Front. Comput. Sci., 2013, 7(6): 914–923
DOI 10.1007/s11704-013-2260-x

RESEARCH ARTICLE

A novel architecture for ahead branch prediction

Wenbing JIN (✉)[1,2], Feng SHI[1], Qiugui SONG[2], Yang ZHANG[1,3]

1 School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
2 Department of Trade and Military Industry, North Automatic Control Technology Institute, Taiyuan 030006, China
3 School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang 050018, China

Abstract In theory, branch predictors with more complicated algorithms and larger data structures provide more accurate predictions. Unfortunately, overly large structures and excessively complicated algorithms cannot be implemented because of their long access delay. To date, many strategies have been proposed to balance delay with accuracy, but none has completely solved the issue. The architecture for ahead branch prediction (A^{2}BP) separates traditional predictors into two parts. First is a small table located at the front-end of the pipeline, which makes the prediction brief enough even for some aggressive processors. Second, operations on complicated algorithms and large data structures for accurate predictions are all moved to the back-end of the pipeline. An effective mechanism is introduced for ahead branch prediction in the back-end and small table update in the front. To substantially improve prediction accuracy, an indirect branch prediction algorithm based on branch history and target path (BHTP) is implemented in A^{2}BP. Experiments with the standard performance evaluation corporation (SPEC) benchmarks on gem5/SimpleScalar simulators demonstrate that A^{2}BP improves average performance by 2.92% compared with a commonly used branch target buffer-based predictor. In addition, indirect branch misses with the BHTP algorithm are reduced by an average of 28.98% compared with the traditional algorithm.

Keywords branch prediction, branch speculation, branch target buffer, indirect branch, instruction pipeline

Received August 11, 2012; accepted June 15, 2013

E-mail: wenbbo@sina.com

1 Introduction

Accurate branch prediction is essential in sustaining an efficient pipeline, which is critical to boost the performance of modern processors. Branch prediction is similar to machine intelligence: a predictor employing more complicated algorithms and larger data structures yields higher accuracy. As such, branch predictors become more complicated with larger data structures. Meanwhile, growing chip capacities brought about by advances in industrial technology and material science provide more opportunities for physical implementation. In recent years, international competitions focusing on accuracy but avoiding delay have stimulated predictors to become more complicated. Analysis on predictors proposed in the past decade corroborate this trend [1, 2].

Theoretically, predictors with complicated algorithms and large data structures can achieve prediction accuracies of 98% [1, 2], under the assumption that the predictors can be accessed within a single cycle each time [3, 4]. However, in reality, some existing predictors with conventional algorithms and data structures cannot fulfill every prediction within a single cycle [5]. Particularly, predictors with complicated algorithms and large data structures, such as neural network predictors, are currently unacceptable in the industry because of their overly long delay. As the processor frequency increases, the access delay measured in clock cycles also extends [6, 7]. Figure 1 illustrates different pipeline efficiencies achieved in the simulations for each benchmark at the same

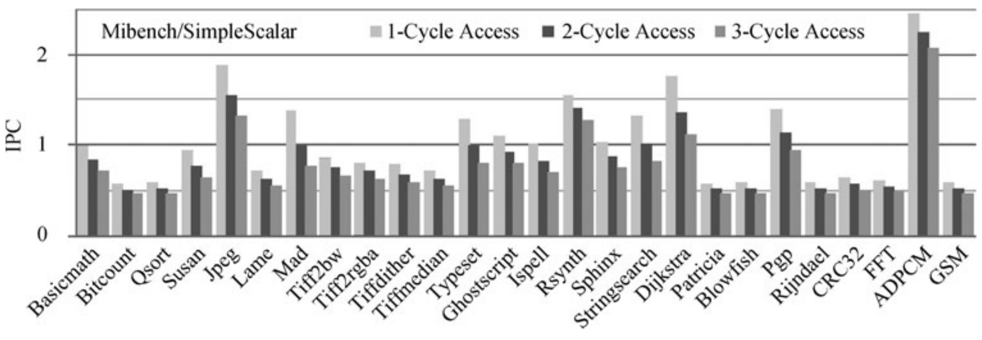


Fig. 1 Branch prediction delay vs. pipeline IPC

prediction rate but with different delays. With a delay of two cycles, pipeline efficiency (instructions per cycle, IPC) decreases by 15.7% on average. If the delay becomes three cycles, pipeline efficiency decreases by 26.8%. In a word, increased accuracy alone using complicated algorithms and large data structures in the predictor cannot overcome the penalties in delay that arise with this large predictor. Thus, in addition to accuracy, access delay should be treated as an important issue on pipeline efficiency.

As mentioned above, promoting accuracy is incompatible with decreasing delay. We explore the tradeoff between delay and accuracy, and finally propose a novel architecture for ahead branch prediction (A^{2}BP) to provide high accuracy without additional delay penalty. A^{2}BP separates traditional predictors into two parts: a small table located at the front-end of the pipeline, which makes the prediction brief enough even for some aggressive processors; operations on complicated algorithms and large data structures for accurate predictions are all moved to the back-end of the pipeline. At the same time, an effective mechanism of updating the small table in the front is introduced in A^{2}BP, which ensures an accurate prediction besides rapid access in the front-end of the pipeline. In addition, an indirect branch prediction algorithm based on branch history and target path (BHTP) is implemented in A^{2}BP, contributing to improved prediction accuracy.

Section 2 presents a review of the literature. Section 3 introduces the proposed architecture, with an instance predictor and its algorithms described in detail in the following Sections. Experiments are reported in Section 6. Finally, Section 7 concludes the paper.

2 Literature review

Jimenez and Seznec first noted the trend toward increasingly complex organization for predictors, which caused excessive access delay with high accuracy [3, 8]. Illustrating the considerable effect of delay on pipeline, Jimenez et al. explored three hierarchical schemes for accommodating delay: a caching approach, an overriding approach, and a cascading lookahead approach. All the schemes had a common theme: a small table for quick prediction and a large but slow table for accuracy. While both tables were located on the critical path of the pipeline and were constrained by one-cycle access, pipeline efficiency was degraded as the small table misses. In [5], Jimenez proposed a pipelined predictor that achieved a fast prediction by gradually reducing the number of targets in four consecutive pipeline stages ahead of the predicted branch arrival and finally selecting the target within a single cycle as the branch arrived at the front-end of the pipeline. The idea can be applied to any simple predictor that uses only global history and a few bits from the branch address. It does not use local histories, compute hashing functions, or create dependence between the branch address and the pre-fetched pattern history tables. Jimenez concluded that prediction delay was limited within one cycle; otherwise, a new architecture must be proposed to hide the branch prediction latency, which should be moved off the critical path of a pipeline.

Seznec et al. believe that hierarchical predictors waste a significant part of the fetch bandwidth whenever the slow but accurate prediction corrects or overrides the fast one. Furthermore, they are not scalable with technological developments [9] because tables that can be accessed in a single cycle become smaller and smaller as the time goes by [6]. At the same time, more complex indexing schemes and extra logic are required for the backup tables, which become larger and larger [10]. Therefore the accuracy of fast prediction will reduce with each new processor generation, while the response time of the state-of-the-art predictor will become comparatively longer and longer. This will generate more and more pipeline stalls in the front-end of pipeline, wasting more time than ever before. As an alternative solution to the hierarchy of predictors, Seznec et al. propose another pipelined predictor that divides prediction work into five cycles, and initiates five cycles ahead of the forthcoming branches. The illustrated example shows that even when the instruction address generation is partially initiated five cycles ahead of its use, it is possible to reach approximately the same prediction accuracy as the one of a conventional complex predictor. Unfortunately, pipelined predictors use certain global branch histories rather than local information, and most importantly, some complicated algorithms preferred for accurate prediction nowadays in research field cannot be pipelined effectively.

Santana et al. state that prediction overriding requires too much complexity to recover from wrong speculations, and pipelining the branch predictor is not as straightforward as pipelining other processor circuits [11]. In contrast, they focus on instruction streams and traces which have been academically proven to be more effective in branch prediction. Their experiments show that long prediction is possible to hide the prediction latency, keeping the execution engine of the processor busy while a new prediction was being generated. Actually, overlapping the execution of a prediction with the generation of the following prediction allows the tolerance of the access delay of this second prediction, removing the need for an overriding mechanism, and thus reducing the fetch engine complexity. We highly appreciate the stream and trace mechanism and even partly base our proposal on it.

Burcea et al. propose a virtualized branch target buffer (BTB) design [4, 7] to address the tradeoff between delay and accuracy, which is called Phantom-BTB. Ideally, on-chip BTB would be sufficiently large to capture the entire working set of the application and sufficiently small for fast access and practical on-chip dedicated storage. Depending on the application, these requirements are at odds. As such, Phantom-BTB augments a small and dedicated BTB on the chip with a large virtual table on the L2 cache for storing branch information to accommodate application demands for a larger BTB. With an elaborately designed pre-fetch engine, entries in the virtual table are proactively prefetched and installed in the dedicated BTB, thus increasing accuracy. Theoretically, predictors with Phantom-BTB can provide accurate prediction, but overcoming the high access latency of the virtual table in the L2 cache is a little difficult. Even though, our inspiration mainly comes from Burcea's idea of prefetching proactively, which results in a mechanism of ahead branch prediction and timely update to the small table in our proposal.

3 A^{2}BP

Predictors can be accessed within one cycle on the condition that the number of BTB entries is less than 1K [3], and complicated algorithms also cause additional delay [2]. To alleviate delay effectively, we propose a new architecture with a small table at the front-end of the pipeline for fast prediction. Operations on algorithms and data structures for accurate predictions are all moved off the critical path of the pipeline to the back-end. That is, the traditional large predictor in the front-end is replaced by a small table with several entries of branch-target pairs, and prediction in the front becomes simplified by matching between fetched branches and each entry in the small table. The target is immediately returned after a hit because the small table is very short.

Being moved to the back-end of pipeline, operations on algorithms and data structures are not constrained within one cycle, but are instead limited by the mean spaces between branches [2], which are on average five to six cycles. Thus, more complicated algorithms and larger data structures can be implemented, which lead to more accurate prediction. Moreover, occasionally prolonged operations over five to six cycles in the back-end are acceptable because the small table in the front-end of pipeline contains several branch-target pairs covering a series of incoming branches.

Ensuring that the small table at the front-end of the pipeline holds all the branches emerged in the fetch stage is a challenge for the proposed architecture. As a solution, hybrid algorithms are employed in the back-end for more accurate prediction. Branch history and target path are traced and stored in several data structures for continuous speculations and predictions. Meanwhile, BTBs are augmented in capacity to reduce aliasing. Each executed branch in the back-end triggers a process of cyclical speculations and predictions for the subsequent branches and targets. At the end of each process, a series of branch-target pairs is obtained. Then, the small table in the front-end is updated at the right time with these obtained branch-target pairs. Owing to continual and progressive updates, the small table replaces overdue entries each time with up-to-date branch-target pairs. Thus, the small table behaves like a window sliding forward with necessary predictions to cover all incoming branches in the front-end of the pipeline. Hence, the name "ahead branch prediction" is chosen.

4 Ahead branch predictor

Figure 2 illustrates an ahead branch predictor (ABP) based on the proposed architecture. The small table in the front-end of the pipeline is structured as a standard fully associative BTB with four entries of branch-target pairs. Physically, the small table is similar to a dual-port cache that can be accessed simultaneously by the fetch component through one port and the prediction engine (PE) through another port.

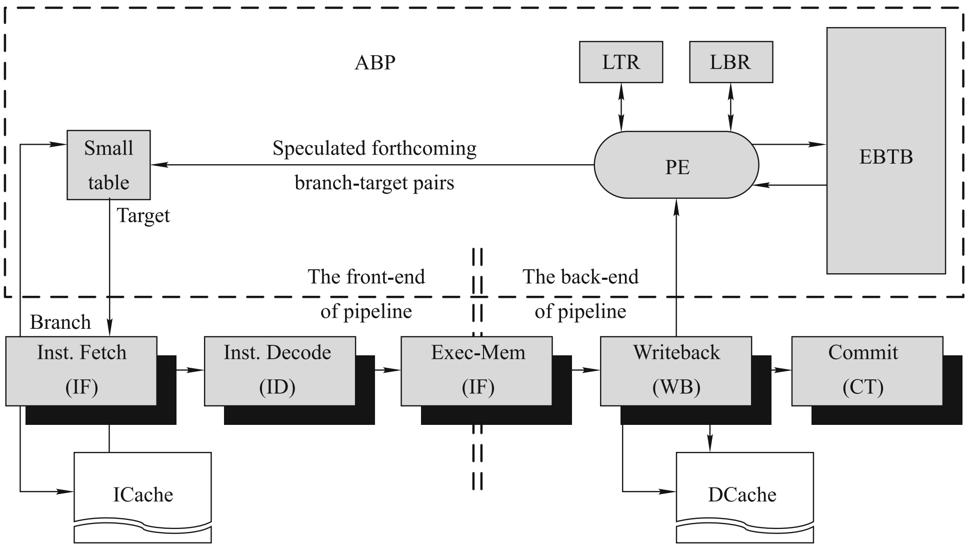


Fig. 2 Ahead branch predictor

In the back-end, complicated algorithms are embedded in PE, which plays a very important role in the entire prediction mechanism. PE classifies the application footprints and places them into several data structures, including last branch register (LBR), last target register (LTR), and so on. With stored last branch and target in registers, the PE links the committed branch up into a piece of instruction stream and separately stores each message into the extended BTB (EBTB) fields. Table 1 depicts the structure of EBTB. When an executed branch is committed, PE is triggered and uses EBTB to speculate the next branch, which is identified by an indexed branch and its target. Afterwards, PE predicts the target of the next branch by using embedded algorithms. With the predicted target, PE continues to speculate the sequential branch through EBTB. Circularly, four branch-target pairs are yielded. After the speculative predictions are completed, PE updates the small table with branch-target pairs obtained in the back-end. In other words, the back-end provides predictions to the front each time after a series of speculations and predictions in the back is completed. While the small table contains four succeeding branches after the executed one and their corresponding targets, it can provide predictions to four incoming branches that arrive at the front-end of pipeline sequentially, rather than the next one following the executed branch. Thanks to the continual update to the small table, again and again, after each executed branch, the small table is refreshed with partly overlapped but up-to-date branch-target pairs, and is kept forward ahead of the incoming branches. ABP is able to make accurate predictions quickly.

Table 1 Fields of extended branch target buffer

BrAddr	SeqtAddr	NextBrS	TargAddr	NextBrT

Two EBTBs exist with the same fields, and both are indexed by branches. EBTB0 is shared by conditional and unconditional branches with only one target each; it covers all direct branches and several indirect branches. EBTB1 is a dedicated buffer for indirect branches with multiple targets.

4.1 Case study of the early arrived branches

If the next branch arrives at the front-end of the pipeline early before the back-end provides a prediction for it, a miss occurs in the small table. In this case, the fetch engine will choose the fall-through address that follows the branch for the next instruction.

Figure 3 shows a scenario that occasionally occurs when many continuous branches individually arrive at the front-end of the pipeline. We assume that the back-end takes three cycles to fulfill each process of speculation and prediction, and to finally provide updates to the small table. We also assume that b_{i-1}, b_i, b_{i+1}, and b_{i+2} are covered by the small table that was updated before their arrival. Therefore, we focus on the prediction for branch b_{i+3}. As shown in Fig. 3, b_{i-1} arrives at the front-end of the pipeline at cycle T. It is committed at cycle $T + 3$ in the back-end, which triggers the process of speculations and predictions for b_i, b_{i+1}, b_{i+2}, and b_{i+3}. The small table is then ready for b_{i+3} after cycle $T + 6$. As b_{i+3} arrives at the front-end of the pipeline at cycle $T + 4$, no prediction has yet been prepared for it in the small table. In other words, the next branch b_{i+3} arrives early before the back-end provides a prediction for it.

While we cannot control the spaces between two branches in the program, we can deal with misses in the small table by increasing the branch-target pairs in the small table, or reducing the time used by the back-end to yield these pairs. These

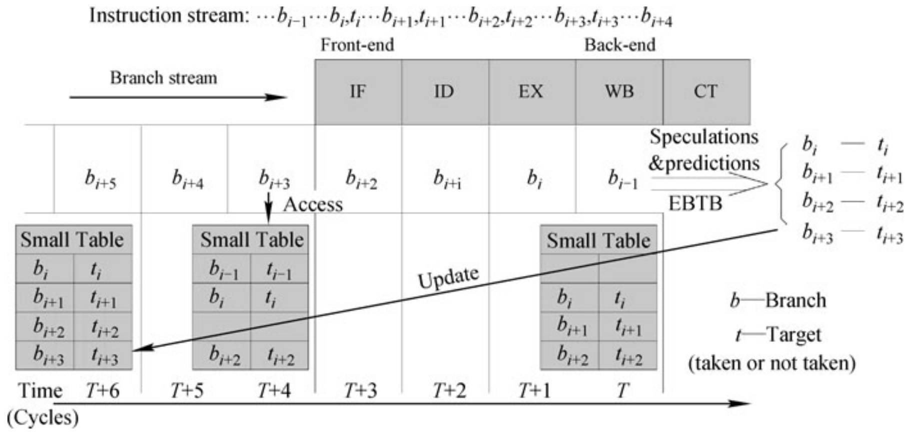

Instruction stream: ⋯b_{i-1}⋯b_i,t_i⋯b_{i+1},t_{i+1}⋯b_{i+2},t_{i+2}⋯b_{i+3},t_{i+3}⋯b_{i+4}
Front-end
Back-end
Branch stream
IF
ID
EX
WB
CT
Speculations &predictions
EBTB
Access
Update
Small Table
b—Branch
t—Target (taken or not taken)
Time (Cycles)
T+6
T+5
T+4
T+3
T+2
T+1
T

Fig. 3 Missed prediction for an early arrived branch

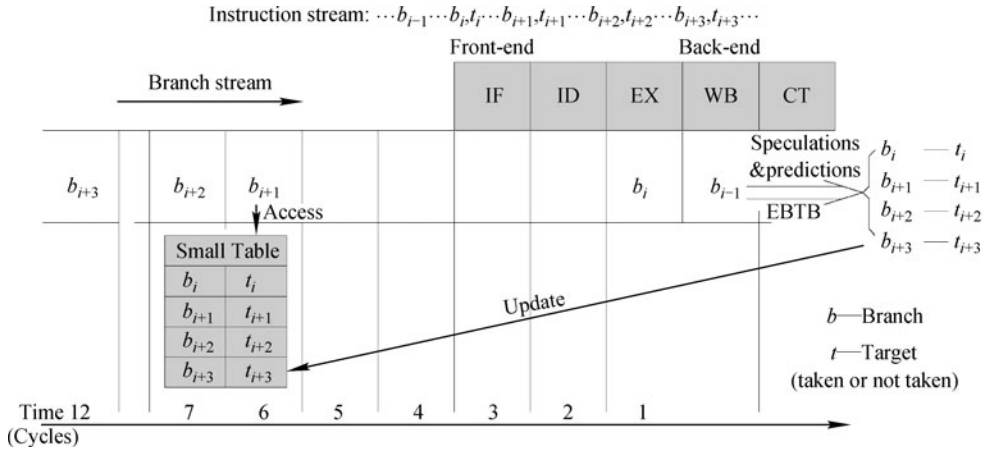

Instruction stream: ⋯b_{i-1}⋯b_i,t_i⋯b_{i+1},t_{i+1}⋯b_{i+2},t_{i+2}⋯b_{i+3},t_{i+3}⋯
Front-end
Back-end
Branch stream
IF
ID
EX
WB
CT
Speculations &predictions
EBTB
Access
Update
Small Table
b—Branch
t—Target (taken or not taken)
Time 12 (Cycles)
7
6
5
4
3
2
1

Fig. 4 Buffering against irregularly arrived branches

two methods are incompatible. Hence, we must explore the optimal policy to balance the length of branch-target pairs and the time to provide them in terms of effective implementations and pipeline efficiency. To simplify our design, a small table with four entries in the front-end is adopted to accommodate the branch-target pairs yielded by the back-end. Later after the experiments, we will discuss the probability of early arrived branches at the front, and the effect of the length of the small table on the prediction accuracy.

4.2 Case study of the irregularly arrived branches

Generally, the arrival time of branches at the front-end of the pipeline is irregular. In other words, the spaces between two continuous branches are not fixed. As shown in Fig. 4, branches b_i, b_{i+1}, b_{i+2}, and b_{i+3} arrive at the 1st, 6th, 7th, and 12th cycles, respectively. The irregularly arrived branch b_{i+2} squashes the spaces between b_{i+1} and b_{i+2} into null, making the proportional spacing between succeeding branches unavailable. Even though, the proposed architecture works well because the small table in the front-end buffers against incoming branches by means of its four branch-target pairs. In detail, we assume that branch b_{i-1} arrives before b_i. The executed b_{i-1} in the back-end triggers predictions for b_i, b_{i+1}, b_{i+2}, and b_{i+3}. After the 6th cycle, the small table is updated with b_i, b_{i+1}, b_{i+2}, and b_{i+3}. Therefore, branch b_{i+2}, which arrives irregularly at cycle 7th, will not affect the proposed predictor because the small table covers its target at that time. In a word, the predictor works well with branches that occasionally arrive irregularly.

5 Algorithm for ABP

Similar to all the pipelined predictors, ABP predicts target through speculation several cycles ahead before the relative branch arrives, leading to more misses than traditional predictors with the same algorithm. Reducing mispredictions require the usage of effective algorithms in ABP. It is well known the mispredictions for indirect branches occur as many times as 41% of the total misprediction despite the implementation of specialized hardware [12–14]. Hence, we propose a new algorithm for indirect branch prediction based on branch history and target path (BHTP). BHTP algorithm is simple but effective compared with other previously proposed algorithms.

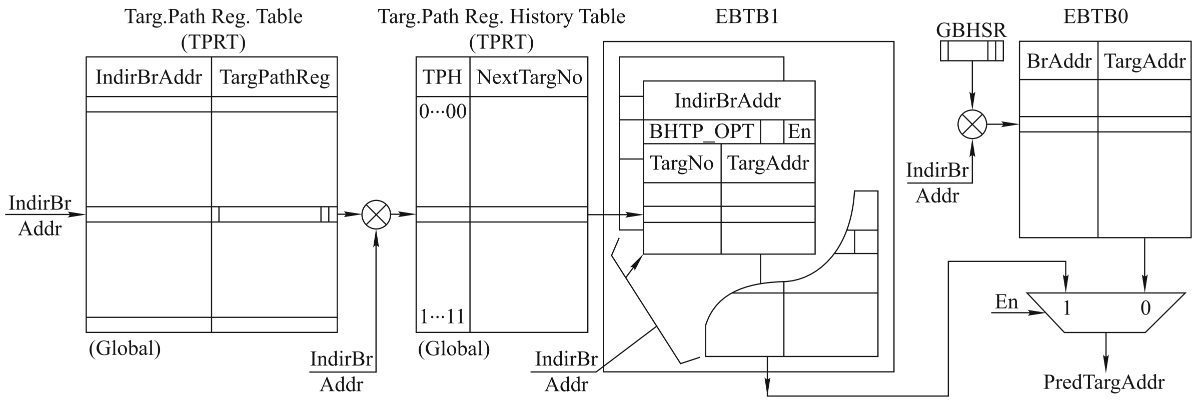


Fig. 5 The structures of indirect branch prediction base on BHTP

The main concept for BHTP is described as follows. Some targets for indirect branches are related to the global branch history (GBH), and they can be identified in EBTB0 through GBH shift register (GBHSR) exclusive-OR indirect branch addresses. Some targets are related to each other in sequence and can be predicted through a table with sequent target number in EBTB1. Otherwise, we predict the target from several candidates based on maximum likelihood. Figure 5 depicts the structures for BHTP algorithm. EBTB0 stores some indirect branches with targets associated with GBH. An indirect branch and its relative targets are moved to EBTB1 if it experiences several mispredictions in EBTB0. The prediction then is based on target path history, which is a stream formed one by one by the previously executed targets. While the target path is unstable, we predict the target on the maximum likelihood by holding one frequency counter for each target that has been frequently used in the recent past. The aging mechanism keeps these counters representative of the current phase of the application.

Pseudocodes for BHTP algorithm are listed below.

Algorithm 1 Branch prediction

```
if the branch is in EBTB1 then
    if target path prediction flag is enable then
        get target path history from TPRT;
        predict next target number by TPHT;
        predict target in EBTB1;
    else
        predict target by maximum likelihood
        method in EBTB1;
    end if
else
    if the branch is in EBTB0 then
        predict target in EBTB0;
    else
        return null;
    end if
end if
```

Algorithm 2 Prediction update

```
if the branch is in EBTB1 then
    if the target is not in EBTB1 then
        add the target to EBTB1;
    end if
    update TPHT with target;
    update TPRT with target;
    if update the branch too many times then
        set target path prediction flag disable;
    end if
else
    if the branch is in EBTB0 then
        update EBTB0 with target;
        if update the branch too many times then
            move the branch and its target to EBTB1;
            update TPHT and TPRT with target;
        end if
    else
        add the branch and its target to EBTB0;
    end if
end if
```

The branch prediction algorithm for ABP is finally a hybrid with at least two algorithms, including BHTP algorithm and some conventional algorithms.

6 Experiments and analyses

6.1 BHTP algorithm

The gem5 simulation environment [15] provides detailed simulation of a complete in-order CPU, which emphasizes instruction timing and simulation accuracy with a pipeline configured to model different numbers of stages and issue width [16, 17]. The branch prediction aspects of gem5 were modified to provide a predictor with BHTP algorithm (BHTP Predictor). A comparison of predictors was set up on the original gem5 (Original Predictor). Table 2 depicts the detailed configurations for the two predictors, which were simulated with in-order CPUs. Benchmarks in SPEC CPU 2006 suite were used because of the presence of several large object-oriented C++ programs with many indirect branches.

Table 2 Configurations for two compared predictors

BHTP predictor	Original predictor
Tournament+BHTP algorithms	Tournament algorithm
Choice predictor: 8K	Choice predictor:8K
Global predictor: 8K	Global predictor:8K
Local predictor: 2K	Local predictor:2K
Local history table: 2K	Local history Table:2K
TPRT:1K-entry,TPHT:4K-entry	
EBTB0:1K-entry,4-way set-assoc	BTB: 1K-entry
EBTB1: 64-entry, 8-way.	4-way set-assoc.
LRU repl. 16-entry RAS.	LRU repl. 16-entry RAS.
Mispred. penalty:3 cycles	Mispred.penalty:3 cycles

After being cross-compiled statically with O2 parameters, benchmarks were run separately on two predictors in the system call emulation mode on the condition that no kernel codes or other application processes could interfere in each tested benchmark. For the purpose of the study, each simulation ran the benchmark until its termination, or for 100 million instructions, whichever came first. Table 3 displays statistics on several typical benchmarks with a relatively large number of indirect branches; others with fewer indirect branches are omitted.

Experimental results show that the predictor with BHTP algorithm performs better with an average of 28.98% reduction of misprediction over the compared predictor with original tournament algorithm only. Further study on source code and profiles shows that benchmarks (445.gobmk, 450.soplex) with limited targets for each indirect branch benefit a lot from BHTP algorithm, albeit to different extents. But BHTP algorithm is a little ineffective while target path exhibits insufficient temporal and spatial locality, such as on 454.calculix, 459.GemsFDTD.

6.2 Ahead branch predictor

SimpleScalar is a well-known tool set for architectural simulation [18]. We prefer using the branch prediction simulator (Sim-bpred) in SimpleScalar because it clearly frames a front-end and a back-end like a pipeline with five stages. Sim-bpred is similar to a single-issue and in-order simulator, which allows focusing solely on the feasibility of the proposed architecture.

The source code of Sim-bpred and Bpred modules were modified to realize a system on A^{2}BP (ABP system), which was based on a hybrid algorithm combined with BHTP and Comb algorithm in SimpleScalar. Table 4 lists the configurations for the predictor in ABP system and compares it with a relative predictor based on original SimpleScalar (B-BTB System).

The predictor in the compared system consists of a bimodal algorithm and a BTB with 1K entries, so that one cycle access is viable. The small table in ABP system contains four entries for the same purpose. Both of the predictors can be accessed within a single cycle. Hence, we investigate prediction accuracy instead of instruction throughput (IPC) as performance indications.

Table 3 Mispredictions on two predictors

Benchmarks SPEC CPU 2006	Data type language	Indirect branches	Tournament mispredictions	Tournament+BHTP mispredictions	Improvement/%
403.gcc	Integer C	2 390	3 992	2 920	26.85
445.gobmk	Integer C	533	316	270	14.56
456.hmmer	Integer C	205	8	3	62.50
464.h264ref	Integer C	245	341	335	1.76
471.omnetpp	Integer C++	998	27 852	26 876	3.50
483.xalancbmk	Integer C++	2 805	26 201	24 772	5.45
433.milc	Floating C	189	41	35	16.67
454.calculix	Floating C	447	2 589	2 582	0.27
482.sphinx3	Floating C	184	91	79	13.19
435.gromacs	Floating C/Fortran	189	9	0	100.00
459.GemsFDTD	Floating Fortran	277	3 249	3 230	0.58
444.namd	Floating C++	120	4	3	25.00
447.dealII	Floating C++	885	85	46	45.88
450.soplex	Floating C++	638	180	137	23.89
453.povray	Floating C++	464	56	3	94.64

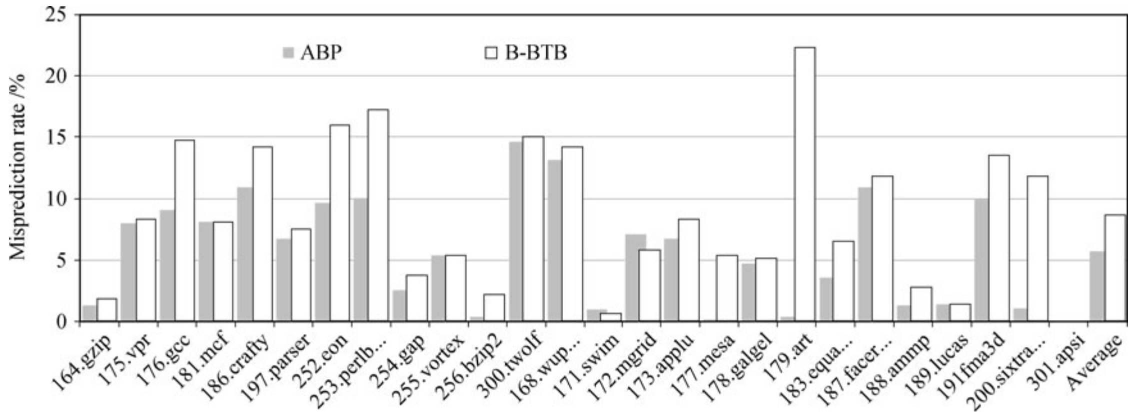


Fig. 6　SPEC CPU 2000 on ABP and B-BTB systems

Table 4　Configurations for predictors in ABP and B-BTB systems

ABP system	B-BTB system
Comb+BHTP algorithms	Bimodal algorithm
Comb:1024,Bimodal:1024	Bimodal:1024
2-Level:1024,TPRT:1024	BTB:1K-entry
TPHT:1024	4-way assoc.
EBTB0:1K-entry,4-way assoc	LRU repl.
EBTB1:64-entry,8-way assoc	16-entry RAS.
LRU repl. 16-entry RAS.	Mispred.penalty:3 cycles
Mispred.penalty:3 cycles	

SPEC CPU 2006 suite has some functional calls that were not supported by SimpleScalar. As such, SPEC CPU 2000 was used in the experiments. All benchmarks were run until completion or execution of 200 million instructions, whichever came first. Figure 6 shows the misprediction rates of the two systems in comparison with each other. Experimental results demonstrate that ABP system achieves a prediction accuracy of 94.27% on average, which is 2.92% higher than that of the B-BTB system.

Results of two benchmarks on ABP system were lower than those on the compared system. Analysis of the code showed that the total number of instructions in 171.swim was insufficient to warm up the ABP system. We repeated benchmarking 171.swim several times in the simulation. The prediction accuracy gradually increased on the ABP system, finally exceeding what was reached on the compared system. As for 172.mgrid, the ABP system was not superior to the compared system on the condition of its few indirect branches.

Figure 7 shows the branch prediction accuracies achieved on two compared systems with different instruction number executed each time. For instance, we separately ran 175.vpr for 100 million instructions on two systems. The prediction accuracy on ABP system was 92.22%, 1.57% higher than that on B-BTB system. ABP system outperforms B-BTB system all the time, but they have almost the same characteristic, which proves ABP system as feasible as B-BTB system.

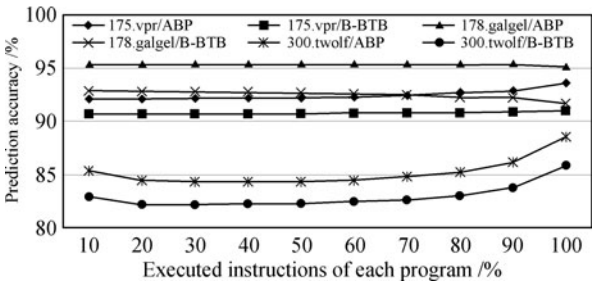


Fig. 7　Branch prediction accuracy vs. executed instructions

Extensive testing is necessary. As such, nine benchmarks from MiBench were selected [19]. Most of the benchmarks are control intensive programs with a lot of branches. All the benchmarks were cross-compiled for SimpleScalar/ALPHA with O2 optimizations. Each benchmark was run for 100 million instructions with the large data set input. Figure 8 shows the mispredictions of these benchmarks run on two compared systems. It can be seen that the average prediction accuracy for ABP system is 94.73%, while B-BTB system achieves at most 92.64% average prediction accuracy.

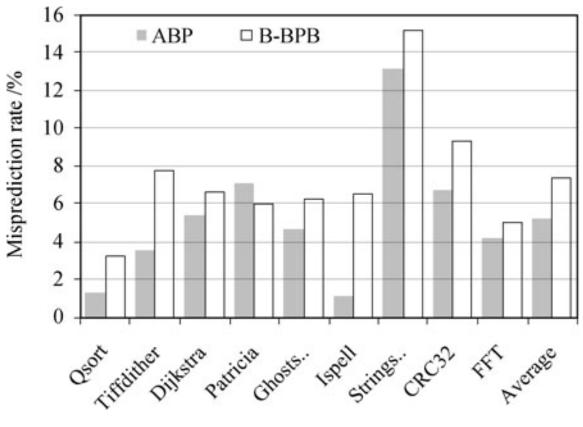


Fig. 8　MiBench on ABP and B-BTB systems

6.3 Analysis on the length of small table

Being a critical component in the front-end of the pipeline, the small table should cover all the incoming branches. In contrast, Fig. 3 illustrates a missed prediction for an early arrived branch. The probability of missed prediction for some incoming branches in the small table can be theoretically derived from the assumption that one fifth of the executed instructions are branches, and four succeeding branches following a previous branch closely arrive at the front-end of pipeline within next five cycles. As such, the probability is estimated to be 0.13%. Comparatively, Table 5 shows the probability of two interval branches b_{i-1} and b_{i+3} arriving at the front-end of pipeline within six cycles in our simulation. Missed prediction for a small table with four entries is empirically and theoretically no more than 0.34%, which is negligible compared to the prediction accuracy over 90%. Therefore, a small table with four entries is enough for a typical 5-stage pipeline. As for a deeper pipeline, the small table should be extended to accomodate more branch-target entries. Correspondingly, branch-target pairs produced by the back-end of the pipeline should be increased.

Table 5 Probability of 2 interval branches b_{i-1} and b_{i+3} arriving w/i six cycles

Benchmarks SPEC CPU 2000	Occurrences	Total branches	Probability/%
164.gzip	1	48 931 220	0
175.vpr	1	53 412 671	0
176.gcc	26 823	75 728 623	0.035
181.mcf	148 289	43 845 323	0.34
186.craft	2 436	58 093 360	0.004 2
197.parser	10	99 299 551	0
187.facerec	124	251 226 605	0
188.ammp	1	90 202 773	0
189.lucas	29	8 230 270	0
191fma3d	174	492 895	0.035
200.six track	66 966	74 353 562	0.09
301.apsi	746	21 340 474	0

6.4 Trade-off analysis between delay, area, and power consumption

The prediction access time in ABP is negligible because there are only four entries in the small table. But prediction delay in the back-end should be controlled effectively because it affects the pipeline recovery from mispredictions. This is the reason why we configured limited EBTB in the experiments. Even then the storage budget in the back-end of ABP is estimated over 13 KB, which implies a large body as well as high power consumption over compared B-BTB system with 11 KB only.

6.5 Improvements

Previous research has shown that some applications have more indirect branches than others, and some dynamic indirect branches in object-oriented programs have more targets than eight [20]. As such, we increased EBTB1 from 64 entries to 128 entries, each with 16 targets instead of 8. The total budget of ABP was increased by 23%. At the same time, the prediction accuracy rose from 94.27% to 96.43% on average for SPEC CPU 2000 benchmarks, and from 94.73% to 96.8% on average for MiBench. Increasing EBTB1 continuously helped little for SPEC CPU 2000, none for MiBench.

7 Conclusions

In this paper, a novel architecture for ahead branch prediction is proposed and evaluated. A^{2}BP is divided into two parts: 1) a small table in the front-end of the pipeline to provide quick prediction; and 2) complicated algorithms and large data structures in the back-end to provide high prediction accuracy. Thus, the issue of prediction accuracy versus access delay is completely solved. Most importantly, A^{2}BP is scalable with future technologies because it can take advantage of growing chip capacities for higher prediction accuracy. At the same time, it can keep a small table for one-cycle access as clock rates increase aggressively. In conclusion, A^{2}BP is a promising architecture for future branch prediction.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 60973010). The authors would like to thank the anonymous reviewers for their constructive feedback. Wenbing Jin would like to thank Yanhong Wang for engaging discussions at different stages of this work.

References

1. Seznec A. The L-TAGE branch predictor. Journal of Instruction-Level Parallelism, 2007, http://www.jilp.org/vol9/v9paper6.pdf
2. Srinivasam R, Frachtenberg E, Lubeck O. An idealistic Neuro-PPM branch predictor. Journal of Instruction-Level Parallelism, 2007, http://www.jilp.org/vol9/v9paper8.pdf
3. Jimenez D A, Keckler S W, Lin C. The impact of delay on the design of branch predictors. In: Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'00). 2000, 67–76
4. Burcea I, Moshovos A. Phantom-BTB: a virtualized branch target buffer design. In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09). 2009, 313–324
5. Jimenez D A. Reconsidering complex branch predictors. In: Proceedings of the 9th International Symposium on High-Performance Com-

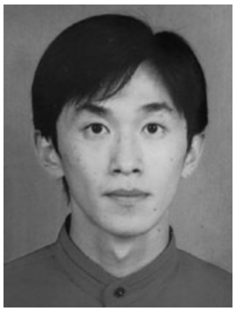
puter Architecture (HPCA'03). 2003, 43–52

6. Agarwal V, Hrishikesh M, Keckler S W. Clock rate versus IPC: the end of the road for conventional microarchitecture. In: Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00). 2000, 248–259
7. Burcea I, Somogyi S, Moshovos A. Predictor virtualization. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09). 2008, 157–167
8. Seznec A, Michaud P. A case for (partially)-tagged geometric history length predictors. Journal of Instruction-Level Parallelism, http://www.jilp.org/vol8/v8paper1.pdf
9. Seznec A, Fraboulet A. Effective ahead pipelining of instruction block address generation. In: Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03). 2003, 241–252
10. Seznec A, Felix S, Krishnan V. Design tradeoffs for the alpha EV8 conditional branch predictor. In: Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02). 2002, 295–306
11. Santana O J, Ramirez A, Valero M. Latency tolerant branch predictors. In: Proceedings of Innovative Architecture for Future Generation High-performance Processors and Systems. 2003, 30–39
12. Joao J A, Mutlu O, Kim H. Improving the performance of object-oriented languages with dynamic predication of indirect jumps. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08). 2008, 80–90
13. Li T, Bhargava R, John L K. Adapting branch-target buffer to improve the target predictability of java code. ACM Transactions on Architecture and Code Optimization, 2005, 2(2): 109–130
14. Joao J A, Mutlu O, Kim H. Dynamic prediction of indirect jumps. IEEE Computer Architecture Letters, 2007, 6(2): 25–28
15. Binkert N, Beckmann B, Black G. The gem5 simulator. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1–7
16. Nathan B L, Ronald D G, Lisa H R. The M5 simulator: modeling networked Systems. IEEE Micro Magazine, 2006, 26(4): 52–60
17. Milo M K, Daniel S J, Bradford B M. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. ACM SIGARCH Computer Architecture News, 2005, 33(4): 92–99
18. Austin T, Larson E, Ernst D. SimpleScalar: an infrastructure for computer system modeling. IEEE Micro Magazine, 2002, 35(2): 59–67
19. Guthaus M R, Ringenberg J S, Ernst D. MiBench: a free, commercially representative embedded benchmark suite. In: Proceedings of the 2001 IEEE International Workshop on Workload Characterization. 2001, 3–14
20. Kim H, Joao J A, Mutlu O. VPC prediction: reducing the cost of indirect branches via hardware-based dynamic devirtualization. In: Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07). 2007, 424–435

Wenbing Jin received his BS from Beijing Institute of Technology in 1990, and his MS from Taiyuan University of Technology in 2006. He is a senior engineer with North Automatic Control Technology Institute, and is currently a PhD candidate of Beijing Institute of Technology. His research interests include computer architecture, parallel computing, and artificial intelligence. He is a member of ACM.

Feng Shi received his BE in Physics in 1983 from Peking University and received his PhD from Beijing Institute of Technology, in 1999. He is currently a professor with the School of Computer Science and Technology, Beijing Institute of Technology. His research focuses on parallel computing and computer architecture.

Qiugui Song received his BS from North University of China in 2002. He is a senior engineer with North Automatic Control Technology Institute. His research interests include computer architecture and embedded systems.



Yang Zhang is a lecturer with Hebei University of Science and Technology, and currently a PhD candidate of Beijing Institute of Technology. His research interests include computer architecture and high-performance computing.