# LVCP: A Load Value Correlated Predictor for TAGE-SC-L

Yang Man
Institute of Computing Technology, Chinese Academy of Sciences
manyang24s@ict.ac.cn

Lingrui Gou
Institute of Computing Technology, Chinese Academy of Sciences
goulingrui19s@ict.ac.cn

Yuhang Liu*
Institute of Computing Technology, Chinese Academy of Sciences
liuyuhang@ict.ac.cn

Mingyu Chen
Institute of Computing Technology, Chinese Academy of Sciences
cmy@ict.ac.cn

Yungang Bao
Institute of Computing Technology, Chinese Academy of Sciences
baoyg@ict.ac.cn

## Abstract

Conditional branch prediction remains a crucial technology for modern high-performance processors. A substantial portion of mispredictions arise from data-dependent branches. Although various mechanisms targeting data-dependent branches have been proposed, none were employed in previous championships. To address this, we implemented a Load Value Correlated Predictor (LVCP) that leverages prior load data values to improve the predictability of hard-to-predict branches. Additionally, we incorporated several techniques introduced since the last championship. The resulting LVC-TAGE-SC-L achieved 3.372 branch mispredictions per kilo instructions (BrMisPKI) and 144.076 cycles on wrong path per kilo instructions (CycWpPKI) across 105 training traces, operating within a 192 KB storage budget. This performance represents a 2.07% reduction in branch mispredictions compared to the baseline predictor.

## 1 Introduction

As a cornerstone of modern high-performance processors, the branch predictor is becoming increasingly important due to the rising penalty of mispredictions in deeper and wider pipelines. Enhancing branch prediction accuracy offers a relatively straightforward approach to improving performance while simultaneously reducing energy consumption.

Prior research into branch prediction primarily focused on utilizing past branch history. Since 2006, research in branch prediction has progressed very slowly. The TAGE predictor [15] has already excelled in storage efficiency, and all the following branch prediction championships were won by TAGE-based predictors [11–13].

However, branch history is not the only source of information available to the processor for branch prediction. Various types of execution context can also be used, including load addresses, load values, and register contents. For certain hard-to-predict branches, relying solely on historical patterns is ineffective, since branch history alone struggles to distinguish between disparate execution contexts that share the same recent branch patterns.

*Corresponding author


*'CBP 2025', June 21, 2025, Tokyo, Japan*


The load values, as a specific aspect of the execution context, can directly influence or even determine the outcome of subsequent branches in certain scenarios. A prominent example is the exit condition of loops with dynamically determined iteration counts. While challenging for history-based predictors like TAGE due to their noisy patterns, these branches are often accurately predictable by observing the value loaded into the loop's iteration counter.

Building on this observation, this paper proposes LVC-TAGE-SC-L, which augments TAGE-SC-L with a Load Value Correlated Predictor. This proposed predictor is more capable of predicting load-dependent hard-to-predict (H2P) branches than the baseline TAGE-SC-L predictor. Section 2 provides a historical context for the TAGE predictor and previous research on load-dependent predictions. Section 3 will describe the proposed predictor in detail, including:

- a load tracking mechanism that allows each branch to obtain previous load information.
- a correlation table and its training algorithm to capture the correlation between the load-dependent branch and its predecessor load.
- several new features used in the Statistical Corrector.

Finally, Section 4 provides an experimental evaluation of our proposed predictor, and Section 5 briefly discusses other findings during the championship.

## 2 Related Work

### 2.1 TAGE Variant

TAGE has long been the most accurate branch predictor since it was proposed in CBP-2 [15]. Then various improvements were made by André Seznec during the next several CBPs, including adding a loop predictor to handle loop exit branches [10], adding an O-GEHL-based statistical corrector to deal with branches that are not well correlated with specific history patterns but present statistical bias [11]. Other techniques such as table sharing and partial associativity are also applied [12], forming the winner TAGE-SC-L in CBP-5 [13]. After that, Pierre Michaud proposed BATAGE to mitigate the "cold counter problem" [7]. Seznec proposed a hardware-friendly TAGE-SC in 2024 [14].

### 2.2 Load-Correlated Prediction

Load-correlated branch prediction techniques leverage information from load instructions to enhance branch outcome prediction [1, 3, 4]. Early approaches typically relied on the value of a

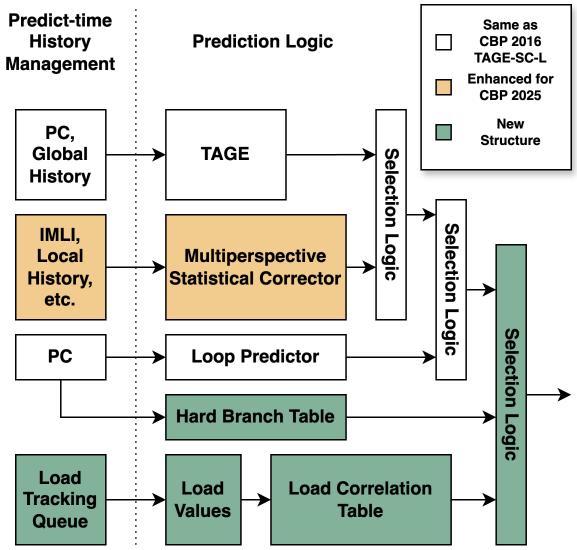


**Figure 1: Load Value Correlated Predictor with TAGE-SC-L**

load instruction located at a fixed distance before the branch [1, 4]. Alternatively, using the load address—available earlier than the corresponding data value has been proposed to reduce prediction latency [3]. A key challenge for value-based prediction arises when a subsequent store modifies the expected load value, resulting in mispredictions. To address this issue, active update mechanism that leverage store values to update predictor state has been proposed [1]. In addition, complementary techniques such as pre-executing hard-to-predict branches has also been explored to improve prediction accuracy [2, 8, 18].

## 3 Structure Design

Our proposed LVC-TAGE-SC-L predictor consists of four parts: TAGE, Multiperspertive Statistical Corrector (MPSC), Loop Predictor and Load Value Correlated Predictor (LVCP). A block diagram of the relationship between the components is shown in Figure 1.

The TAGE component and the Loop Predictor are based on the CBP-5 implementation by Seznec [13], with only minor parameter tuning to satisfy storage constraints. The MPSC component incorporates features such as the per-set history and the regional IMLI, inspired by techniques developed since the last championship [6, 14]. The Load Value Correlated Predictor (LVCP) is designed to capture load-branch correlations through a dedicated load tracking structure and a correlation table. A separate H2P branch table is employed to determine whether a branch is sufficiently hard to predict to warrant the use of LVCP. A detailed discussion of its operation is provided in Section 3.3.

At prediction time, all predictor components are accessed in parallel, and the final prediction is selected based on a hierarchical decision process. If the branch is identified as a hard-to-predict (H2P) branch and the Load Correlation Table yields a hit with a saturated confidence counter, the prediction from the Load Value Correlated Predictor (LVCP) is used. If this condition is not met but the Loop Predictor provides a prediction with saturated confidence, its result is adopted. In all other cases, the prediction from TAGE-SC is used, following the same logic as in the CBP-5 implementation [13].

After the prediction is completed, the load tracking queue is updated with potential load instructions. The detailed update logic is described in Section 3.3. Some of the SC histories are speculatively updated (mimic a speculative-update-mispredict-rollback mechanism in the championship). See Section 3.4 for detailed discussion on the update mechanism of SC components.

TAGE-SC-L is updated at execution resolve time to allow for a more timely update. LVCP is updated at commit time since LVCP needs an accurate order between branches and loads.

### 3.1 H2P Branch Table

A H2P Branch Table (HBT) is used to determine whether a branch is hard enough for any allocation in the LVCP. HBT is a small register-file-based set-associative table that is PC-indexed and tagged. There is a saturation counter in each entry that increases when a branch is mispredicted by TAGE-SC-L. When the counter saturates, the branch is considered hard to predict.

When a branch mispredicts, HBT tries to allocate a slot for it if any counter in the same set is zero. The counter decays with a certain period. In our case, the decay period is set to 20000 branches committed. Since the average branch per kilo instruction (BPKI) is 131.60 across the traces, the HBT tracks the branches whose MPKI is greater than 0.38. This should be enough to cover most of the hard-to-predict branches.

### 3.2 Load Tracking Structure

The load tracking structure is responsible for tracking the positional relationship between branches and preceding loads. Fig. 2 shows the specific load tracking structure used in the Load Value Correlated Predictor.

Before predicting a branch, information about loads preceding it enters a load tracking queue. Once a tracked load instruction completes execution, its value is updated in the corresponding entry within the load tracking queue. The load values stored in the queue are later used to index a banked correlation table, leveraging a one-to-one mapping between each queue position and a specific table bank. The size of the load tracking queue is limited (16 in our case) primarily to keep the number of SRAM banks required for the correlation tables reasonable.

However, a significant amount of load-branch dependencies can occur over distances exceeding the capacity of this limited queue. To capture some of these long-distance dependencies, a distant load buffer, indexed by the load PC low bits, is employed. When a load instruction is dequeued from the load tracking queue, it is transferred to this distant load buffer. Values within the distant load buffer are also used later to query the correlation tables, similar to the values in the load tracking queue. The distant load buffer entries are directly mapped to the correlation table banks.

To allow the detection of load instructions at prediction time, a small set-associative SRAM table, termed the Load Marking Table, is employed. Each entry within this table contains a bitmap indicating the presence and positions of load instructions within a

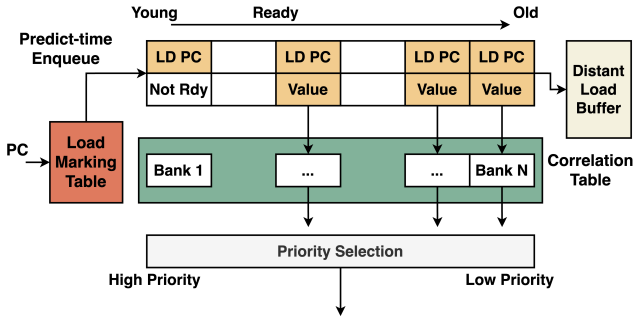


**Figure 2: Load tracking structure including a load marking table, a load tracking queue, and a distant load buffer**

corresponding cache line. Entries also include a useful counter used for replacement policy. This useful counter is incremented whenever a load instruction whose address maps to that entry contributes to a correct load-correlated prediction, particularly those that correct errors from the baseline predictor (TAGE-SC-L). To prevent stale information from persisting indefinitely, these counters are gradually decayed over time. The bitmaps themselves are populated during the decode stage upon the detection of load instructions.

## 3.3 Load Value Correlated Predictor

The core of the Load Value Correlated Predictor is a set-associative SRAM-based correlation table. For lookup, it uses a hash computed from the branch PC, load PC, and load value, which serves as both the index into the set and the tag for matching entries. Each entry in the correlation table stores a payload consisting of: a direction bit, a 5-bit confidence counter, and a 1-bit direction-changed marker used for invalidation.

When predicting a branch, the predictor retrieves load PC and load value information from both the load tracking queue and the distant load buffer. This branch's PC, the retrieved load PC, and the load value are then hashed to calculate the set index and the tag used to access the correlation table. If multiple load candidates are available from the load tracking queue, the predictor selects the prediction from the youngest load with saturated confidence. For candidates found in the distant load buffer, the predictor selects the entry that has the smallest index and saturated confidence. When both information from the load tracking queue and the distant load buffer can provide prediction, the prediction from load tracking queue is used for more timely information.

In the training procedure, new correlation table entries are allocated under the following conditions: 1) the branch is classified as hard-to-predict (H2P), and 2) the overall predictor (e.g., TAGE-SC-L plus LVCP components) mispredicts. The allocation policy searches for candidates containing entries marked as not useful (i.e., their useful counter is zero). Multiple entries may be allocated to potentially speed up the training procedure. To retain the most critical correlation entries, a 2-bit useful counter is associated with each entry. This counter is incremented when the baseline predictor (TAGE-SC-L) mispredicts but the load-correlated entry provides the correct prediction. When allocation fails because no table contains unused entries (i.e., all applicable entries are marked as useful), a probabilistic decay mechanism is employed where useful counters in all the tables are decremented with a small probability (controlled by an LFSR random number generator). A single load-dependent branch instance can often be correlated with multiple preceding loads. To optimize storage and prioritize the most effective correlations, if one entry providing a correct prediction for a branch instance reaches saturated confidence, the useful counters of other entries potentially correlating with the same instance are reset to zero. This mechanism encourages keeping only one provider for a given context, freeing space for other branches or contexts.

Whenever the correct branch direction differs from the direction stored in a correlation table entry, the entry's direction-changed bit is set. Once this bit is marked, the entry becomes invalid and will no longer participate in prediction or training processes. This mechanism helps prevent the predictor from using stale or incorrect historical correlations, thus reducing mispredictions.

To conserve storage, the load information from the tracking queue and the distant load buffer share the same correlation table storage. This requires dual-port SRAM or fine-grained SRAM banking in real hardware implementation.

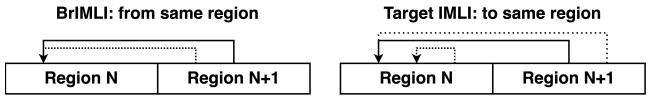


**Figure 3: Branch IMLI and Target IMLI**

## 3.4 Multiperspective Statistical Corrector

The Multiperspective Statistical Corrector (MPSC) is derived from the GEHL predictor by Seznec [9] and Multiperspective Perceptron Predictor by Jiménez [5]. It follows Seznec CBP-5 submission for dynamic threshold and weight.

**Bias** TAGE are known to predict some biased branches poorly [11]. Three bias tables indexed by different strategies are used, which is the same as the CBP-5 winner.

**Global History** Three global history tables are also used as in [11–13]. The global history tables are shared with the IMLI tables. When each of the three IMLI counters is non-zero, IMLI counter will be used instead to access the table.

**Local History** 32 per-address and 16 per-set [19] history are used. These local histories are speculatively updated, as they are small.

**Path History** IMLI-filtered backward history [14] and normal global forward history are used. These histories are also speculatively updated.

**IMLI** IMLI is short for Inner-Most Loop Iteration counter [16]. IMLI and IMLI-OH feature are used in our predictor. IMLI is reengineered to branch IMLI (brIMLI) and target IMLI (tarIMLI), which helps with multiexit loop [14]. Figure 3 shows the definition of the two new IMLI. A forward target IMLI is also used to capture some reverse loop [5]. IMLI counter is speculatively updated and checkpointed to ensure accurate tracking. However, IMLI-OH table is updated at execution resolve to avoid checkpointing. IMLI-OH update does not need to be very timely according to [16].

## 4 Experimental Results and Analysis

We evaluate our proposed LVC-TAGE-SC-L on the CBP simulation framework across 105 provided traces. We compare our proposed predictor with two different configurations of the original TAGE-SC-L. There are two key differences between these two configurations, which are **1) the number of local history entries**: "unreal" has 2048 entries while "realistic" has only 48 entries; **2) when to update the IMLI-OH components of the statistical corrector**: "unreal" updates at prediction time while "realistic" updates at execution. The "unreal" configuration generally corresponds well to the CBP-5 version. Both configurations are tuned to the 192KB budget. For rationality of selecting such a baseline, see discussions in Section 5.

Table 1 shows the branch mispredictions per kilo instructions (BrMisPKI) and cycles on wrong path per kilo instructions (CycWpPKI) compared to the baseline TAGE-SC-L. The proposed predictor assigned 173.19KB storage to TAGE-SC-L and 18.7KB to LVCP. For detailed storage cost analysis, see Appendix A. Evaluation results show that the proposed optimized TAGE-SC-L configuration retains most of the performance benefits of an idealized local history setup while requiring significantly less storage. Moreover, by integrating the Load Value Correlated Predictor, the overall predictor achieves a 2.07% reduction in BrMisPKI compared to the baseline predictor at equivalent storage capacity.

**Table 1: Average BrMisPKI and CycWpPKI on all traces**

| Predictor | BrMisPKI | CycWpPKI | Rdc. |
|---|---|---|---|
| 192KB realistic TAGE-SC-L | 3.444 | 145.647 | - |
| 192KB unreal TAGE-SC-L | 3.421 | 145.181 | 0.66% |
| 173KB our TAGE-SC-L | 3.428 | 145.436 | 0.45% |
| 192KB TAGE-SC-L w/ LVCP | **3.372** | **144.076** | **2.07%** |

Detailed BrMisPKI improvements over the baseline are shown in Figure 4. The proposed predictor yields significant BrMisPKI reductions on infrastructure (4.57%), floating point (3.42%), and integer (1.89%) workloads. Performance on media workloads, however, presents a different picture. While a predictor using an unrealistic amount of local history significantly reduces BrMisPKI on it, neither our proposed TAGE-SC-L nor the baseline TAGE-SC-L achieves a comparable reduction in this category.

Figure 5 shows a feature contribution analysis result of each feature used in LVCP and SC (except for Bias in SC). The BrMisPKI reduction is obtained by disabling each feature in both predicting and training (no extra storage added to other components). The result shows that LVCP contributes to 0.05 BrMisPKI reduction, which is the most effective feature. Total BrMisPKI reduction of all the features in Figure 5 is 0.156 (4.6%).

## 5 Discussions

### 5.1 Local History Usage

Local history is known to significantly improve branch prediction accuracy across various predictor designs, including GShare predictors, TAGE, and perceptron predictors. For example, the CBP-5 winning TAGE-SC-L predictor utilizes a large 1024-entry local history table [13]. This allows for significant misprediction reduction

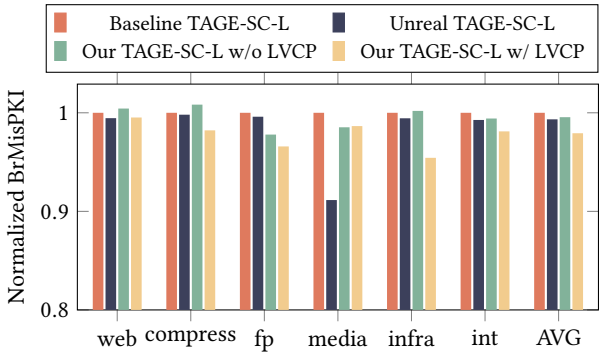


**Figure 4: Comparison of BrMisPKI Across Different Workload Categories**

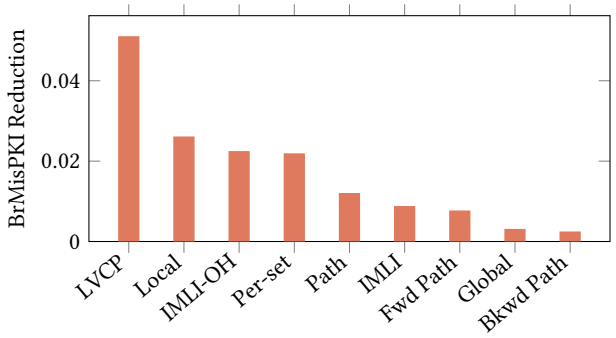


**Figure 5: Feature Contribution to BrMisPKI (Bkwd Path refers to IMLI filtered backward path history)**

on certain workloads (e.g., media workloads, as shown in Figure 4). However, accurately maintaining such an amount of local history is impractical in modern deep and wide processors. Although techniques like partial restoration have been proposed to address this challenge [17], our design opts for a smaller number of local history entries. We find that by combining this limited local history with per-set history and IMLI, the majority of the prediction accuracy benefits that a large local history table brings can still be attained. Furthermore, checkpointing these histories for speculation management remains well within acceptable power and area limits (under 1Kbits per fetch block, or roughly 10KB total for the whole processor).

### 5.2 Timeliness

While exploring the predictability potential, we found that more than 20% of the mispredicted branches actually correlate well with specific fixed load values prior to the branch. However, these load values come too late to be used at the time when the correlated branches are actually predicted. In the CBP framework, the prediction of a certain branch can happen only once. But in real processors, the prediction made by the main predictor can possibly be corrected before the branch reaches the execution stage. During that period, if the corresponding load values are ready, the pipeline can re-steer earlier, thus reducing cycles the processor spends on wrong path.

## 6 Conclusions

Unlike prior research that primarily focuses on branch history, this work investigates a branch prediction mechanism based on load tracking and load-value correlation, leveraging the execution context of load instructions. Although this is an initial exploration, we hope it will motivate further research into load-based correlations and the broader utilization of diverse processor contexts to enhance prediction accuracy.

Although our proposed techniques successfully achieve a Br-MisPKI reduction over the baseline predictor, we identified two primary areas for future improvement. Firstly, the current load tracking mechanism is relatively complex. Secondly, the storage requirements for the load correlation table are substantial. Consequently, promising avenues for future work include developing smarter mechanisms for utilizing load information and improving the overall storage efficiency of the predictor.

## Acknowledgments

This work is partially supported by the National Key Research and Development Program of China under Grant No. 2023YFB4503904, and the Innovation Funding of ICT, CAS under Grant No. E361100. The authors also thank the Beijing Institute of Open Source Chip (BOSC) for providing resources and facilities that supported this research.

## References

[1] Muawya Al-Otoom, Elliott Forbes, and Eric Rotenberg. 2010. EXACT: explicit dynamic-branch prediction with active updates. In *Proceedings of the 7th ACM international conference on Computing frontiers* (Bertinoro Italy, 2010-05-17). ACM, 165–176. https://doi.org/10.1145/1787275.1787321

[2] Aniket Deshmukh, LingzheChester Cai, and Yale N. Patt. 2024. Timely, Efficient, and Accurate Branch Precomputation. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Austin, TX, USA, 2024-11-02). IEEE, 480–492. https://doi.org/10.1109/MICRO61859.2024.00043

[3] Hongliang Gao, Yi Ma, Martin Dimitrov, and Huiyang Zhou. 2008. Address-branch correlation: A novel locality for long-latency hard-to-predict branches. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture* (2008-02). 74–85. https://doi.org/10.1109/HPCA.2008.4658629 ISSN: 2378-203X.

[4] T.H. Heil, Z. Smith, and J.E. Smith. 1999. Improving branch predictors by correlating on data values. In *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture* (1999-11). 28–37. https://doi.org/10.1109/MICRO.1999.809440 ISSN: 1072-4451.

[5] Daniel A Jiménez. 2016. Multiperspective perceptron predictor. In *5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*.

[6] Pierre Michaud. 2014. Five poTAGEs and a COLT for an unrealistic predictor. In *4th JILP Workshop on Computer Architecture Competitions (JWAC-4): Championship Branch Prediction (CBP-4)*.

[7] Pierre Michaud. 2018. An alternative tage-like conditional branch predictor. *ACM Transactions on Architecture and Code Optimization (TACO)* 15, 3 (2018), 1–23.

[8] Stephen Pruett and Yale Patt. 2021. Branch Runahead: An Alternative to Branch Prediction for Impossible to Predict Branches. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2021-10-17) *(MICRO '21)*. Association for Computing Machinery, 804–815. https://doi.org/10.1145/3466752.3480053

[9] André Seznec. 2005. Analysis of the o-geometric history length branch predictor. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 394–405.

[10] André Seznec. 2007. A 256 kbits l-tage branch predictor. *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)* 9 (2007), 1–6.

[11] André Seznec. 2011. A 64 kbytes ISL-TAGE branch predictor. In *JWAC-2: Championship Branch Prediction*.

[12] André Seznec. 2014. Tage-sc-l branch predictors. In *JILP-Championship Branch Prediction*.

[13] André Seznec. 2016. Tage-sc-l branch predictors again. In *5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5)*.

[14] André Seznec. 2024. TAGE: an engineering cookbook. https://files.inria.fr/pacap/seznec/TageCookBook/RR-9561.pdf

[15] André Seznec and Pierre Michaud. 2006. A case for (partially) tagged geometric history length branch prediction. *The Journal of Instruction-Level Parallelism* 8 (2006), 23.

[16] André Seznec, Joshua San Miguel, and Jorge Albericio. 2015. The inner most loop iteration counter: a new dimension in branch history. In *Proceedings of the 48th International Symposium on Microarchitecture* (Waikiki Hawaii, 2015-12-05). ACM, 347–357. https://doi.org/10.1145/2830772.2830831

[17] Niranjan Soundararajan, Saurabh Gupta, Ragavendra Natarajan, Jared Stark, Rahul Pal, Franck Sala, Lihu Rappoport, Adi Yoaz, and Sreenivas Subramoney. 2019. Towards the adoption of Local Branch Predictors in Modern Out-of-Order Superscalar Processors. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2019-10-12) *(MICRO '52)*. Association for Computing Machinery, 519–530. https://doi.org/10.1145/3352460.3358315

[18] Akash Sridhar, Nursultan Kabylkas, and Jose Renau. 2020. Load Driven Branch Predictor (LDBP). arXiv:2009.09064 [cs] http://arxiv.org/abs/2009.09064

[19] Tse-Yu Yeh and Yale N Patt. 1993. A comparison of dynamic branch predictors that use two levels of branch history. In *Proceedings of the 20th annual international symposium on computer architecture*. 257–266.

## A Cost Analysis

| **pred_time_histories** | |
|---|---|
| GHIST | 64 bits |
| phist | 64 bits |
| ptghist | 12 bits |
| tage_index ch_i | 11 × 30 bits |
| tage_tag ch_t | (10+11) × 9 + (13+14) × 21 bits |
| per_set_local_history | 16 × 14 bits |
| local_history | 32 × 11 bits |
| ltable | 32 × 14 bits |
| WITHLOOP | 7 bits |
| last_back_br | 64 bits |
| last_forward_br_target | 64 bits |
| last_back_target | 64 bits |
| brIMLI | 10 bits |
| brIMOH | 10 bits |
| tarIMLI | 10 bits |
| tarIMOH | 10 bits |
| fwdtarIMLI | 10 bits |
| forward_path_history | 11 bits |
| filtered_backward_path_history | 11 bits |
| **TOTAL** | **2521 bits** |

| Component | Details of field | Cost |
|---|---|---|
| **TAGE** | | |
| gtable(low) | ctr: 3 bits<br>tag: 11 bits<br>useful 1 bit<br>9 tables × 2048 entries | 33.75 KB |
| gtable(high) | ctr: 3 bits<br>tag: 14 bits<br>useful 1 bit<br>21 tables × 2048 entries | 94.5 KB |
| use_alt_on_na | 5bits, 16 entries | 80 bits |
| active_hist.ghist | 3000 bits | 0.366 KB |
| active_hist.phist | 27 bits | 27 bits |
| TICK | 10 bits | 10 bits |
| btable | pred: 1 bit × 128K<br>hyst: 1bit × 32K | 20 KB |
| **Loop Predictor** | | |
| ltable | NbIter: 14 bits<br>confid: 4 bits<br>CurrentIter 14 bits<br>TAG: 20 bits<br>age: 4 bits<br>dir: 1 bit<br>64 entries | 0.445 KB |
| **Statistical Corrector** | | |
| updatethreshold | 12 bits | 12 bits |
| Pupdatethreshold | 8 bits, 128 entries | 0.125 KB |
| Bias (Bias, BiasSK, BiasBank) | 7 bits, 3 × 1024 entries | 2.625 KB |
| Global history (GGEHLA, PGEHLA) 3 history lengths | 7 bits, 2 × 3 × 1024 entries | 5.25 KB |
| Local history (PSGEHLA, LGEHLA) 2 history lengths | 7 bits, 2 × 2 × 1024 entries | 7 KB |
| Local history registers | 11 bits × 32 entries | 352 bits |
| Per-set history registers | 14 bits × 16 entries | 224 bits |
| IMLI-OH (SCTARIMOHA, SCBRIMOHA) 2 history lengths | 7 bits, 2 × 2 × 1024 entries | 3.5 KB |
| IMLI-OH (SCTARIMOH_Hist, SCBRIMOH_Hist) | 10 bits, 2 × 1024 entries | 2.5 KB |
| SCFWDA | 7 bits, 2 × 1024 entries | 1.75 KB |
| SCBKDA | 7 bits, 3 × 512 entries | 1.31 KB |
| Weights (WG, WL, WP, WbrIMLI, WtarIMLI, WfwdbrIMLI, WIMOH, WILIPath, WB) | 6 bits, 8 entries | 432 bits |
| FirstH | 7 bits | 7 bits |
| SecondH | 7 bits | 7 bits |

| Component | Details of field | Cost |
|---|---|---|
| **HardBranchTable** | | |
| _table | tag: 23 bits<br>ctr: 5 bits<br>128 entries | 1.078 KB |
| replacer | $8 \times 16 \times (16-1)/2 = 960$ bits | 0.117 KB |
| retired_branches | 15 bits | 15 bits |
| **CorrelationTable** | | |
| _tables | valid: 1 bit<br>tag: 16 bits<br>dir: 1 bit<br>dir_changed: 1 bit<br>dir_conf: 5 bits<br>useful: 2 bits<br>16 tables × 256 entries | 13 KB |
| lfsr | 32 bits | 32 bits |
| _replacer | 2048 bits | 0.25 KB |
| **LoadValueCorrelatedPredictor** | | |
| pred_prev_<br>taken_br_info<br>pred_prev_br_info | id: 10 bits<br>br_pc: 64 bits<br>target: 64 bits | 276 bits |
| pred_load_queue | id: 10 bits<br>pc: 32 bits<br>prev_taken_br_id: 10 bits<br>load_value: 32 bits<br>16 entries | 0.164 KB |
| distant_load_buffer | id: 10 bits<br>pc: 32 bits<br>load_value: 32 bits<br>16 entries | 0.145 KB |
| commit_load_queue,<br>commit_distant<br>_load_buffer | id: 10 bits<br>pc: 32 bits<br>16 entries | 0.164 KB |
| load_marking_table | tag: 16 bits,<br>bitmap: 16 bits<br>useful: 3 bits<br>1024 entries | 4.375 KB |
| load_inst_num | 14 bits | 14 bits |
| decay_lmt_idx | 7 bits | 7 bits |
| lfsr | 32 bits | 32 bits |
| **TOTAL** | | **191.96 KB** |