Journal of Instruction-Level Parallelism 9 (2007) 1-19 Submitted 4/07; published 5/07

Fused Two-Level Branch Prediction with Ahead Calculation

Yasuo Ishii Y-ISHII@BC.JP.NEC.COM
3rd engineering department, Computers Division, NEC Corporation
1-10, Nisshin-cho, Fuchu-shi, Tokyo, Japan, 183-0036

Abstract

In this paper, we propose a Fused Two-Level (FTL) branch predictor combined with an Ahead Calculation method. The FTL predictor is derived from the fusion hybrid predictor. It achieves high accuracy by adopting PAp-base Geometrical History Length (p-GEHL) prediction, which is an effective prediction scheme exploiting local histories. The p-GEHL predictor has several prediction tables indexed from independent functions of the local branch histories and branch addresses. The prediction is computed through the summation of values read from the prediction tables. This approach effectively uses limited budget and allows accurate predictions. The Ahead Calculation is an effective implementation scheme for neural predictors exploiting local histories such as the p-GEHL predictor. This scheme is the so-called pre-calculation method. The prediction result is computed when a previous branch with the same address was predicted, and the result is stored in a RAM, which is called Local Prediction Cache (LPC). The LPC reduces prediction latency since the predictor only has to read the RAM by branch address instead of computing the prediction through adder trees. We optimized our FTL branch predictor for the CBP-2 realistic track infrastructure. This optimized-FTL branch predictor with Ahead Calculation achieved 3.466 MPKI with a 262,400-bit budget.

1. Introduction

This study focuses on branch predictors that exploit local histories. The local history correlates with accurate branch prediction as demonstrated by the fact that three finalists of the First Championship Branch Prediction Competition (CBP-1) exploited local histories [7], [10], [14]. However, prediction with local histories has difficulties. One of the most significant problems is the long latency of the prediction phase. Long prediction latency prevents the processor from improving its performance. We describe an effective prediction method using local history and a feasible prediction structure that has sufficiently short latency to exploit local history.

Section 2 introduces the principles and features of the FTL predictor, a derivative of the fusion hybrid branch predictor. In section 3, we propose an Ahead Calculation method to reduce the latency of complex local predictors. In section 4, we describe the Enhanced Folded Indexing method, which is a complexity-effective indexing function for the global predictors. Other optimization tricks for the CBP-2 are described in section 5. In section 6, we describe a FTL branch predictor optimized for the CBP-2 contest rule, evaluate it in several configurations, and present the simulation results. Other analyses are shown in section 7. Finally, we conclude in section 8.

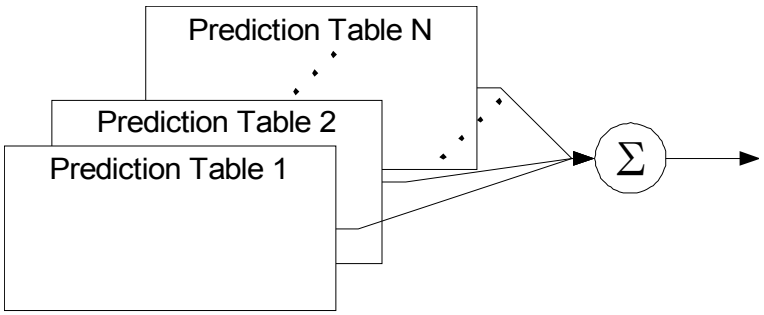

Figure 1: Fused Two-Level Branch Prediction.

2. Fused Two-Level (FTL) Branch Prediction

FTL Branch Prediction is derived from the fusion hybrid branch predictor, which has been used for accurate predictions in previous studies [9], [14]. These prediction algorithms combine several independent prediction results by skewing methods. This eliminates the weak points of each prediction algorithm. In order to achieve a more accurate prediction we analyze traditional branch predictors by dividing them into four groups. We then propose a more effective branch prediction algorithm based on the analysis, called the FTL branch prediction.

FTL predictor can achieve more accurate predictions than other fusion hybrid branch predictors; however it has some implementation difficulties. One problem is the long prediction latency since it is detrimental to processor performance. However, here we focus on prediction algorithm, and later implementation difficulties such as the prediction latency will be discussed later in the following sections.

2.1. Four Groups of Two-Level Branch Predictors

Overview of the FTL algorithm is shown in Figure 1. The structure of this predictor is similar to the known fusion hybrid base predictors. This predictor has many prediction tables, each indexed with independent functions. These generate prediction through summations of values from each prediction table. In this scheme, the indexing function for each sub-predictor is one of the critical elements in prediction accuracy, since the prediction result is determined by these indexing functions and tables. To explore the best indexing function, we divide the known two-level branch prediction algorithms into four groups. These groups are shown in Table 1.

In this table, the lines Global and Local indicate whether the predictor exploits global or local history. The two columns, Total and Partial indicate whether the indexing function involves all recent histories when it exploits an old history, and that indexing function does not involve recent histories (Figure 2). For example, the gshare predictor belongs to Global/Total category because it exploits (1) global history, and (2) H[1:N] when its global history length is N. On the other hand, the Nth neuron of a simple perceptron predictor is computed by a single bit of global history H[N], which is not the recent history. This makes perceptron predictor a Global/Partial predictor.

Each group has advantages and disadvantages. To cover each disadvantage, FTL employs three groups of predictors from Global/Total, Global/Partial, and Local/Total. We describe indexing policy of these three groups in the rest of this section.

	Total	**Partial**
Global	GAp[13], Gshare[12], GEHL[2]	Path-Base [6], Piecewise [7], [8]
Local	PAp / Pag[13]	Local Perceptron [5]

Table 1: Group of Known Predictors.

```
0. Boolean Total_Predict (Addr : integer)
1.   sum := 0
2.    for each i
3.      sum := sum + Wi(indexi(Addr, Hist[1:Li]))
4.      /* Each neuron exploits all histories which are newer than H[Li] */
5.    end for
6.   return (sum ≧ 0)
7. End Total_Predict

0. Boolean Partial_Predict (Addr : integer)
1.   sum := 0
2.    for each i
3.      sum := sum + Wi(indexi(Addr, Hist[Li:Li+1]))
4.      /* Each neuron does not exploit any histories which are newer than H[Li] */
5.    end for
6.   return (sum ≧ 0)
7. End Partial_Predict
```

Figure 2: Total Prediction & Partial Prediction.

2.1.1. Global/Total predictor

Indexing functions of the Global/Total group are derived from GEHL predictor [2] since it is one of the most accurate Global/Total predictors. History length for each prediction table is determined by a geometrical series such as L(j) = aj-1 · L(1). The indexing functions also involve path information since it has good correlation with branch accuracy. In this study, we exploit the least significant bit (LSB) of an instruction address as the path information.

Most state-of-the-art predictors such as gshare, bi-mode, 2BC-gskew, and GEHL belong to this group, thus we exploit this group as the main part of our FTL predictor. The other predictors support the Global/Total predictor.

2.1.2. Global/Partial predictor

Indexing functions of the Global/Partial group are derived from the MAC-RHSP predictor [4], which exploits partial histories effectively. An indexing function generates the address of a prediction table by a bit vector, which is a small part of global histories. The length of the bit vector is one of the parameters. This group also exploits the path information for accurate prediction. We also exploit the LSB of an instruction address as the path information. This MAC-

```
0. Boolean Predict (A : integer)
1.   sum := 0
2.    for each i
3.      sum := sum + Wi(indexi(Addr, Hist))
4.    end for
5.   return (sum ≧ 0)
6. End Predict

0. Void Update (A : integer, Outcome : boolean)
1.   if ((p!=Outcome) or (|Sum| < θ))
2.    for each i in parallel
3.      if Outcome = Taken then
4.        Wi(index(Addr, Hist)) :=Wi(indexi(Addr, Hist)) + 1
5.      else
6.        Wi(index(Addr, Hist)) :=Wi(indexi(Addr, Hist)) - 1
7.      end if
8.    end for
9.   end if
10.End Update
```

Figure 3: Predicting and Updating Algorithm.

RHSP approach also reduces the predictor complexity by reducing the number of prediction tables and skewing adders from the classical perceptron predictor [5].

This part affords less destructive collisions of the pattern history tables, since each table involves smaller bit vector than the Global/Total predictors.

2.1.3. Local/Total predictor

The CBP-1 finalists did not exploit this group. However, this group has the potential for effective prediction. The indexing functions of Local/Total groups are the same as for the GEHL predictor, except that these indexing functions exploit local histories. We call this approach PAp-base Geometrical History Length (p-GEHL) prediction. This approach uses a smaller adder tree than the Local/Partial approach since it does not need to employ many tables like a local perceptron.

This group has a serious problem in terms of prediction latency because the local predictor must read two tables—the local history table and pattern history table. This latency decreases the processor performance. The solution to this issue will be discussed in the following section.

2.2. Prediction and Updating Algorithm

The prediction algorithm was already introduced in section 2.1 and the updating algorithms are shown in Figure 3. This prediction algorithm is almost the same as the previous fusion hybrid predictors. An adder tree implements the fusing function. A prediction result is decided from the sign bit of the summation of values read from prediction tables. This scheme is derived from fusion hybrid predictor. The FTL updating policy is also derived from the fusion hybrid updating

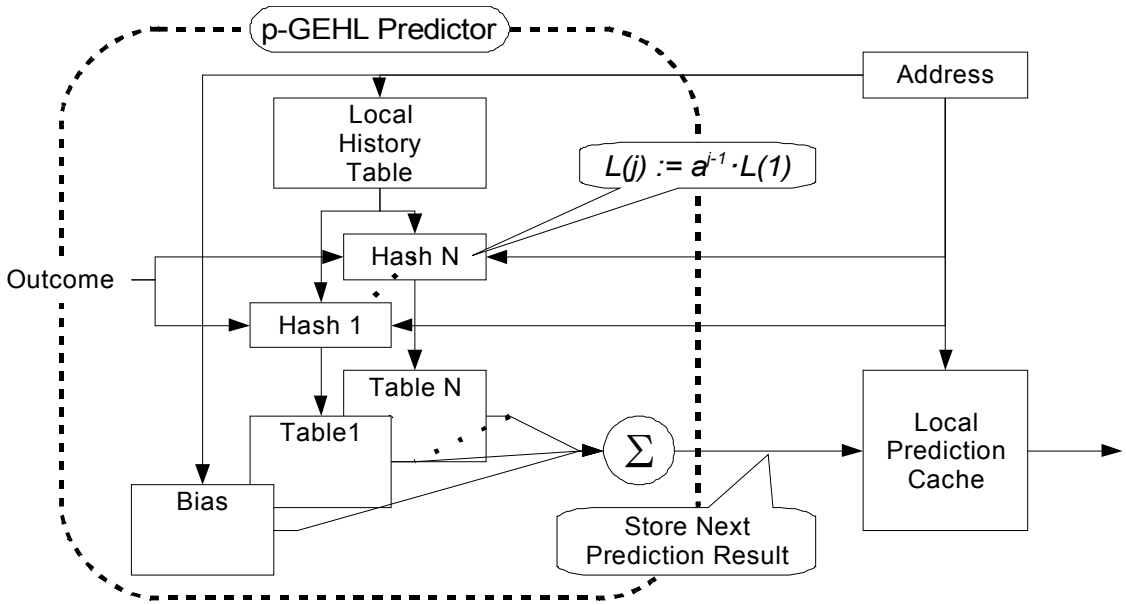

Figure 4: Ahead Calculation Method on the p-GEHL Predictor.

policy. The FTL predictor is updated in case of missing prediction or when the absolute value of summation is smaller than the threshold θ.

3. Ahead Calculation Method

Local predictors suffer from the need to read two tables, producing latency far too long for modern processors. To resolve this issue, we propose an Ahead Calculation, which is an implementation technique that reduces the prediction latency.

3.1. Implementation Issues of Local Predictor

Modern predictors based on skewing fusion method suffer from long prediction latency, since it is difficult to finish their calculation quickly. In particular, it is difficult for a Local predictor, since the Local predictor must access two different tables—one for the local history table and another for the pattern history table. The long latency affects the processor performance. To avoid this latency, the modern predictors employ implementation techniques such as Ahead Pipelining. However, this method cannot be applied for Local predictors because they cannot read the local history table until the target branch instruction is determined [15]. Thus, we propose a new implementation method called the Ahead Calculation method.

3.2. Ahead Calculation Algorithm

In this part, we explain the Ahead Calculation method. This technique reduces the prediction latency of Local predictors. When the predictor makes a prediction, it pre-calculates the next prediction result. This calculated result is stored for the next prediction in an area of RAM, called Local Prediction Cache (LPC). The predictor exploits the stored values when it predicts a branch with the same instruction address. The predictor with Ahead Calculation only need to read the LPC when it predicts a target branch. The prediction latency is then limited to the read latency of

the LPC, whose latency is the same as that of the simple bimodal predictor, since the LPC is a simple RAM. This reduced latency is acceptable for modern processors. Figure 4 provides an overview of the Ahead Calculation.

3.3. Cost Estimation

External cost for Ahead Calculation is only that of the LPC. We exploit a 2048-entry LPC for squeezing for the CBP-2 contest rule and each entry is 7 or 8 bits. This implementation does not require any other costs such as the duplication of computing logic. Because LPC is a simple RAM, it also requires a relatively smaller space and has lower power consumption than computational logics such as duplicated adder trees. Therefore, addition of LPC is not expensive for modern processors.

3.4. Read After Write Hazard Issue

The read after write (RAW) hazard, which can cause inaccurate prediction, can occur with the Ahead Calculation, since the speculative update of the LPC will require several cycles. However, the effect of the performance will be small. The RAW hazard occurs only when the prediction interval for one branch instruction is within a few cycles. Such tight loops will be unrolled in optimized code or filtered by loop counter. This issue was intentionally ignored in the evaluation for the CBP-2 competition, and the performance impact of this RAW hazard is evaluated in Section 7.1.

4. Enhanced Folded Indexing

The FTL predictor exploits long global history lengths, as does the GEHL predictor. To achieve complexity-effective indexing circuit for Global predictors, we propose an enhanced folded indexing function.

Figure 5 shows first complexity-effective folded indexing method for branch predictors, which was proposed in the previous CBP competition [11]. This folded indexing function can fold only one bit-vector series such as the global history. When a predictor exploits other bit-vector information, such as path information, the predictor must use another folded indexing circuit or other complex indexing function.

To combine other information in a single folded indexing function, we extend the original folded indexing function. Figure 6 shows an example of combining a couple of bit vectors in one folding function. The enhanced folded indexing circuit is extended by including another couples of 2-bit exclusive-or circuits to fold other bit-vector information. This same approach can combine three or more bit-vectors by employing exclusive-or pairs. In this indexing function, the implementation cost of the indexing function is still circular shift register and about ten 2-bit exclusive-ors. There is one 2-bit exclusive-or gate in the critical path of the indexing generation. This indexing function provides sufficient complexity-effectiveness for real processor and can be applied to other state-of-the-art predictors, such as GEHL or TAGE predictors.

Pipelined processors must employ some rollback method for missed predictions. We assumed that the rollback should occur only during the commit phase, since the folded indexing function requires in-order update. The predictor employs a pair of these enhanced indexing functions. One is used speculatively and updated during the prediction phase; the other is updated during the

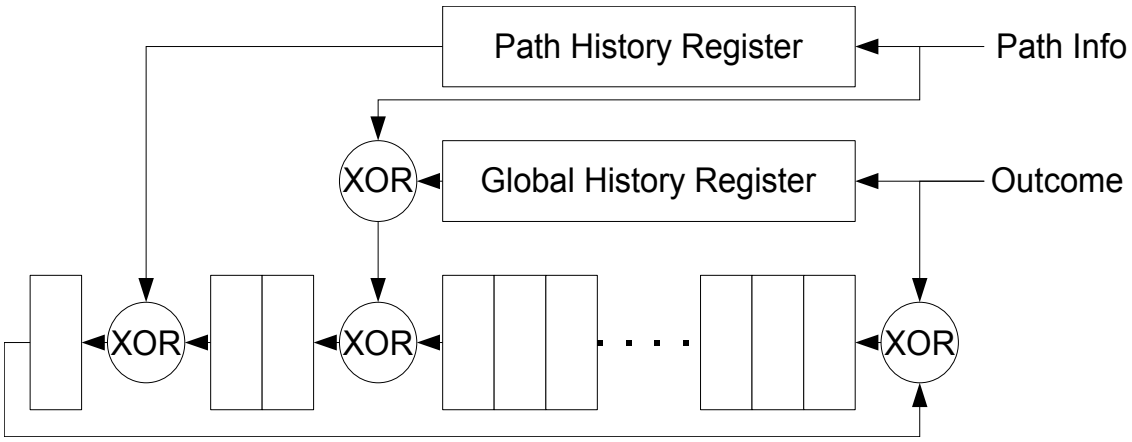

Figure 5: Enhanced Folded Indexing Function.

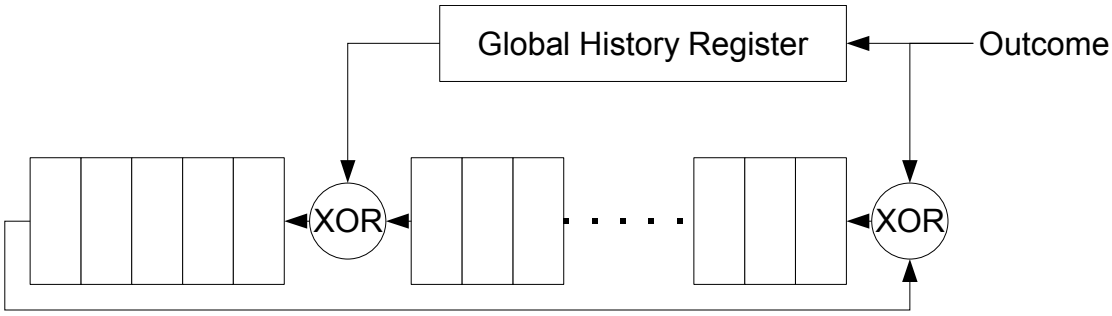

Figure 6: Classical Folded Indexing Function.

commit phase and used for checkpointing. Once the rollback occur, the checkpoint register overwrite the speculative updated register. The external cost of this rollback mechanism is the copy registers, which requires only 10-bit circular shift register and a few pairs of exclusive-or circuits. When the processor employs tandem prediction scheme, it should employ one more checkpointing register for overwriting the speculative predictor.

5. Other Tricks

In this section, we describe other borrowed implementation techniques and simple filtering methods for the FTL predictors. These techniques are not essential for the FTL predictor; however, they can improve branch prediction accuracy.

5.1. Simple Bias Filtering

Simple bias filtering filters strongly biased branches. It is implemented in a 1-bit array. This table is indexed by the branch address and each entry is initialized by 0. When a prediction is required, the predictor reads this array with the branch address. When the value is 0, the predictor generates its prediction based only on the bias table of the FTL predictor. When the prediction result is wrong, the bias table must be updated and the used entry of the filtering array is set to 1 except when the read bias value is zero. The predictor exploits other prediction methods when the value read from the bias filtering array is 1.

In a real processor, each entry of this array should be reset to 0 at the appropriate timing. Without any reset mechanisms, almost all entries of this array will be 1 after some context switches, and then simple filtering mechanism does not work well. We assumed that set-associative structured branch target buffer Hit/Miss information is suitable for the reset timing. However, this resetting method cannot be implemented in the CBP-2 infrastructure because it has no branch target buffer instance. Since almost all modern microprocessors employ the BTB instance, this filtering method is feasible on real processors even if the filter does not have any reset mechanisms in the CBP-2 infrastructure. A case of the BTB-collaborated bias filtering is evaluated in Section 7.2.

Even if the processor cannot provide sufficient branch target buffer entries, there are several information sources such as an instruction cache replacement, which can be used as the reset timing of this simple bias filtering.

5.2. Bias Filtering by Branch Target Buffer

Thus, simple bias filtering should be implemented with reset mechanisms. The most feasible implementation is branch target buffer support, since almost all modern microprocessors employ a branch target buffer. Generally, the miss rate of a set-associative structured branch target buffer is less than that of a branch predictor since an accurate branch prediction becomes meaningless while the next address is not clear. Thus, it serves for the filtering mechanism. The prediction structure is very simple, and is shown in Figure 7. In this branch target buffer, the AlwaysTaken bit is added to each entry as an external branch information. An overview of this enhanced branch target buffer is shown in Figure 7.

The bias filter works when the AlwaysTaken bit is 1 or when the branch target buffer misses the target branch address. In this filter mechanism, the AlwaysTaken bit indicates that the branch result should be Taken; the branch target buffer miss means that the branch prediction should be NotTaken. The state diagram of this filtering algorithm is shown in Figure 8. In this algorithm, no entries of the branch target buffer are replaced while the branch outcome is NotTaken. It contributes to the effective use of limited storage, since any AlwaysNotTaken branches are not put in the branch target buffer, and it is good for filtering AlwaysNotTaken branches.

This feature is not evaluated in the CBP-2 competition since the CBP-2 infrastructure does not employ any branch target buffer instances. However, we evaluate this feature in Section 7.2 since this filtering mechanism is suitable for modern microprocessors.

5.3. Loop Counter

We employ a loop counter to filter the simple loop branch instruction. Our loop counter technique is based on Gao's loop counting method [10]. It is a set-associative cache-structured loop counter. Each entry of this loop counter contains loop count information with a confidence value. This loop counter is feasible since it is not as complex as a set-associative structured branch target buffer. The optimized FTL predictor employs an 8-way set-associative loop counter. Gao's loop counter detects loops from backward branch instructions only, but our evaluation indicate that prediction accuracy is better when the counter detects forward branches as well. Moreover, our loop counter detects loop patterns from all conditional branches.

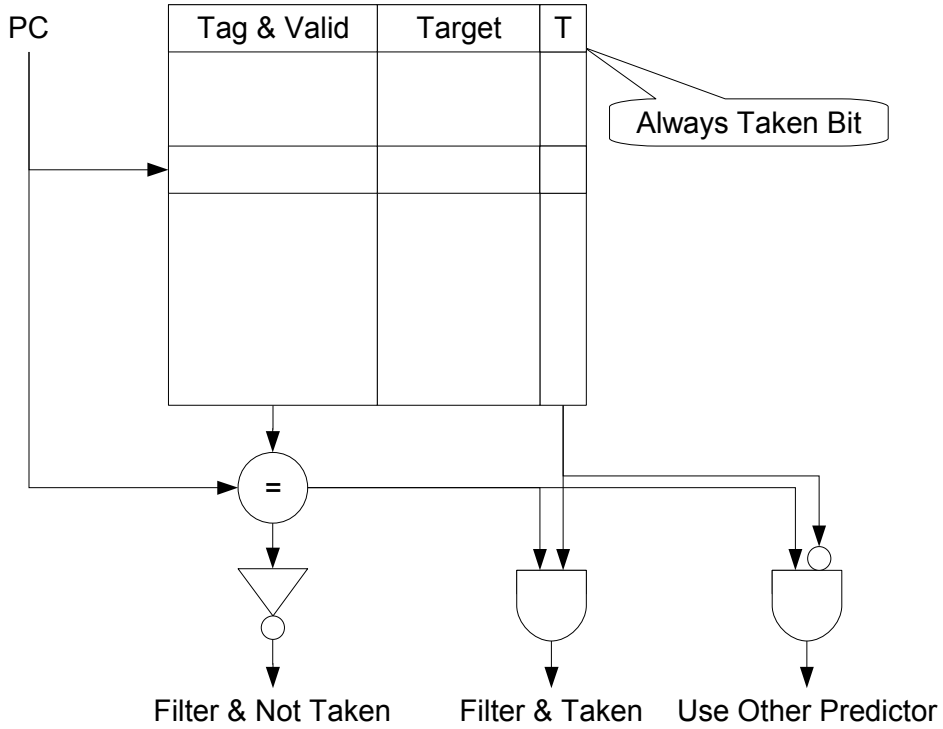

Figure 7: Simple Bias Filtering Implementation with Branch Target Buffer.

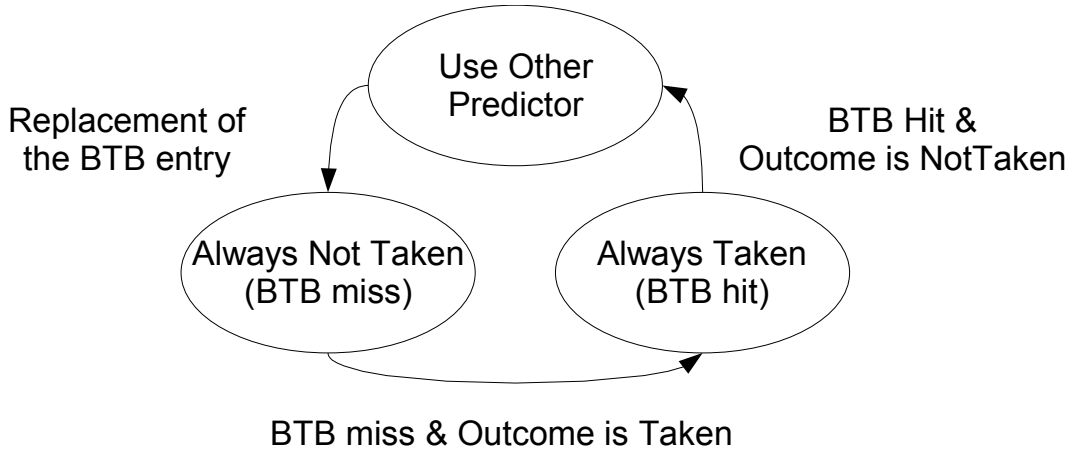

Figure 8: State Diagram of BTB-base Bias Filtering.

5.4. Dynamic Threshold Fitting

The value of the updating threshold θ significantly affects the accuracy of a predictor whose updating policy is derived from the perceptron predictor, and the best threshold value differs among benchmarks. To optimize the threshold value, we employ the dynamic history length fitting. The fitting algorithm is already proposed [2], and this threshold fitting is sufficiently cost effective. We designed it as a 14-bit counter, 7 bits for threshold value and 7 bits for threshold history counter. This threshold counter is updated only when the FTL predictor is updated. When

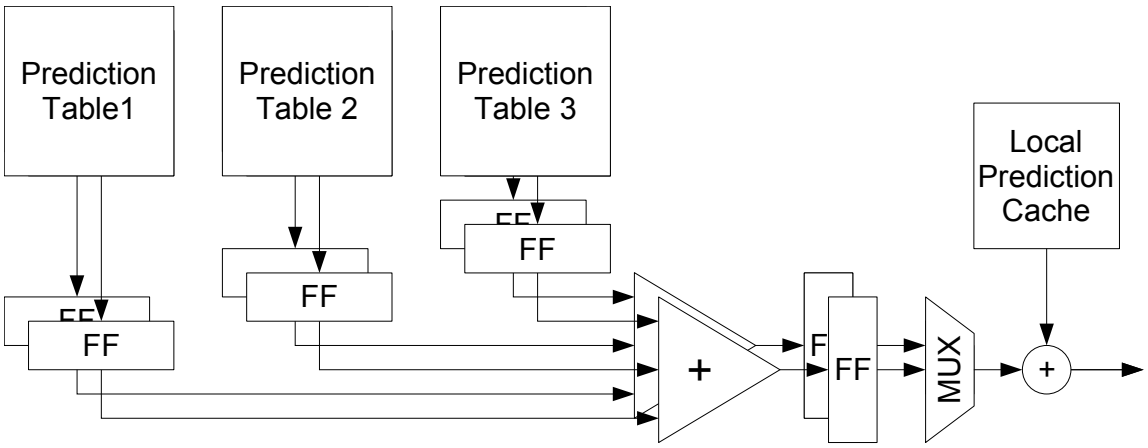

Figure 9: Simultaneous Ahead Pipelining.

the FTL generates a correct prediction, the threshold counter is incremented otherwise the counter is decremented. When the threshold counter overflows or underflows, the threshold value is incremented or decremented.

5.5. Dynamic Adaptation

The three finalists of the CBP-1 competition employed dynamic history length adapting. This study employs a similar dynamic adaptation through exploiting some execution information, including the hit rate of the bias filter and the loop counter, and the number of static conditional branches.

This trick may not be appropriate for realistic track, and therefore, the predictor was also evaluated in a configuration with disabled dynamic adaptation.

5.6. Ahead Pipelining for Global Predictor

The FTL predictor exploits an adder tree so complex that its latency is not tolerated in real processors. To achieve a reasonable latency, the Local predictor exploits Ahead Calculation (explained in the previous section), and the Global predictor employs Ahead Pipelining [1].

There are two major strategies for implementing Ahead Pipelining. One is a systolic-array approach, which is used in a piecewise linear branch predictor [8], and the other is a simultaneously reading approach, which is used in the GEHL predictor [3]. We adopt a hybrid of these two approaches. The predictor reads four values from several tables in each step simultaneously, and computes the intermediate prediction value with a systolic-array approach. The predictor selects an appropriate intermediate prediction result at the last step of Ahead Pipelining. This Ahead Pipelining policy requires computational circuit duplication. The FTL predictor employs four duplications of computational circuits, but is still less complex than the O-GEHL, predictor which employs 16 duplications. Our predictor requires only 56 adders while the Piecewise Linear Branch Predictor requires more than 100 adders for its skewing function. Figure 9 shows the simultaneous reading ahead pipelining, and Figure 10 shows the systolic-array like ahead pipelining.

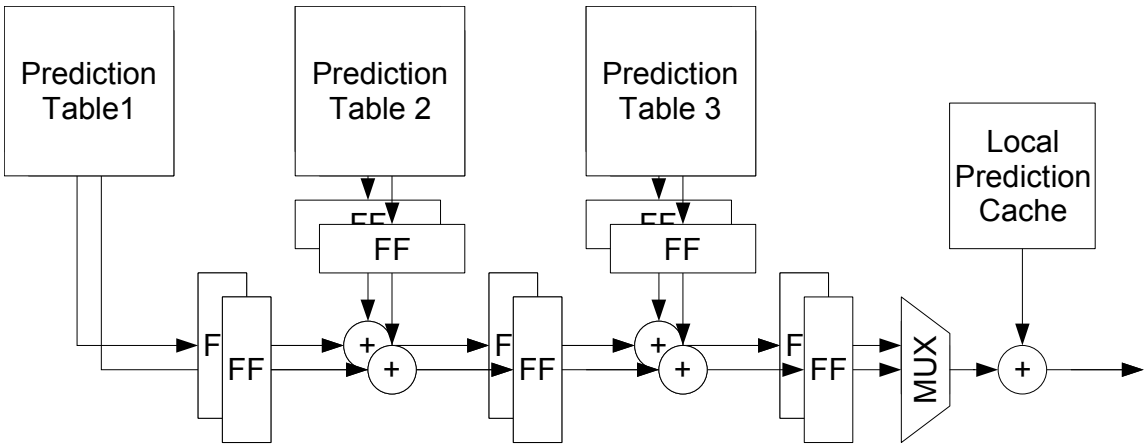

Figure 10: Systolic-Array like Ahead Pipelining.

6. Performance Evaluation in CBP-2 Contest Rule

Here we evaluate some features of the FTL predictor. All of the evaluations are based on the CBP-2 infrastructure and distributed traces, thus these results are shown in terms of the number of missed predictions per 1000 instructions (MPKI).

In this section, we evaluate the FTL predictor optimized under the CBP-2 contest rules. This means the predictor is restricted by the rule of the CBP-2 realistic track, such as the predictor's budget size.

6.1. Optimized-FTL branch predictor

In this part, we described an optimized-FTL (o-FTL). The o-FTL predictor is an optimized configuration for the CBP-2 realistic track rules. The predictor exploits several techniques explained in previous sections. For example, the o-FTL employs the simple bias filter and the loop counter for anti-aliasing. Figure 11 provides an overview of the o-FTL branch predictor.

The FTL configuration is divided into Global and Local parts for effective implementation. The Global part is implemented by Ahead Pipelining structure (explained in the previous section). The Local part is implemented by Ahead Calculation with an LPC. The prediction result of the FTL is computed through the addition of the results of these parts. The FTL prediction is filtered by the bias filter and the loop counter. A prediction of the FTL is used only when both filters miss. The FTL predictor is never updated unless all filters miss the prediction.

6.1.1. Predictor Configurations

The predictors were evaluated in three configurations, Systolic, Simultaneous, and No Ahead Pipe. These configurations vary in complexity. The features of each predictor are shown in Table 2. The Systolic indicates systolic-array like hybrid Ahead Pipelining. It is the most reasonable configuration for modern processor, since it produce its prediction with 9 pipeline stage. Each pipeline stage is sufficiently simplified for modern pipelined processor. The Simultaneous configuration adopts simultaneous reading hybrid Ahead Pipelining which is similar to simultaneous pipelining. It read 4 values simultaneously and calculate prediction through adder tree. Its adder tree has 3 pipeline stages. The simultaneous configuration adopts some complex structures such as dynamic adapting; however it is still implementable. The No Ahead Pipe shows the best theoretical case without considering prediction latency.

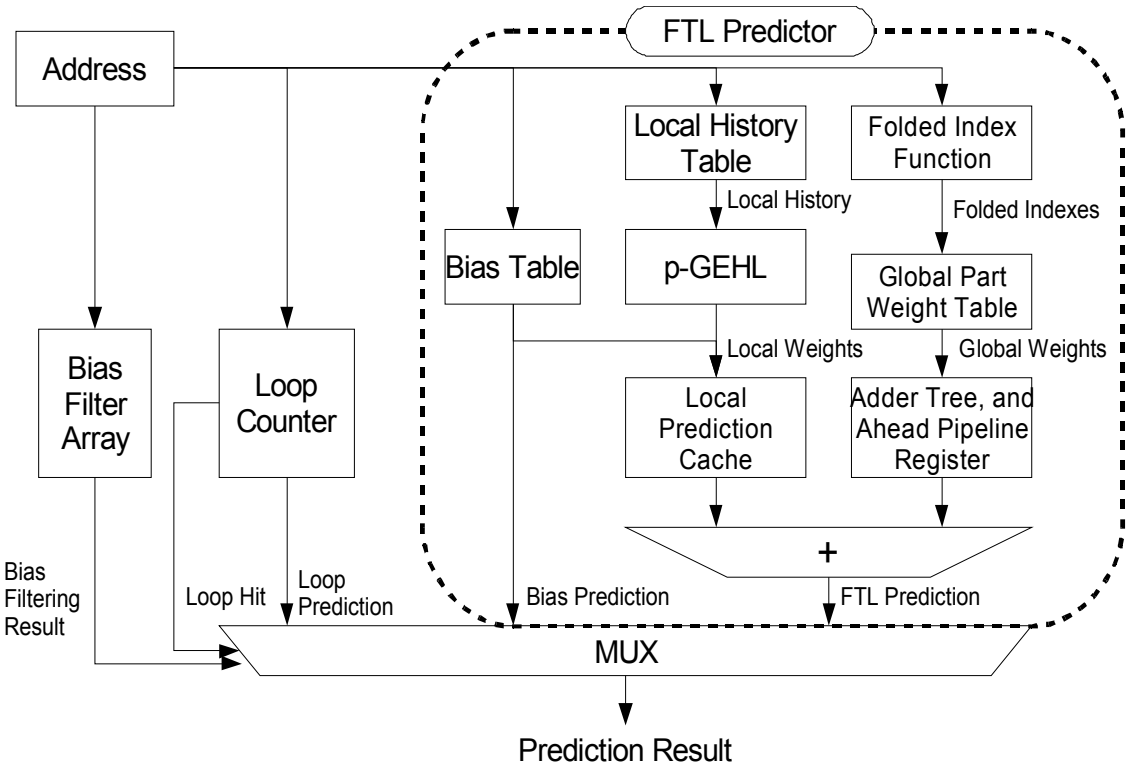

Figure 11: Overview of the Optimized FTL Branch Predictor.

	Systolic	Simultaneous	No Ahead Pipe
Ahead Pipelining	Systolic-array hybrid	Simult-reading hybrid	Disable
Ahead Calculation	Enable	Enable	Disable
Bias Filtering	Enable	Enable	Enable
Indexing Functions	Enable	Enable	Enable
Dynamic Thres Fit	Enable	Enable	Enable
Dynamic Adapting	Disable	Enable	Enable
Weight Boosting	Disable	Enable	Enable

Table 2: Predictor Configuration.

6.1.2. Budget Counting

Budget counting is shown in Table 3. No Ahead configuration employs 18 tables, and the others employ 17 tables and an LPC for Ahead Calculation. The LPC entry of the Simultaneous configuration requires only 7 bits, as the LSB is rounded. This rounding method enables us to squeeze in the constraint of the hardware budget size. Fewer than 262,400 bits, are used for each of the three predictors.

6.1.3. Simulation Results

The detailed results for each benchmark are shown in Table 4 and Figure 12. We also evaluated two branch predictors, a GEHL predictor and a Ghybrid predictor. GEHL was proposed in the

	Systolic	Simultaneous	No Ahead Pipe
Bias Filter	4096 entry * 1 bit	4096 entry * 1 bit	4096 entry * 1 bit
Bias Table	2048 entry * 6 bit	2048 entry * 6 bit	2048 entry * 6 bit
Prediction Table	17 table 2048 entry * 6 bit	17 table 2048 entry * 6 bit	18 table 2048 entry * 6 bit
LPC	2048 entry * 8 bit	2048 entry * 7 bit	0 bit
Indexing Function	398 bit	760 bit	380 bit
Dyn Threshold Fitting	14 bit	14 bit	14 bit
Ahead Pipeline Reg	4 dup * 9 depth * 11 bit	4 dup * 3 depth * 11 bit	0 bit
Ahead Pipeline Address	9 depth * 11 bit	3 depth * 11 bit	0 bit
Global History	121 bit + 3 * 65 bit	201 bit + 3 * 81bit	201 bit + 3 * 81bit
Local History	1024 entry * 16 bit	1024 entry * 16 bit	1024 entry * 16 bit
Loop Counter	6 entry * 8-way * 58 bit	8 entry * 8-way * 58 bit	16 entry * 8-way * 58 bit
Adaptive Info	0 bit	2 bit	2 bit
Performance Counter	0 bit	160 bit	160 bit
Total Budget Size	262055 bit	261257 bit	262376 bit

Table 3: Budget Count for Each Configuration.

previous competition, and this predictor is one of the state-of-the-art predictor. Ghybrid predictor is one of the configurations of the FTL predictor, but it does not exploit Local/Total prediction. It combines two groups of the two-level branch predictor—Global/Total and Global/Partial. These two configurations also employ an 8-way 8-entires loop counter, 4K-entry simple bias filtering array for anti-aliasing, but it does not employs any implementation techniques such as an ahead pipelining as this does not improve the prediction accuracy. Other configurations, which are Systolic, Simultaneous, and Ideal, are already described in previous part of this section.

A comparison of GEHL and Ghybrid shows the performance impact of Global/Total and Global/Partial hybrid approaches. Ghybrid configuration improves the missed prediction rate by 2.2% from simple Global/Total predictor. The comparison of Ghybrid and the other FTL predictors shows the performance impact of the Local predictor. Systolic configuration, which is the most realistic FTL predictor, improves the prediction accuracy by 3.8% even though Systolic configuration waste its budget in LPC, which is not essential for accurate prediction.

Furthermore, the overall missed prediction rate is reduced compared with existing predictors including previous CBP's finalists[1]. The Ahead Pipelined FTL predictor increases the missed prediction rate from 0.045 MPKI to 0.091 MPKI. However, this prediction accuracy is still better than that of first CBP's finalists.

As the o-FTL predictor exploits some unrealistic features, such as Dynamic Adaptation and Simple Bias Filtering requiring reset mechanism, which the o-FTL predictor does not employ, we evaluated the effect of these features. The results are shown in Table 5. This table shows that the optimized-FTL predictor still improves the prediction accuracy, even if the predictor cannot employ some tricks.

[1]The finalists of CBP-1 have been optimized by the author with the new CBP-2 infrastructure. In this optimization, some important tricks such as Gao's dynamic adaptation [10] were disabled to meet the CBP-2 requirement.

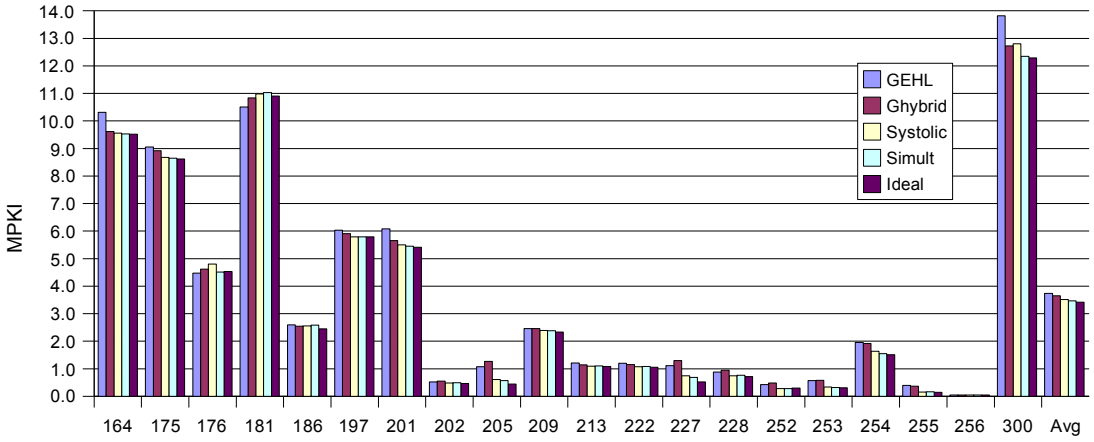

Figure 12: Prediction Accuracy for Distributed Traces.

	GEHL	Ghybrid	Systolic	Simult	Ideal
164.gzip	10.310	9.614	9.554	9.530	9.519
175.vpr	9.048	8.916	8.676	8.644	8.616
176.gcc	4.476	4.616	4.802	4.513	4.529
181.mcf	10.503	10.840	10.985	11.026	10.903
186.crafty	2.590	2.547	2.553	2.582	2.454
197.parser	6.035	5.907	5.793	5.790	5.784
201.compress	6.078	5.653	5.497	5.450	5.416
202.jess	0.519	0.553	0.484	0.493	0.465
205.raytrace	1.071	1.263	0.606	0.571	0.441
209.db	2.457	2.461	2.395	2.378	2.333
213.javac	1.209	1.146	1.094	1.101	1.081
222.mpegaudio	1.201	1.150	1.069	1.086	1.054
227.mtrt	1.112	1.294	0.747	0.681	0.524
228.jack	0.876	0.952	0.749	0.768	0.718
252.eon	0.426	0.486	0.276	0.283	0.297
253.perlbmk	0.570	0.579	0.333	0.319	0.307
254.gap	1.960	1.913	1.638	1.553	1.506
255.vortex	0.390	0.369	0.153	0.160	0.145
256.bzip2	0.044	0.048	0.046	0.045	0.046
300.twolf	13.819	12.720	12.798	12.349	12.292
Average	3.735	3.651	3.512	3.466	3.422

Table 4: Simulation Result for Distributed Traces.

	Systolic	Simultaneous	No Ahead Pipe
Disable Bias Filtering	N/A	N/A	3.474 MPKI
Disable Dyn Adaptatior	3.512 MPKI	3.514 MPKI	3.473 MPKI
Disable Both Feature	3.542 MPKI	3.542 MPKI	3.490 MPKI

Table 5: Evaluation Result when Disabling Several Tricks.

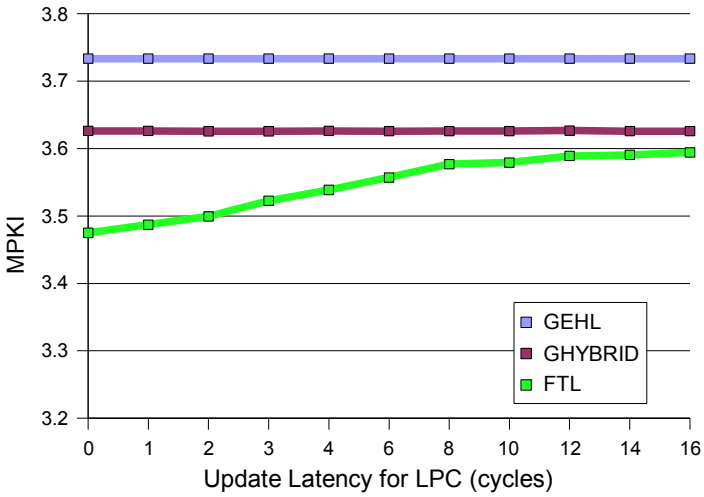

Figure 13: Performance Impact of the RAW Hazard.

7. Performance Analysis of the FTL predictor

We have introduced features of the FTL predictor. However, FTL predictor has some unclear points in its implementation, such as the effect of the RAW hazard of an LPC, the effect of the number of components, and the simple bias filtering. In order to clarify these points, we analyzed the performance of the FTL predictor in several ways. The evaluations were made with the systolic-array like ahead-pipelined FTL, since its configuration is the simplest in three configurations.

7.1. Performance Impact of Read After Write Hazard of LPC Update

In this section, we evaluate the effect of the RAW hazard, which occurs during the speculative update of the LPC. When the FTL predictor generates its prediction, the local history table and local prediction cache are updated speculatively. However, this speculative update leads the RAW hazard. During the LPC's speculative update phase, the FTL predictor must calculate the next prediction result. This calculation requires several processor cycles since it comprises two RAM accesses and an adder tree of p-GEHL predictor.

To evaluate the negative effects of RAW hazard on the performance, we add some latency for the LPC update in the CBP-2 infrastructure. In this environment, the LPC is updated after parameterized latency. This parameter is defined as the number of processor cycles, and it is assumed that the processor fetched one branch instruction during each cycle. When a missed prediction occurs, the speculative update, which is not yet written back to the LPC, is written back immediately. Rollbacks require pipeline squash and this provides adequate time for an LPC update.

We evaluate the RAW impact in 0 to 16 cycle update latency of the LPC since an earlier study [6] shows that the latency of a large adder tree is about 2 to 8 cycles. If the latency is less than 8 cycles, the FTL predictor with Ahead Calculation still has sufficient advantages over the GEHL and Ghybrid predictors which exploit only global information.

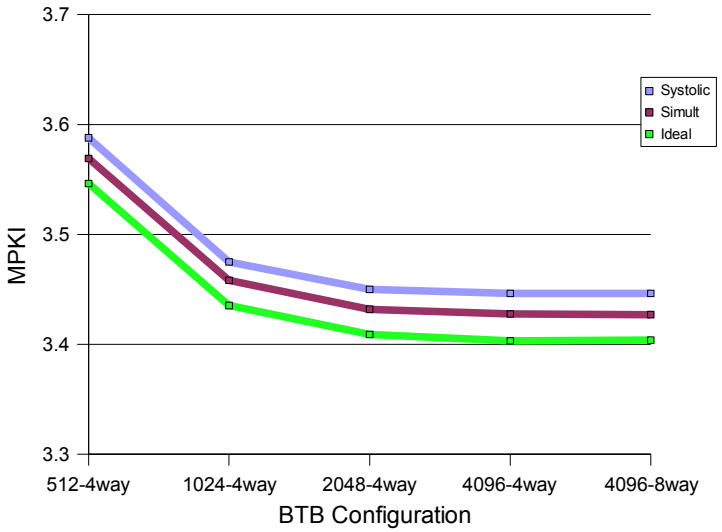

Figure 14: Performance Impact of the Number of BTB entries.

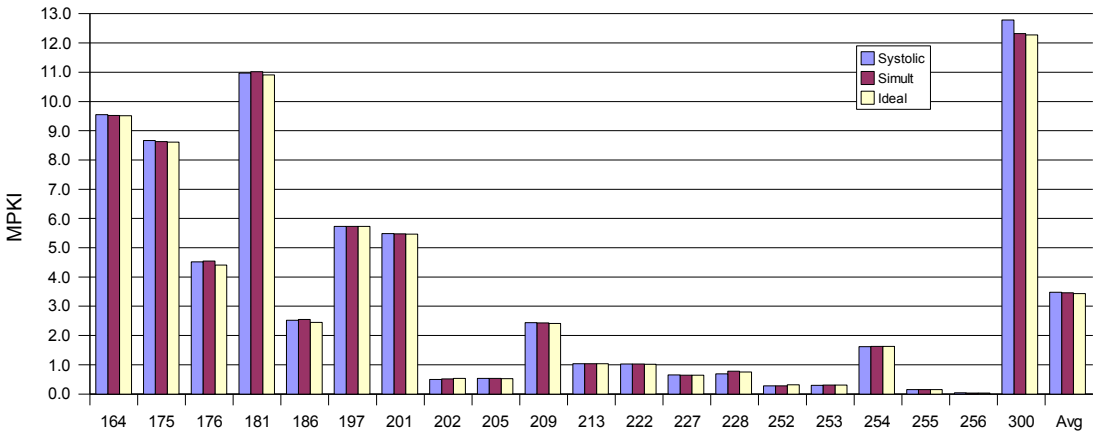

Figure 15: Performance Evaluation of the BTB-base Bias Filtering (4K entry BTB).

7.2. Performance Impact of BTB-base Bias Filtering

In the CBP-2 competition, we exploited simple bias filtering, but we also mentioned that this simple bias filtering requires a reset mechanism in the case of the predictor is implemented in a real microprocessor. For this reset mechanism, we already mention that the predictor should exploit hit, miss, and replace information of the branch target buffer (BTB). This filtering mechanism will be feasible in modern processors since most modern processors employ a branch target buffer for the effective next address generation.

We evaluate the performance impact of BTB-base bias filtering in the CBP-2 infrastructure. The number of BTB entries is used as the parameter. The results of this evaluation are shown in Figure 14. In this result, the performance improves in a BTB with 4096 or more entries, and decreases for the BTB with 2048 entries. One cause of this performance degradation is that a FTL

	Global/Total	Global/Partial	Local/Total	Bias
6 Components	4	0	1	1
12 Components	6	4	1	1
23 Components	9	10	3	1

Table 6: The configuration of the number of components.

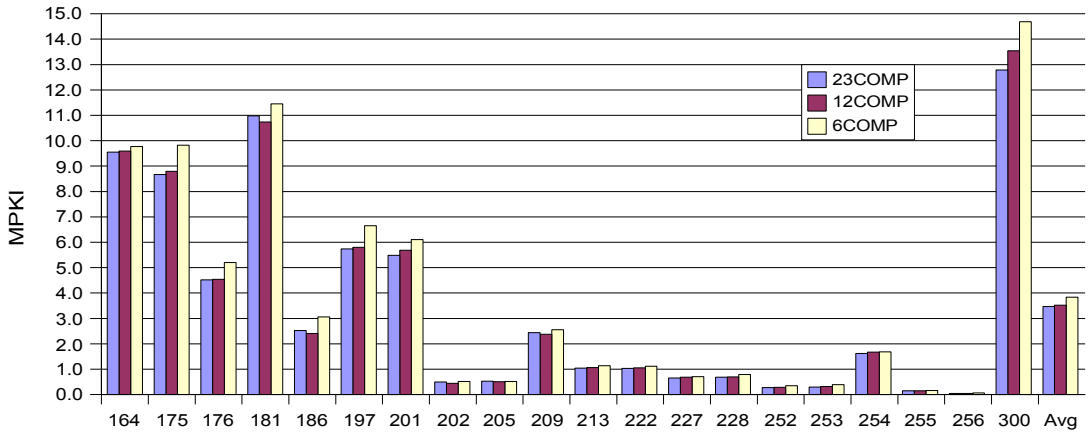

Figure 16: Performance Impact of the Number of Components.

prediction with a 32 KB budget is too accurate for the BTB with 2048 entries. The BTB-base filtering with 2048 entries should be used with simpler predictor such, as a GEHL predictor with 8KB budget, since it cannot achieve such accurate predictions.

7.3. Performance Impact of the Number of Components

The optimized FTL predictor for the CBP-2 competition consists of 23 components. A number of components make the computational logic complex; thus we evaluate the performance impact of the number of components.

The effect of the number of components is evaluated for three configurations of the systolic-array like FTL configuration. These three configurations have 23, 12, and 6 components configuration respectively. The detailed parameters for the configurations are shown in Table 6, and each configuration is evaluated by distributed traces in the CBP-2 infrastructure. The evaluation results are shown in figure 16. The prediction accuracy compared with the 23-component configuration is degraded to 1.3% for 12 components, and 9.5% for 6 components. In the case of the 12-component configuration, the FTL predictor still has several advantages, since the no ahead pipelined GEHL predictor achieves only 3.735 MPKI.

8. Conclusion

In this paper, we proposed a combination of the FTL branch predictor and the Ahead Calculation method, which is an effective implementation scheme for Local predictors, such as the PAp-base GEHL predictor. The FTL predictor combines results generated by several two-level branch predictors. These predictors are derived from earlier works. Ahead Calculation reduces the

latency of the Local branch predictor by pre-calculating the next prediction result. The Ahead Calculation incurs the external cost of a Local Prediction Cache (LPC). However, we show that this storage is not expensive for a branch predictor with 32KB budget size. We also proposed a complexity-effective indexing function, enhanced folded indexing. This indexing function can applied for other branch predictors, such as the GEHL predictor.

The optimized FTL branch predictor achieves accurate predictions with practical prediction latency for real microprocessor. The evaluation results in the CBP-2 infrastructure shows that the optimized FTL predictor reduces the missed prediction rate by 8.4% from GEHL predictor. It also improves the prediction result in an implementable configuration.

Acknowledgments

This work was supported by members of Kei Hiraki's Laboratory in University of Tokyo. The author appreciates their helpful advice.

References

[1] A. Seznec and A. Fraboulet. “Effective Ahead Pipelining of the Instruction Address Generation”, in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 241-252, June 2003.

[2] A. Seznec. “The O-gehl Branch Predictor”, in *The 1st JILP Championship Branch Prediction Competition* (CBP-1), December 2004.

[3] A. Seznec. “Analysis of the O-GEometric History Length branch predictor”, in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pp. 394-405, June 2005.

[4] A. Seznec. “Revisiting the perceptron predictor”. Technical Report RI-1620, IRISA Report, May 2004.

[5] D. A. Jiménez and C. Lin. “Dynamic Branch Prediction with Perceptrons”, in *Proceedings of the Seventh International Symposium on High Performance Computer Architecture*, pp. 197-206, February 2001.

[6] D. A. Jiménez. “Fast Path-based Neural Branch Prediction”, in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 243-252, December 2003.

[7] D. A. Jiménez. “Idealized Piecewise Linear Branch Prediction”, in *The 1st JILP Championship Branch Prediction Competition* (CBP-1), December 2004.

[8] D. A. Jiménez. “Piecewise Linear Branch Prediction”, in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 382-393, June 2004.

[9] G. Loh. “The Frankenpredictor”, in *The 1st JILP Championship Branch Prediction Competition* (CBP-1), December 2004.

[10] H. Gao and H. Zhou. “Adaptive Information Processing: An Effective Way to Improve Perceptron Predictors”, in *The 1st JILP Championship Branch Prediction Competition* (CBP- 1), December 2004.

[11] P. Michaud. “A PPM-like, Tag-based Predictor”, in *The 1st JILP Championship Branch Prediction Competition* (CBP- 1), December 2004.

[12] S. McFarling. “Combining Branch Predictors”, in Technical Report TN-36, Digital Western Research Laboratory June 1993.

[13] T. Yeh and Y. N. Patt. “Alternative Implementations of Two-Level Adaptive Branch Prediction”, in *The 19th Annual International Symposium on Computer Architecture*, pp.124-134, May 1992.

[14] V. Desmet, H. Vandierendonck, and K. D. Bosschere. “A 2bcgskew Predictor Fused by a Redundant History Skewed Perceptron Predictor”, in *The 1st JILP Championship Branch Prediction Competition* (CBP-1), December 2004.

[15] D. Tarjan, K. Skadron, and M. Stan. “An Ahead Pipelined Alloyed Perceptron with Single Cycle Access Time”, in *The Workshop on Complexity-Effective Design* (WCED), pp. 105-112, June 2004.