Shared Pattern History Tables in Multicomponent Branch Predictors With a Dealiasing Cache

Moumita Das, Ansuman Banerjee, Mainak Chaudhuri, and Bhaskar Sardar

***Abstract*—In this letter, we study the problem of designing multicomponent branch predictors for low-resource embedded systems. The highlight of our design is a shared pattern history table with a tiny dealiasing cache for intercomponent interference resolution. Experiments on the CBP-5 traces are shown to record the performance in terms of prediction accuracy. Synthesis results generated using the Synopsys design compiler with TSMC 90-nm libraries confirm the area and power benefits of our design.**

***Index Terms*—Branch prediction, history table, interference.**

I. Introduction

CLASSICAL multicomponent hybrid branch predictors have found wide-spread use in the front-end of the commercial processor pipelines; representative examples include the Alpha 21264 Tournament predictor [11], the skew branch predictors such as the 2Bc-gskew branch predictor which was planned for the Alpha EV8 front-end [15]. The main motivation driving multicomponent predictors has been the observation [10] that different dynamic predictors fare differently on different branches in terms of prediction accuracy, thereby necessitating the use of multiple predictor components for predicting a branch. Multicomponent predictors have been extensively studied in the literature, with a number of design strategies, attempting to improve prediction accuracy and power [2], [5].

A typical and widely popular class of multicomponent predictors consists of a local and a global predictor component and uses a sophisticated tournament prediction scheme to choose the final prediction between these predictors at run time. A *local* history-based predictor uses past outcome information only about the branch under consideration for its current prediction, while a *global* one takes into account the outcome histories of the preceding branches in addition to the present while making a prediction for a specific branch. The local and global components maintain separate pattern history tables (PHTs) to store prediction information as 2-bit saturating counters [13], which is used for predicting directions of future instances of branch instructions. This ensures the predictors can operate in their individual spaces, without interference.

Implementing multicomponent predictors on low-resource embedded processors has been a challenge due to the high area footprint that the separate tables demand. On the other hand, sophisticated contemporary predictors like the TAgged GEometric length predictor [14] and its variants also demand separate history structures for their components to perform well, and have been shown to be storage-sensitive in terms of performance [6].

The main highlight of this letter is the design of a shared single-PHT multicomponent hybrid predictor that can leverage the benefits of the local and global histories, while at the same time, require a low area footprint. In addition to the shared PHT, we keep a small dealiasing cache to hold history information for the address entries on which the local and global components collide, while the operation of the PHT remains shared between predictors otherwise.

This letter on economizing the pattern history table budget is applicable to the generic class of multicomponent hybrid branch predictors that traditionally employ separate PHTs for individual predictor components. While our proposal can be successfully applied to a wide variety of hybrid predictors, the other predictors that do not employ multiple PHTs are beyond the scope of this letter. There exist proposals that replace the PHT of two-level branch predictors by a reasonably large cache to address under-utilization of the PHT entries [8], [9]. In contrast, this letter augments a traditional PHT with a tiny specialized de-aliasing cache with the sole goal of reducing destructive interference in the PHT entries. This distinguishes us from earlier proposals.

Experiments on the CBP-5 [4] traces show that our proposal achieves comparable performance in terms of prediction accuracy in comparison to the classical non shared implementation. Further, we present synthesis results using the Synopsys design compiler (DC) compiler with TSMC 90-nm libraries to show the energy and area benefits of our design.

II. Proposed Multicomponent Predictor Design

Fig. 1 shows an overview of our architecture, with 2 predictor components, a local and a global operating independently. The local one uses a branch history table (BHT) to store the branch-specific history patterns (branch outcomes). The local branch history register (LBHR) corresponding to a branch

Manuscript received September 9, 2019; accepted November 10, 2019. Date of publication December 3, 2019; date of current version August 27, 2020. This manuscript was recommended for publication by Y. Xie. *(Corresponding author: Ansuman Banerjee.)*

M. Das is with the Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata 700108, India, and also with the Department of Information Technology, Jadavpur University, Kolkata 700098, India.

A. Banerjee is with the Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata 700108, India (e-mail: ansuman@isical.ac.in).

M. Chaudhuri is with the Department of Computer Science and Engineering, IIT Kanpur, Kanpur 208016, India.

B. Sardar is with the Department of Information Technology, Jadavpur University, Kolkata 700098, India.

Digital Object Identifier 10.1109/LES.2019.2957512

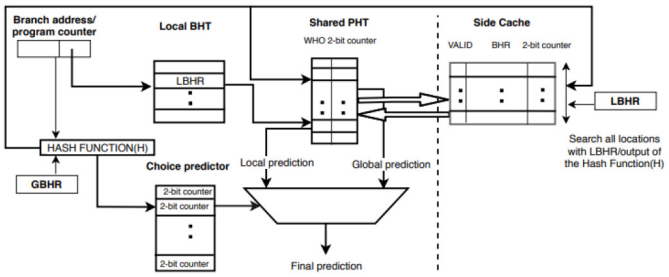

Fig. 1. Our proposal: Shared PHT with side cache.

stores an n-bit pattern corresponding to the last n outcomes of the branch. For a given branch, the BHT is indexed by the last m bits of the branch address or the program counter (PC) value and the n-bit LBHR value stored at that index is used to index the PHT. The BHT stores 2^m entries, each of size n, thus occupying a storage of $n * 2^m$ bits. The PHT has 2^n entries. Each entry in the PHT stores a 2-bit saturating counter, the most significant bit of this counter value is the predicted direction of a branch instance. The global predictor uses the hash function in [14] with a k-bit global branch history register (GBHR) and PC to index the PHT

$$H = \text{PC}[0 : n-1] \oplus \text{GBHR}[0 : n-1] \\ \oplus \text{GBHR}[k-n : k-1].$$

The GBHR stores a running outcome history of the n previous branch instances. The predictions of the local and the global predictors are read from the corresponding indices of the PHT. To select between the local and global predictions for a branch, a choice predictor using the same hash function H is used to index into a predictor selection table. Each entry of this table stores a 2-bit saturating counter whose MSB determines the best predictor (either global or local) for every branch. If the counter state is 00 or 01, the prediction from the global predictor is considered, while the local one is used for 10 or 11. When the branch outcome gets known, this counter is updated only if the predictions from the two predictors are different, based on whose prediction matched the outcome [11]. The counter state is decremented if the branch outcome matches that of the global, otherwise, it is incremented. When a branch retires, the corresponding LBHR, GBHR, and the 2-bit counter in the shared PHT needs to be updated with the actual outcome to train the predictors for future prediction. This may lead to a potential interference if the PHT entry to be updated is owned by the other component. To enable PHT sharing and ensure noninterference, we identify and resolve the PHT interference between the components and allocate side cache entries. Additionally, we modify the predictor lookup of each in the fetch stage for incorporating the cache with the PHT, and augment the retire stage with a cache entry creation step.

Handling Intercomponent Interferences: An interference occurs when an entry in the PHT which one of the components (global or local) uses to keep its 2-bit counter history information is modified by the other component. To identify interference, each entry in the shared PHT has an *ownership* bit (WHO bit in Fig. 1), which holds the current owner of the entry, a value 0 indicates that the global component owns it, while 1 indicates ownership of the local one. A predictor can modify the prediction information of a PHT entry only if it owns the entry or nobody owns it. WHO bits for all PHT entries are initially set to 0, assigning ownership to the global. However, this ownership can change. We adopt a *first touch* allocation policy [12] to ensure that whoever accesses the entry first with a valid address becomes the owner and that disallows the other component from accessing the entry. If an owned entry is accessed by a nonowning component, a side cache entry is created for the latter.

Algorithm 1: Global Prediction at Fetch Cycle

```
1  T : Shared PHT; S : Side cache; GBHR : Global BHR
2  gindex = index of T using indexing function H
3  Access T[gindex] and find the WHO bit stored in that entry.
4  if WHO = 0 then
5  |   gpred = MSB of the T[gindex].2BitCounter
6  else
7  |   sindex = Side cache entry (S) for GBHR
8  |   if sindex is not NULL and has VALID = 1 then
9  |   |   gpred = MSB of S[sindex].2BitCounter
10 |   else
11 |   |   gPred = 00 /* not-taken default prediction */
```

Dealiasing Side Cache: The role of the side cache is to hold the prediction information of all PHT entries for which interferences occur. Each valid side cache entry is associated with some PHT entry in which an interference occurs. Whenever a side cache entry is created, the corresponding PHT index (GBHR or LBHR) is stored as well. To store the prediction information, each entry contains a 2-bit counter. Additionally, each entry contains a valid bit to indicate whether it contains a valid PHT entry.

Fetch Stage Modification: When a branch instance is fetched, the two predictors are active and simultaneously access the shared PHT to retrieve the predictions. The WHO bit for each entry records which predictor updated this entry most recently. Based on the current value of the WHO bit, the prediction is generated. Algorithm 1 shows the detailed steps for the global component. Initially, all WHO bits are 0 and all 2-bit counters are initialized to 00. Hence, 0 as the WHO bit for a PHT entry can mean one of the following.

1) No predictor has accessed this PHT entry as yet.
2) The global predictor owns this entry.

If the global component finds the entry with WHO bit 0, it takes the MSB of the current 2-bit counter value as its prediction. Since the 2-bit counter is initialized as 00, its MSB also gives the default prediction as *not-taken*. The working of the global component is summarized in Algorithm 1.

A similar activity is carried out for the local predictor as well, except that in this case, the shared PHT is accessed with the LBHR and the side cache is looked up when the WHO bit is 0. If the WHO bit of that entry is 1, it takes the MSB of the current 2-bit counter value as the local prediction. If the WHO bit is 0, the side cache is searched with the corresponding LBHR and the prediction is generated as in Algorithm 1.

Branch Retire and Update Stage Modification: Creation of an entry in the side cache and updation of prediction information occur at the branch retire stage, when the outcome of the branch is known. The predictions from the global prediction *gpred* and local predictor *lpred* are known previously from the fetch stage. The state of the choice counter

Algorithm 2: Retire Stage for Global Component

```
1  begin
2      Find PHT entry e = T[gindex]
3      if WHO bit of e is 0 then
4          Update 2-bit counter of PHT entry with actual outcome
5      else
6          if a valid side cache entry S available for GBHR then
7              Update 2-bit counter of S with the actual outcome
8          else
9              /*side cache entry not present, insert new entry */
10             if an empty entry E is found in the cache then
11                 Set VALID of E as 1
12             else
13                 Replace an entry E using LRU replacement
14             Insert GBHR at E, Initialize 2-bit counter of E as
               00
15             Update 2-bit counter of E with the actual outcome
16     GBHR is updated according to the actual branch outcome
```

stored in the choice predictor indicates the predictor chosen for the final prediction in the corresponding fetch stage. If the state is 00 or 01, the global predictor gave the final prediction, otherwise the local predictor gave the final prediction. However, update methods are invoked for both the predictors. The state of the choice counter is updated if and only if the predictions provided by the two predictors are different, decremented if the actual branch outcome is the same as *gpred*, and incremented if it is same as *lpred*. We describe in Algorithm 2 the steps to update the shared PHT/cache entry associated with the global predictor. A similar activity is done for the local component, with the following changes.

1) The PHT is indexed with the LBHR.
2) The side cache creation or update is done when the WHO bit of the PHT entry is 0.
3) Both the GBHR and the LBHR are updated with the actual outcome when a branch retires.

Additionally, if the local component finds that the WHO bit of the PHT entry is 0 and the prediction stored therein is 00 (either due to initialization or as stored by the global component), we transfer ownership to the local component by setting the WHO bit of the PHT entry as 1. This does not harm the global component even though it might have stored 00 in this PHT entry, since when it reaccesses with the same PHT entry and finds the WHO bit set, it will not get a side cache entry and hence, restart with the prediction as 00.

The methodology above creates a cache entry whenever an interference occurs in a PHT entry, thereby eliminating all PHT interferences. We now describe a further optimization.

III. PHT Optimization

We classify interpredictor interference into two different classes: *positive interference* and *negative interference*. An instance of *positive interference* for a shared PHT entry occurs when the prediction given by the current predictor and the state of the 2-bit counter stored at that shared PHT entry by the other component owning the entry are aligned in the same direction (either both taken or both not taken). In such a case, we can avoid the creation of an additional side cache entry and allow both the components to operate on the same PHT entry. An instance of *negative interference* occurs when they are not aligned in the same direction. The different scenarios for positive and negative interference are summarized in Table I.

TABLE I
CASES FOR INTERFERENCE

Current Branch direction	Counter State for owning predictor = 11/10	Counter State for owning predictor = 00/01
taken (1)	Positive Interference	Negative Interference
not taken (0)	Negative Interference	Positive Interference

In the previous method, we disallow any interference, hence the benefit of *positive interference* is lost as well. We now propose another methodology that eliminates only negative interferences and allows the components to work on the same PHT entry as long as their predictions match. In such a case, a smaller number of cache allocations is expected to happen. We now describe the modifications needed in the fetch and retire stages to incorporate this enhancement.

Fetch Stage Modification: In the fetch stage, for the global predictor, if the WHO bit of the corresponding PHT entry is 1, the side cache is looked up. If a valid side cache entry is available for that PHT index, the MSB of the 2-bit counter there gives the prediction. If no valid side cache entry is available for that PHT index, it indicates either no interference occurred till that point or only instances of positive interference occurred between the components for that entry. At the time of positive interference, the 2-bit counter of the PHT index was updated. Hence, to get the benefit of the positive interference, the MSB of the 2-bit counter stored in that PHT index is taken as the global prediction. A similar action follows for the local component as well.

Branch Retire and Update Stage: The methodologies for creating an entry in the side cache and corresponding prediction information update for a branch instance are now different for both the components. We explain the action taken for the global component in the retire stage. As earlier, if the WHO bit of the corresponding entry is 0, the shared PHT is looked up, and the action is exactly as outlined in the first case. However, if the WHO bit is 1, it indicates that the global component does not own the entry and there is an interference. This interference can be positive or negative. We first check for a valid side cache entry for the GBHR. If found, we update the 2-bit counter of that entry according to the actual branch outcome. If the side cache entry for this GBHR is not available, we check for the interference type. We create a new side cache entry only if the interference is negative. For the local component, the methodology is similar.

IV. Implementation and Results

We compare the prediction accuracy of our design with 3 others, as summarized in Table II as (a), (b), (c), Design-1, and Design-2. The choice predictor in each case has 4096×2 bits. For iso-storage comparison, we scale up the PHT in (b) to obtain (c) with a PHT of size 8192×2 bits. For accessing this shared PHT, a 13-bit index is needed. Hence, BHT stores a 13-bit LBHR in (c). All designs are implemented inside the CBP-5 [4] predictor evaluation infrastructure. After execution, it reports the mispredictions per kilo instructions (MPKI). In Design-1, once the local predictor gains ownership of a PHT location through the WHO bit, the ownership does not change

TABLE II
DESIGN DETAILS

Design Details	BHT (Accessed By 9 bit PC)	# of PHT	PHT (2^n * 2-bit)	Side Cache
(a) Classical multi-component Predictor	$2^9 * 12$	2	$2^{12} * 2$	No
(b) Shared PHT multi-component Predictor-1	$2^9 * 12$	1	$2^{12} * 2$	No
(c) Shared PHT multi-component Predictor-2	$2^9 * 13$	1	$2^{13} * 2$	No
Our multi-component predictor Design 1 : without optimization Design 2 : with optimization	$2^9 * 12$	1	$2^{12} * 2$	32 entry fully associative Each entry 15 bits (1 Valid bit + 12-bit TAG + 2-bit counter) LRU replacement policy

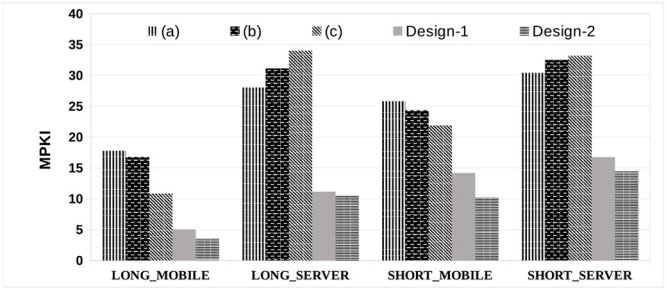

Fig. 2. Average MPKI for different implementations.

TABLE III
CACHE ENTRY CREATION STATISTICS AND ENERGY RESULTS

CBP-5 Trace	Side Cache Allocation		Energy Improvement(%)		Utilization in our Designs	
	Design-1	Design-2	Design-1	Design-2	PHT (%)	BHT (%)
LONG_MOBILE	6920140	1195691	17	18	99	72
LONG_SERVER	56531536	5051012	21	23	100	75
SHORT_SERVER-8	5884486	1512181	18	21	99	69
SHORT_MOBILE-8	38827839	4472732	12	13	100	76

further. To address this unfairness toward the global component, we perform a periodic reset of the WHO bits after every 10-million branch instructions.

Fig. 2 shows the average MPKIs of (a), (b), (c) and our designs for CBP-5 trace categories. MPKI reduces when a side cache is used along with the shared PHT compared to the MPKI of the shared PHT designs without the side cache. With a 32-entry side cache, the MPKI reduces for both Design-1 and Design-2, even when compared to (c). The increase in storage in (c) leads to less number of instances of positive interference due to increase in the number of index bits and thereby, the benefits of the extra storage are not achieved as expected. Further, we compare the number of cache entries allocated in Design-1 and Design-2. Results in Table III shows that Design-2 indeed fares better. The average PHT and BHT utilization is quite high in our proposed designs for each trace category, thereby justifying the sizes chosen for each.

To evaluate the area and power consumption, we develop Verilog RTL and use the Veriwell simulator [1] to verify the designs. The DC [7] is used to synthesize our designs with TSMC 90-nm libraries. Our design achieves a 1.6% area benefit over (a). The 2 PHTs in (a) are synthesized as 2 single-port SRAMs, while in the rest, the shared PHT is synthesized as a dual-port SRAM to enable simultaneous access by the 2 components in the fetch and retire stages. The synthesis of the BHT and the choice tables are similar in all the 3 designs, while the cache in our design is a fully associative SRAM. The PHT and the side cache are looked up in parallel in the fetch stage of the pipeline, hence the critical path is unchanged. The PHT update in the retire stage is not on the critical path as well. We

TABLE IV
MPKI FOR CBP-4

Designs	Average MPKIs				
	(a)	(b)	(c)	Design 1	Design 2
LONG-SPEC2K6	36	31	29	10	7
SHORT-FP-1	3.2	3	2.5	1.7	1.5
SHORT-INT	28.6	29.8	30	10	9
SHORT-MM	32	39	31	9	8
SHORT-SERVER	28	30	29	14	13

also present our analysis on using a Verilog module to measure per access read energy for both the SRAM and the side cache. We collect the total number of read and write accesses for all the designs from the CBP-5 program traces and multiply these numbers with the per access energy obtained from DC to obtain the total energy expended in each CBP-5 program. Table III shows the average % energy improvement for our design over the nonshared implementation for each CBP-5 program category. Further, we experiment with the CBP-4 traces, some of which are derived from SPEC 2006 [3]. Table IV presents a comparison of the average MPKI for each trace category with the different designs. Our proposal significantly outperforms the other designs in this case as well.

V. CONCLUSION

We propose a storage-efficient design for multicomponent branch predictors. The main highlight of our proposal is to enable the predictor components work on a shared prediction table, while maintaining a tiny dealiasing cache to resolve entries on which the predictors collide. We believe this design can be useful for resource constrained embedded devices.

REFERENCES

[1] *Veriwell Simulator*. Accessed: Sep. 9, 2019. [Online]. Available: https://veriwell-verilog-simulator.soft112.com/
[2] M. I. Bielby. *Ultra Low Power Cooperative Branch Prediction*. [Online]. Available: https://era.ed.ac.uk/handle/1842/14187
[3] CBP-4. (2014). *4th JILP Workshop on Computer Architecture Competitions*. [Online]. Available: https://www.jilp.org/cbp2014/
[4] CBP-5. (2016). *5th JILP Workshop on Computer Architecture Competitions*. [Online]. Available: https://www.jilp.org/cbp2016/
[5] P.-Y. Chang, E. Hao, and Y. N. Patt, "Alternative implementations of hybrid branch predictors," in *Proc. 28th Annu. Int. Symp. Microarchit. (MICRO)*, 1995, pp. 252–257.
[6] M. Das, A. Banerjee, and B. Sardar, "An empirical study on performance of branch predictors with varying storage budgets," in *Proc. IEEE 7th Int. Symp. Embedded Comput. Syst. Design (ISED)*, 2017, pp. 1–5.
[7] G. Dupenloup, "Automatic synthesis script generation for synopsys design compiler," U.S. Patent 6 836 877, Dec. 28, 2004.
[8] C. Egan, G. B. Steven, W. Shim, and L. Vintan, "Applying caching to two-level adaptive branch prediction," in *Proc. IEEE Euromicro Symp. Digit. Syst. Design*, Warsaw, Poland, 2001, pp. 186–193.
[9] C. Egan, G. Steven, and L. Vintan, "Cached two-level adaptive branch predictors with multiple stages," in *Proc. Int. Conf. Archit. Comput. Syst. (ARCS)*, 2002, pp. 179–191.
[10] M. Evers, P.-Y. Chang, and Y. N. Patt, "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches," in *Proc. ACM Comput. Archit. News (CAN)*, vol. 24, pp. 3–11, 1996.
[11] R. E. Kessler, "The alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar./Apr. 1999.
[12] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 105–116, 2000.
[13] G. H. Loh, D. S. Henry, and A. Krishnamurthy, "Exploiting bias in the hysteresis bit of 2-bit saturating counters in branch predictors," *J. Instruct. Level Parallelism*, vol. 5, pp. 1–32, Jun. 2003.
[14] A. Seznec, "A new case for the TAGE branch predictor," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2011, pp. 117–127.
[15] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design tradeoffs for the alpha EV8 conditional branch predictor," in *Proc. ACM Comput. Archit. News (CAN)*, vol. 30, pp. 295–306, 2002.