1 Matrix Profile with Hamming Distance

Input: Branch Trace: T, Length of subsequence: l
Output: Matrix Profile: M

1: **for** i, j in all distinct pairs from 0 to $len(T) - l$ **do**
2: $\quad d = Hamming(T[i : i + l], T[j : j + l])$
3: $\quad min\_d[i] = \min(min\_d[i], d)$
4: **end for**
5: Append min_d to M
6: **return** M

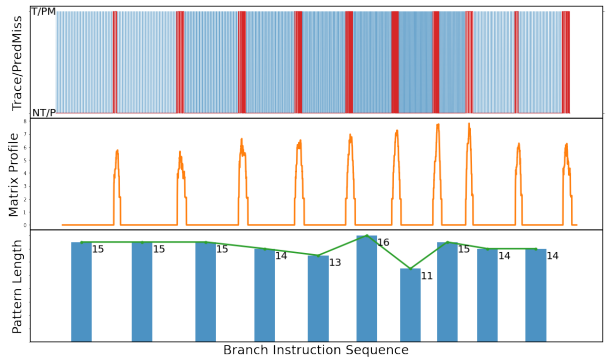

Fig. 2: An example of matrix profile analysis. Three subfigures display the positions of mispredictions, the derived matrix profile vector, and the lengths of the corresponding rule patterns

3) Hash Distribution Analysis: We further analyzed the potential prediction accuracy loss caused by hash collisions, which can bring branch aliasing. In branch prediction, the hash function is used to map the combination of the PC and branch history to a specific position in the prediction table. When hash collisions occur, it's possible for two unrelated branches to end up at the same location, causing them to share prediction information. This can lead to a decrease in prediction accuracy because these two branches may behave very differently. We quantify the potential mispredictions brought about by this phenomenon by calculating the hash collision rate (HCR), which is the proportion of branches that map to the same prediction entry under a given hash function and prediction mechanism. It can be computed as:

$$\text{Hash Collision Rate} = \frac{\text{Number of Collisions}}{\text{Total Number of Hashes}} \quad (3)$$

A higher ratio signifies a more frequent occurrence of hash collisions, potentially diminishing the predictor's performance due to the possible overwriting of valuable information. Given that different predictors employ unique hash functions, resulting in diverse hash distributions, the incorporation of hash collision analysis not only fine-tunes the trace predictability by considering the impact from the PC, but it can also be integrated with varying predictors in later analytical models to yield more accurate metrics.

B. Classification Module

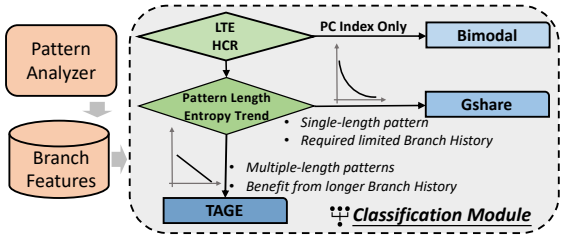

Fig. 3: Classification Module

Now, we apply the previous metrics to discern the suitable branch predictor among Bimodal, Gshare, and TAGE[14] predictors for various applications. These three predictors respectively represent a basic single-table predictor indexed only by PC, a single-table predictor indexed by a mix of PC and branch history, and a multi-table predictor. They offer varying levels of prediction complexity, facilitating a tailored selection for different applications.

Figure 3 demonstrates the workflow of our strategic classification module. We propose several effective criteria to evaluate an application's suitability for a target BP and select the simplest prediction scheme. After choosing a branch predictor, the internal parameter adjustments are handled by subsequent analytical models.

1) Bimodal Predictor: Bimodal uses a table of 3-bit counters indexed by the PC to make predictions with the counter's MSB. This predictor works best for applications that don't use branch history and have regular local branches, which can be reflected by a low local transition entropy (LTE). It also considers PC distribution in the branch trace to minimize hash collisions, which means having a low hash collision rate (HCR). Therefore, the Bimodal Branch Predictor is selected if both the **LTE** and the **HCR** from the pattern analyzer fall below a certain threshold.

2) Gshare Predictor: Gshare is a more advanced prediction technique that uses the hash of global branch history and PC to index the PHT, capturing correlations among different branches effectively and enhancing prediction accuracy.

3) TAGE Predictor: TAGE [14] is currently a state-of-the-art BP for high-performance processors. It extends Gshare by employing multiple tables with varying lengths of global branch history, enabling it to identify and capture intricate correlations between branches. The hierarchical structure of TAGE enables efficient hardware resource utilization and dynamic adaptation to various branch behaviors, capturing longer history correlations.

For a branch trace, our decision to use Gshare or TAGE is based on two factors: (1). Whether the branch trace contains patterns of varying lengths that require multi-table support or not. (2). Whether the branch trace can benefit from an extended history or not. To assess the first criterion, we divide the matrix profile vector by discords into patterns of different lengths and count these lengths. For the second criterion, we calculate the transition entropy values for pattern lengths ranging from 2 to 32, providing an entropy trend line.

As depicted in Fig. 3, if the **GTE(ghr)** drops below a preset threshold and does not continue to decrease with longer patterns, Gshare will be selected. On the other hand, if the entropy continues to decrease and the matrix profile pattern lengths are dispersed, indicating benefits from a multi-table structure and extended history, TAGE will be selected. The preset thresholds mentioned can be tailored by the user. Without customization, the defaults are usually the metrics that correspond to an MPKI of 0.1.

C. Analytical Model Sets

In this section, we introduce analytical models for our targeted Branch Predictors (BPs) to illustrate their miss rate under various hardware parameters. We also estimate storage resource consumption using RAM size as a key metric, a frequent constraint in BP design. Table I lists the adjustable parameters and their potential values for the three targeted BPs. The (min, max, number) shows the range of possible values and the count of candidate values for each parameter. As the parameter configurations increase exponentially with N, all BP types and parameter configurations can form a design space as vast as 0.5×10^{22}.

TABLE I: Design Space of Targeted BPs

BP	Parameters and Description	Candidates
Bimodal	N_b : Number of PHT Entries	(3, 24, 22)
Gshare	ghr : GHR Size	(3, 24, 22)
TAGE	N: Num of tables	(3, 8, 6)
	$\lg b$: Number of entries in basic table T_0	(9, 14, 6)
	$\lg g[i]$: Log number of entries in i_{th} table T_i	(7, 13, 7)
	tag[i] : Tag width of T_i	(7, 14, 8)
	L_n : GHR Size, maximum available history	(6, 128, 2)
	L_1 : Min available history	(4, 8, 5)

We developed performance models inspired by the CPI stack[2] which offers a new perspective on the performance analysis and optimization of Branch Predictors. In our model, Miss Prediction is divided into two main components: one is the basic Miss Prediction based on the regularity of the branch trace, and the other is the additional Miss Prediction caused by hash collisions and compulsory misses. Both of these components can be quantified using the metrics we defined in the pattern analyzer. To further enhance the accuracy of the model, we have introduced some fitting parameters, which are calibrated through a data-driven approach, to bridge the gap between the metrics and the actual Miss rate. This approach enables us to better understand the contribution of each factor to the Miss Prediction Rate and identify directions for optimization.

1) Bimodal Analytical Model: The Bimodal only uses the PC to index the PHT. So we use local transition entropy to quantify the basic miss prediction, and the HCR to quantify hash collisions. Given that the applications opting for the Bimodal typically exhibit simpler branches, our model omits the consideration of compulsory misses to expedite the evaluation process. Specifically, it can be expressed as:

$$mr = \alpha \cdot LTE(N_b) + \beta \cdot HCR \tag{4}$$

$$\text{RAM} = 2^{N_b} \cdot ctr \tag{5}$$

Where mr is the misprediction rate, ctr is the counter width, and α and β are fitting parameters determined by the least squares method.

2) Gshare Analytical Model: The hardware parameter ghr in Gshare reflects its maximum tracked global branch history. Therefore, we use global transition entropy to quantify the basic Miss prediction, and we incorporate the compulsory miss rate obtained from the Matrix Profile as a bias. Therefore, we build the model as follows:

$$mr = \alpha_g \cdot GTE(ghr) + \beta_g \cdot HCR(ghr) + CMR \tag{6}$$

$$\text{RAM} = 2^{ghr} \cdot ctr \tag{7}$$

Where $GTE(ghr)$ represents the global transition entropy at a pattern length of ghr, and CMR refers to the compulsory miss rate, which is obtained by dividing the number of discords given by the matrix profile vector by the total number of branch instructions. α_g and β_g are fitting parameters.

3) TAGE Analytical Model: We regard TAGE as a combination of a Bimodal predictor and N tagged Gshare predictors and leverage the previous two models to build the TAGE model. Furthermore, considering TAGE's selection mechanism, we utilize Table Size, which is calculated as equation(8), to measure the priority of each table and incorporate the tag's hash function into the model. We also ensure compulsory misses are counted separately to prevent double-counting.

$$TableSize_i(TS_i) = 2^{\lg g[i]} \cdot (tag[i] + ctr + u) \tag{8}$$

Based on this, we propose the following equation to reflect the RAM utilization and performance of TAGE:

$$mr_0 = \alpha \cdot LTE(\lg b)$$

$$mr_i = \alpha_g \cdot GTE(L[i], \lg g[i]) + \beta_g(HCR^I + HCR^T)$$

$$mr = \ln[a \cdot mr_0 + b \cdot Avg(mr_i \times TS_i)] + CMR$$

$$\text{RAM} = 2^{\lg b} \cdot ctr + \sum TS_i$$

In this model, mr_0 and mr_i are the refined Bimodal and Gshare models respectively for TAGE. The function $GLE(L[i], \lg g[i])$ computes the global transition entropy for the available history $L[i]$ folded to $\lg g[i]$. HCR^I and HCR^T represent the hash collision rates for index hash and tag hash in TAGE, respectively. a and b are fitting parameters.

III. Experiments and Results

We construct and evaluate our method using the framework provided by the Championship Branch Prediction (CBP-5)[4], which is a BP design competition platform that facilitates the implementation and simulation of various BPs using C++. For a comprehensive validation of our framework, we select a total of 40 applications from three different benchmark sets, which include MiBench[7], SPEC2017[1], and traces in CBP-5[4].

We first demonstrate that transition entropy is a more accurate metric than previous works for characterizing an application's branch behavior. Then, we validate the efficacy of our proposed correction metrics derived from matrix profile and hash collision, as well as our modeling methods,

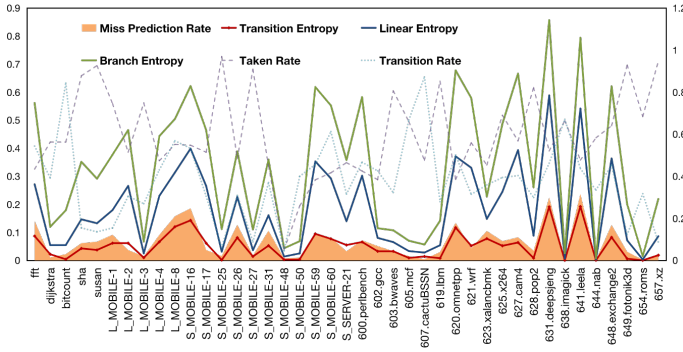

Fig. 4: Values of metrics value and miss prediction for applications under the **same BP configuration**

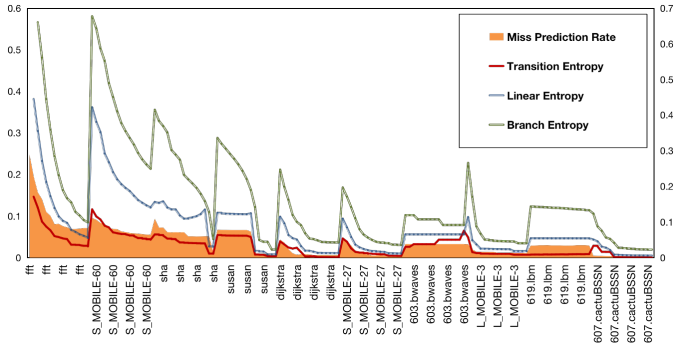

Fig. 5: Values of metrics and miss prediction rate for applications under the **multiple BP configurations**

by demonstrating the performance of our analytical model. Finally, we show the support our framework provides for DSE.

A. Accurate branch behavior characterization

In this part, we simulate all applications under the same BP configuration, which is Gshare with ghr of 9, and collect the misprediction rate to reflect their prediction difficulty. We show that our proposed transition entropy correlates well with the misprediction rate without any fitting or training, accurately reflecting the predictability of different applications.

Figure 4 shows the value of various metrics and miss prediction rates for different applications, including taken rate, transition rate[8], branch entropy[16], linear entropy[13], and our proposed transition entropy. The results indicate that basic metrics such as taken rate and transition rate often fail to adequately represent an application's predictability, while entropy-based metrics are more reliable. It's also clear that our proposed transition entropy outperforms previous works, providing a better estimate of an application's predictability and reflecting differences between various applications without the need for additional scaling or fitting.

To illustrate the generality of our metric, we also expanded our experiments to multiple BP configurations. For each application, we chose 15 different BP configurations, which are Gshare with ghr ranging from 6 to 20, corresponding to different history lengths, and compared three entropy-based metrics. Figure. 5 shows the experimental results. It can be observed that our metrics still outperform other methods, effectively reflecting the impact of BP configuration changes on a specific application.

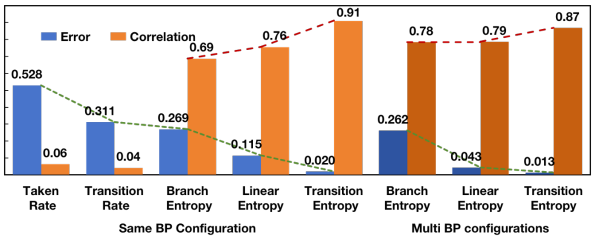

Fig. 6: Comparison of **error** and **correlation** of different metrics

Specifically, we compare the absolute error and correlation between all metrics and the miss rate in Figure. 6, and the correlation is measured by the Pearson correlation coefficient. It shows that under the same BP configuration, the average error between our transition entropy and the miss rate is only **0.02**, and the Pearson coefficient can achieve **0.91**. Compared to the previous best metric[13], the correlation improved by **15%**, and the absolute error reduced by **0.085**. When considering different BP configurations simultaneously, the average error of the proposed transition entropy further drops to **0.013**. Although the correlation coefficient slightly decreases to **0.87**, it still significantly outperforms other metrics. This loss in correlation is due to the different impacts that changes in BP configuration and the differences between applications for a single metric, which also prompts us to employ additional metrics and analytical models for correction.

B. Analytical Model Performance

We demonstrate that our analytical models correlate well with simulated branch miss prediction rates, making them suitable for DSE. Since some parameters need to be fitted in the model, we employ the Sobol method [9] to sample 500 design points for each application and then evaluate the model using leave-one-out cross-validation: we train the model on all benchmarks except one, then evaluate the accuracy for the left-out benchmark, and repeat this process for all benchmarks, with each taking a turn as the left-out benchmark.

Table. II shows the fitting parameters and the correlation results, we can find that the proposed analytical models show a strong correlation with the simulation results, with an average correlation coefficient of **0.88**. Thus it can provide valuable guidance for the selection and parameter adjustment of BPs. Also, by comparing the results of the Gshare model Section III-A, we can prove the effectiveness of our proposed modeling method and correction metrics, which improved the correlation from 0.87 to 0.92. As for the TAGE model, there is no previous work that can reflect the impact of internal parameter changes on the miss replication rate.

Fig. 7 shows the correlation between the predicted misprediction rate from the Gshare and TAGE analytical models, and the misprediction rate obtained from the simulation. Overall, there is a strong correlation between them, particularly in regions of lower misprediction rates, where linearity is even higher. In contrast, distinguishing them becomes challenging

TABLE II: Analytical Model Correlation Results

Model	Fitting Parameters	Value	Pearson Coefficient
Bimodal	α	1.3412	0.8381
	β	0.0398	
Gshare	α_g	0.7684	0.9230
	β_g	0.269	
TAGE	a	-0.1614	0.8861
	b	0.7301	

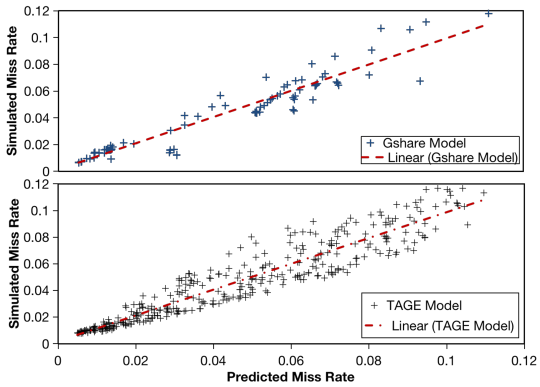

Fig. 7: The correction between predicted miss prediction rate and actual miss prediction rate

in areas of higher MR due to potential imbalances in parameter settings, which can lead to increased branch prediction error rates that the analytical model might struggle to capture. However, in addressing DSE problems, our goal is to achieve optimal performance with the lowest MR. Since regions with better linear correlation are used more frequently, our model successfully provides robust support for BP parameter DSE.

C. DSE Support

We also validated the support of our Analytical Model for DSE. For a specific benchmark, our proposed framework can quickly traverse the design space in parallel, allowing us to find the optimal branch predictor choices and configurations. Under various RAM size constraints from 1KByte to 24KByte, we explored all design points under the constraints, provided the BP configurations with the lowest predicted miss rate, and compared them with sampled simulation data.

Figure 8 presents the results for some benchmarks. We observe significant variations in the design spaces constituted by different applications. Despite these variations, our model demonstrates robust performance within these design spaces, providing configurations that are very close to the Pareto frontier of the sample space.

For time efficiency, once the parameters are set, the analytical model is essentially a simple polynomial, and its computation time is negligible. The main time consumption comes from obtaining metrics from the pattern analyzer, but this process only needs to be done once. We scan all the metrics involved in the design space and obtain them synchronously through a single analysis. When calculating the analytical model, a table lookup is all that's needed. For an application with about a hundred million branches, this analysis process takes about 9 minutes. However, simulating one single point using the CBP-5 takes 15 seconds. This clearly shows that our model has a significant time efficiency advantage compared to traditional simulation methods in previous DSE work.

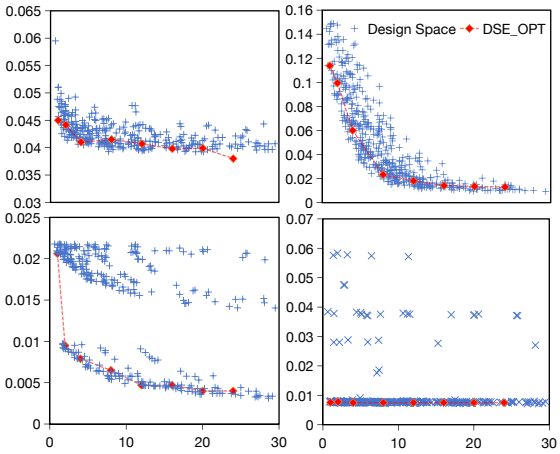

Fig. 8: Design space and the obtained optimal configurations

IV. Conclusion

We propose a modular analytical model approach to enable DSE in branch prediction, which comprises a pattern analyzer, a classification module, and an analytical model set, which together provide fast and accurate end-to-end performance feedback for BP selection and parameter configuration.

References

[1] Spec cpu 2017. https://www.spec.org/cpu2017. 2017. III

[2] Allam et al. An efficient cpi stack counter architecture for superscalar processors. In *GLSVLSI*, 2012. II-C

[3] Bai et al. Boom-explorer: Risc-v boom microarchitecture design space exploration framework. In *ICCAD*, 2021. I

[4] CBP-5. Championship branch prediction. In *2016 International Symposium on Computer Architecture (ISCA)*, 2016. III

[5] Eyerman et al. Characterizing the branch misprediction penalty. In *ISPASS*. IEEE, 2006. I

[6] Ferres et al. A chisel framework for flexible design space exploration through a functional approach. *TODAES*, 2023. I

[7] Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE WWC-4*, 2001. III

[8] Haungs et al. Branch transition rate: A new metric for improved branch classification analysis. In *HPCA-6*. IEEE, 2000. III-A

[9] Joe et al. Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM TOMS*, 2003. III-B

[10] Jongerius et al. Analytic multi-core processor model for fast design-space exploration. *IEEE Transactions*, 2017. I

[11] B. C. Lee et al. Illustrative design space studies with microarchitectural regression models. In *IEEE HPCA*, 2007. I

[12] C.-K. Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. *Acm sigplan notices*, 2005. II

[13] D. Pestel et al. Micro-architecture independent branch behavior characterization. In *ISPASS*. IEEE, 2015. I, II-A1, III-A, III-A

[14] Seznec et al. Tage-sc-l branch predictors again. In *JWAC-5: Championship Branch Prediction (CBP-5)*, 2016. II-B, II-B3

[15] Van et al. Analytical processor performance and power modeling using micro-architecture independent characteristics. *IEEE Transactions*, 2016. I

[16] Yokota et al. Potentials of branch predictors: From entropy viewpoints. In *ARCS*, 2008. I, II-A1, III-A

[17] C. Zhou et al. Design space exploration of tage branch predictor with ultra-small ram. In *GLSVLSI*, 2017. I