Electronic Preprint for Journal of Information Processing Vol.24 No.3

Regular Paper

Energy Reduction of BTB by Focusing on Number of Branches per Cache Line

Ryotaro Kobayashi[1,a)] Kaoru Saito[2] Hajime Shimada[3]

Received: June 30, 2015, Accepted: January 12, 2016

Abstract: The latest processors exploit Instruction Level Parallelism to improve performance, but this strategy is limited by control dependency. To alleviate this problem, the most recent processors utilize branch prediction. A typical branch predictor applies prediction to all instructions; however, this means that the branch predictor requires a high energy input, especially to the BTB (branch target buffer). In this paper, we propose a method that reduces the number of BTB accesses and abolishes the BTB tag by associating the instruction cache line and BTB entry. This proposal allocates a fixed number of BTB entries to a cache line and allocates an index to the corresponding instruction in the cache line as a substitute for the BTB tag. Due to the small fixed numbers of BTB entries compared to the fetch amount and reduction of the BTB tag, our proposal can reduce BTB access energy requirements. Our proposal is anticipated to cut energy consumption, but it cannot apply a branch target prediction to the entire set of instructions if there are too many branch instructions per cache line. We therefore evaluated its effects on processor performance and energy consumption. The evaluation results show that the proposal reduces BTB access energy requirements to 47.5% without any performance loss.

Keywords: branch target buffer, branch prediction, energy reduction

1. Introduction

The most recently-developed processors show improved performance by exploiting instruction-level parallelism (ILP). However, there are many obstacles to ILP, of which control dependency is one of the best known. When the processor executes a branch instruction, the next fetch instruction is decided by choosing the Program Counter (PC) value from the branch target address or a subsequent instruction address in the program order, based on whether the branch condition has been satisfied or not. The branch condition is defined at the execution stage in the latter part of the pipeline, which means that the processor cannot define the next fetch instruction until the branch instruction reaches the execution stage. Thus, the processor has to stall instruction fetch, which causes what is termed a pipeline bubble. This is why control dependency occurs.

Generally speaking, typical programs contain around 20% branch instructions [1], which means that control dependency significantly affects processor performance. Current processors are apt to increase pipeline depth, which aggravates performance degradation due to control dependency. So, to alleviate performance degradation caused by control dependency, current processors exploit speculative execution based on branch prediction.

Branch prediction is a mechanism in which the next fetch instruction is predicted based on past results of branch instruction. By fetching instructions based on the prediction result, branch prediction alleviates performance degradation caused by control dependency. The branch predictor is separated into a branch direction predictor and a branch target predictor. The branch direction predictor predicts whether the branch instruction is Taken or Not Taken, based on past Taken or Not Taken histories. The branch target predictor predicts the branch target address based on the branch target address that was created in the previous execution. A branch target buffer or BTB is a widely-used branch target predictor scheme which is used in many types of processors, from embedded types to high-end models.

A branch predictor effectively alleviates control dependency; however, if an error in branch prediction has occurred, the processor needs to invalidate all the fetched instructions that were based on the incorrectly predicted result and then fetch instructions from the correct branch result. The latest processors, which have deep pipelines, suffer a serious penalty from errors in branch prediction, because the processor has to invalidate all the instructions from the fetch stage to the execution stage. The number of instructions to be invalidated thus increases in proportion to the pipeline depth.

High energy consumption is one of the major drawbacks to branch prediction. It is calculated by multiplying the energy consumed per single access by the number of accesses, and increases in proportion to the complexity of the mechanism. The branch predictor needs to store multiple branch target addresses, a comparatively large volume of data, which increases the energy needed per single access. Furthermore, the processor has to access the branch predictor for all instructions, because the processor itself cannot determine branch instructions. To achieve

[1] Graduate School of Engineering, Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[2] Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[3] Nagoya University, Nagoya, Aichi 464–8601, Japan
[a)] ryotaro.kobayashi@ppl.cs.tut.ac.jp

speculative instruction fetch, the most commonly used method is for the processor to access the branch predictor for all instructions. If the processor can obtain a branch prediction result from the branch predator, the processor determines that the instruction is a branch instruction. This increases the number of accesses to the branch predictor.

In this paper, to reduce the total access count to the BTB and to reduce the energy needed per single access to the BTB, we propose a mechanism which allocates BTB entries to a cache line. We focused our attention on the average number of branch instructions per single cache line and allocated a moderate number of BTB entries to each cache line. We found that by allowing the instructions in each cache line to consist of 25% BTB entries, we could achieve better energy efficiency overall, even taking into account the energy overhead due to performance loss. Allocating BTB entries to a cache line also makes it possible to share the tag array between the BTB and cache line, thus reducing the energy needed for tag array access and tag comparison. By reducing total access count and sharing tags with the cache line, the processor can reduce the power consumption of the BTB. We achieved 43.6% BTB access energy reduction without performance degradation.

It has been reported that the branch predictor accounts for around 10% of the processor's overall energy consumption [2] and that the target predictor accounts for around 87.5% of the branch predictor's overall energy consumption [3]. The BTB therefore appears to consume approximately 8.75% of the total processor energy. However, this is just a snapshot value since, the rate varies from processor to processor; but the key point, that the BTB accounts for a large proportion of the processor's energy consumption, remains true for all processors.

The latter part of this paper is organized as follows. Section 2 explains branch prediction mechanism. Section 3 mentions the related studies. Section 4 describes our strategy. Section 5 presents our proposed mechanism. In Section 6, we perform an evaluation of our mechanism. Section 7 sets out our conclusions.

2. Branch Prediction Mechanism

2.1 Branch Predictor

A branch predictor predicts both a branch direction (Taken or Not Taken) and a branch target address based on foregone execution result of the branch instruction. By fetching and executing instructions with prediction result, the processor can improve performance due to alleviation of control dependency.

The branch predictor contains two types of predictors: a branch direction predictor and a branch target predictor. The branch direction predictor predicts whether a branch is Taken or Not Taken. On the other hand, the branch target predictor predicts a branch target address. If the instruction is predicted as Taken and a branch target address is predicted, the next PC is predicted as the predicted branch target address. Otherwise, the next PC is predicted as the next instruction address in sequential order.

At the fetch stage, the processor cannot determine whether an instruction is a branch or a nonbranch because it has defined after decoding. However, if the BTB is accessed after decode stage, the processor have to insert pipeline bubble after Taken branch because the branch target address is obtained after decoding. Hence, in order to achieve continuous instruction fetch without the pipeline bubbles, we assume that the processor performs both the branch prediction based on PC and instruction fetch simultaneously, regardless of whether the instruction is a branch instruction or not.

2.2 Branch Direction Predictor

There are some well-known branch direction predictors such as gshare [4] and perceptron predictor [5] and there are many variations related to them. Those predictor use a history related to the branch directions that have been obtained in past branch results. The notable characteristic of gshare is that it utilizes XOR of global history and PC value for index of pattern history table. The perceptron prediction also utilizes global history but it calculates Taken probability with multiplying weight to each history similar to perceptron model. In this paper, we use gshare as a direction predictor. We only touch energy consumption of the branch target predictor so that the energy consumption reduction of the branch direction predictor becomes a future work.

2.3 Branch Target Predictor

We introduce detail of the branch target prediction. **Figure 1** shows the organization of the Branch Target Buffer (BTB).

Each BTB entry stores tag and branch target address (TPC: Target PC). The index of BTB obtained from instruction address by curving out middle "$\log_2$(*the number of BTB sets*)" bits. The lower bits are 2-bit width word offset and the higher bits becomes tag. An access to the BTB is separated into the reference access and the update access.

The reference access is done in fetch stage and the BTB is accessed with PC for fetching. If one of the tags stored in corresponding entries matches to the tag comes from PC, the stored TPC becomes a predicted address when the branch direction predictor gives Taken result. Otherwise, a predicted address becomes the next instruction address in sequential order.

The update access is done in commit stage. When a branch instruction reaches commit stage, if the branch instruction is a Taken branch, BTB is accessed with the instruction address of the branch instruction. By comparison result of the tag stored in corresponding entries and tag comes from the instruction ad-

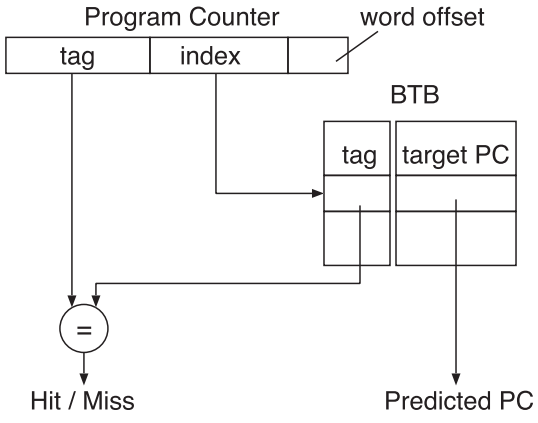

Fig. 1 Branch target buffer.

dress, the processor updates the BTB or updates LRU (Least Recently Used) information. If the branch instruction is a Not Taken branch, the processor does not update the BTB. So, the number of reference access is much larger than that of the update access. In latter part, we utilize "reference" notation for the reference access and "update" for the update access.

With above operation, the BTB works as a branch instruction detector in fetch stage. If the processor can obtain TPC from the BTB, an instruction corresponding to the PC is a branch instruction. Otherwise, the instruction is not branch instruction and the predicted address becomes the next instruction address in sequential order.

2.4 Energy Consumption of BTB

In this paper, we focus on the energy reduction of the BTB. The energy reduction of direction predictors is investigated by future work.

The energy consumption of the BTB is large. The reason for this is as follows. The overall energy of the BTB is the product of BTB energy per access and the number of BTB accesses. The BTB energy per access depends on the BTB capacity increases, where the capacity is the product of the entry length and the number of entries. The entry length is long since each entry holds a tag and a branch target address. The number of entries is large because the BTB needs to store many branch target addresses to accomplish high prediction hit rate. Hence, the BTB capacity is large, leading to high BTB energy per access. On the other hand, the number of BTB accesses also becomes large because, as already mentioned, we assume that every fetched instruction accesses the BTB to effectively fetch instructions.

In this paper, in order to reduce energy consumption, we remove the tag field from the BTB entry to reduce the BTB energy per access, and combine simultaneous accesses into one access to reduce both the number of BTB ports and the number of the BTB accesses.

3. Related Studies

There are several studies that aim to reduce the hardware cost of the BTB. They include a mechanism that reduces the length of the tag field [6] and mechanisms that reduce the length of the TPC field [3], [7], [8]. Hardware cost reduction leads to energy reduction. As described before, the overall energy consumption of the BTB is determined by the product of energy per access and number of accesses. If the bit length of the tag or the TPC field is reduced, energy per access is reduced. As a result, overall energy consumption by the BTB is also reduced.

Parikh et al. [9] utilize an I-cache to cut the number of BTB accesses. The proposed mechanism, called a prediction proof detector (PPD) is a dedicated table. A flag held by the table corresponds one-to-one to an instruction in the I-cache. The flag identifies whether a corresponding instruction is a branch or not. First of all, at the fetch stage, the flag corresponding to a fetched instruction is acquired by accessing the PPD. If the flag indicates that the fetched instruction is a branch, the BTB is accessed; otherwise, it is not. As a result, some parts of the accesses are eliminated, but the PPD access delays the start time of the BTB access or compromises the energy reduction effect, depending on the configuration.

Petrov et al. [10] proposed a technique that reduces the number of accesses to the BTB. The technique consists of a dedicated BTB and dedicated software. The dedicated BTB is called an application customizable branch target buffer (ACBTB). The software generates control-flow information related to the loops at compilation time. This information is held by the ACBTB at run time. The ACBTB is only accessed by branches at the execution stage. The accessed information is used to identify the branches in the fetch stage beforehand. This allows the number of accesses to the ACBTB to be reduced. However, the ACBTB is customized exclusively on single-issue and in-order processors. Unlike hardware-based mechanisms, the ACBTB cannot predict the branch target without control-flow information, since it does not hold anything other than control-flow information.

Chung et al. [11] assumed that the next instruction is usually the next sequential instruction and proposed a mechanism based on this for reducing the number of accesses to the BTB. The mechanism accesses gshare one cycle in advance to predict the branch direction. If the predicted direction is Taken, the BTB is accessed. Otherwise, the BTB is not accessed. The mechanism works well if the fetched instruction is a branch, because the gshare is designed for branch direction prediction. However, if the fetched instruction is a nonbranch, the mechanism may or may not work well because gshare predicts a branch direction in spite of its being a nonbranch.

Kahn et al. [2] proposed a method that adopts a Bloom filter and a small BTB to reduce the number of accesses to the BTB and thus the BTB energy per access. Before the fetched instruction accesses to the BTB, the Bloom filter can indicate whether the BTB hits or misses. The Bloom filter used in this method is equivalent to the partitioned-address Bloom filter proposed by Peir et al. [12]. The small BTB is the same as the BTB except for its smaller size. The small BTB and the Bloom filter are accessed by the fetch instruction at the same time. If the small BTB access misses and the Bloom filter hits, the fetched instruction accesses the BTB. Otherwise, it does not access the BTB. However, the Bloom filter used in this method does not work well due to the problem reported in Ref. [12].

Sadeghi et al. [3] proposed reducing the number of accesses to the BTB and the BTB energy per access by adding a small BTB and modifying both the BTB and the small BTB. The modified BTB provides the distance from the instruction address to the branch target address on behalf of the branch target address. The provided distance is then added to the address of the fetched instruction, resulting in the predicted branch target address. Since the bit length of the distance is smaller than the branch target address, the BTB energy requirement per access decreases. Moreover, both the BTB and the small BTB provide the distance from the branch target address to the next branch instruction address. The provided distance is added to the predicted branch target address to be the next branch address. When the PC changes between the branch target address and the next branch address, neither the BTB nor the small BTB is accessed. However, the next branch address calculation performs two additions in series, in-

creasing the time needed for predicting the branch target address.

4. Our Strategy

The principle of our proposed mechanism is to reference a branch target buffer (BTB) for the cache line, which is conventionally performed for each instruction, and to predict branch target addresses for multiple branch instructions present in the cache line by referencing the BTB only once. In the conventional mechanism, in which the BTB is referenced for each instruction, the BTB is referenced as many times as is needed to fetch instructions. This requires a large number of references to the BTB. However, the number of reference to the BTB can be reduced if multiple branch target addresses required per cycle can be predicted by referencing the BTB only once, in the same way as the most recent processors obtain multiple instructions by referencing the instruction cache only once.

All of the instructions fetched per cycle are stored in the cache lines present in the instruction cache set to be referenced. Using this characteristic, branch target addresses for multiple instructions fetched per cycle are predicted by referencing the BTB once. First, a BTB entry that can hold multiple target PCs (TPCs) is prepared, and a one-to-one correspondence is established between the instruction cache set and the BTB entry. When fetching an instruction, the BTB entry corresponding to the instruction cache set containing the instruction is always referenced, and multiple branch target addresses present in the BTB entry are read by accessing the entry once. This makes it possible to predict branch target addresses for multiple branch instructions fetched per cycle by accessing the entry only once.

5. Proposed Mechanism

5.1 Configuration

Figure 2 shows the configuration of our proposed mechanism.

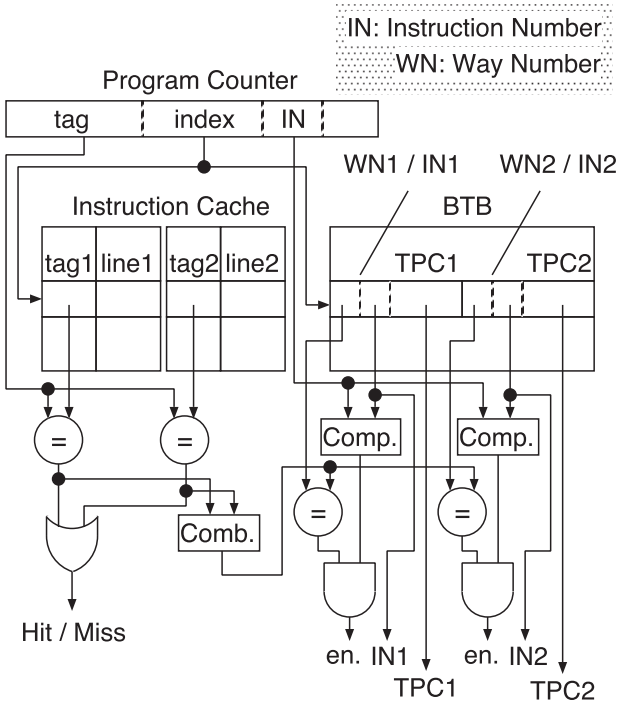

Fig. 2 Proposed mechanism.

In the figure, which shows a simplified explanation, the instruction cache is a two-way cache and the fetch width is 2. The specification of Comp. (Compare Circuit) and Comb. (Combinational Circuit) in the figure is described later in this paper.

The number of BTB sets is made equal to the number of instruction cache sets to correlate them. The index used to access the instruction cache is employed to access the BTB. With this configuration, the instruction cache tag is the same as the BTB tag. Therefore, the BTB tag is eliminated, and the decision of whether the instruction cache tag is a correct or erroneous is used to decide whether the BTB tag is correct or erroneous.

In our proposed mechanism, multiple branch target addresses (in the figure, TPC refers to a target PC) are stored in one BTB entry in the same way as multiple instructions are stored in the cache line. To determine TPC information, the instruction number (IN in the figure) to indicate the ordinal position of the instruction corresponding to the TPC in the cache line and the cache way number (WN in the figure) that indicates the way where the corresponding instructions are stored in the instruction cache are held for each TPC. The instruction number is a "$\log_2$(*the number of instructions in the cache line*)" bits and the way number is a "$\log_2$(*the degree of associations in the cache*)" bits. Therefore, the tag is smaller in size than that held in a conventional BTB. This makes it possible to reduce the energy consumption per access. It is not shown in the figure, but a valid bit is held for each TPC that indicates whether the TPC is valid or not.

In the proposed mechanism, if too many branch instructions are present in the cache line, a sufficient number of TPCs cannot be stored in one BTB entry. As a result, branch target addresses cannot be predicted, and the performance of the processor declines. The percentage of this reduction of an IPC is evaluated in the evaluation section.

5.2 Referencing Operation

This section describes the operation for referencing the BTB in the proposed mechanism. The BTB is referenced concurrently when the instruction cache is referenced in the fetch stage. As previously described, the index used to reference the BTB is created using the same bits as for the instruction cache. All TPCs present in the referenced set are read and controlled by comparing the instruction number, the way number of the TPCs, and the instruction cache tags. Each TPC is checked to ensure that it is a TPC for the instruction fetched in the current cycle. This process is performed by comparing the bit row corresponding to the instruction number and each instruction number in the fetch width in the PC with the instruction numbers in the BTB. This is a process in Comp. shown in the figure. Comp. is a combinational circuit that checks each instruction number in the fetch width, starting from the first instruction number of the fetch. Subsequently, it is determined whether the way number is the same as the number of the way hits in the instruction cache in the current cycle. This is a process in the combinational circuit in Comb.; the subsequent comparator circuit shown in the figure. Comb. is a combinational circuit that outputs the number of way hits using the results of a comparison of the tags in the instruction cache.

The selection circuit is controlled by the above results, and the TPC with the predicted address is determined. Only if all of the above is determined to be true is the TPC set to the predicted address. The predicted address along with the instruction number is sent to the instruction fetch.

The tag is shared with the instruction cache even when the TPC is set to the predicted address. Therefore, if a tag error occurs in the instruction cache, the TPC is invalid.

5.3 Update Operation

This section describes the operation for updating the BTB in the proposed mechanism. The BTB is updated at the commit stage, as with a conventional BTB. When a branch instruction is committed, the BTB is updated only if the branch instruction is valid. If the committed instruction is a branch instruction that is invalid or if it is a non-branch instruction, the BTB is not updated.

For an update, as with the reference, the BTB is accessed using the same index as that employed to reference the instruction cache. To replace the BTB entry, the instruction number and the way number of the TPCs present in the accessed set are compared with the instruction number and the way number of the branch instruction to be committed. The LRU method is used to replace the entry.

Figure 3 illustrates tag and data ports of the instruction cache, revised for our proposed mechanism. For the comparison and the storage of the way numbers, it is necessary to determine the way in the instruction cache where the branch instruction to be committed is present. The way number is obtained by accessing only the instruction cache tag. Since this operation may compete with access to the instruction cache for a normal instruction fetch, the number of instruction cache ports is increased only for the tag in the proposed mechanism. The number of ports is one of the factors that determine the energy of I-cache. If the number of ports increases, the energy increases. Therefore, the increase in the number of ports for the proposed BTB results in an increase in energy consumption of the instruction cache per access. It is also necessary to evaluate this negative impact of the proposed BTB on the instruction cache when the proposed BTB is evaluated in Section 6.

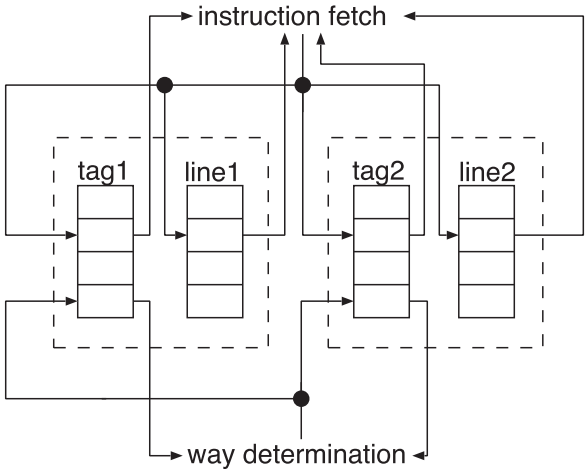

Fig. 3 Tag and data ports of instruction cache.

5.4 Victim BTB

In our proposed mechanism, if the cache line is replaced due to an instruction cache error, all of the TPCs in the BTB for the way where the replaced cache line was present are removed. The purpose of this is to prevent a branch prediction error when the TPC for the instruction replaced and removed from the instruction cache is stored in the BTB and an incorrect TPC is set to the predicted address for the newly stored instruction. However, removal of the TPC is not enough. When the cache line for the TPC is stored in the instruction cache again, a branch prediction cannot be made, and performance declines. To prevent this, we propose a mechanism in which a table (a victim BTB) is added to store the removed TPC and the TPC is returned from the victim BTB to the BTB when the cache line containing the instruction for the TPC returns to the instruction cache.

Figure 4 shows the relationship between the BTB and the victim BTB. The victim BTB is configured in the same way as a normal BTB, except for the difference in the index and the tag. The victim BTB is accessed for a reference and an update when a tag error occurs in the L1 instruction cache and the cache line is replaced.

When the victim BTB is updated, moderately significant to significant bits of the instruction present in the cache line removed due to a cache error and the instruction number of the TPC removed as a result of the removal of the cache line are used as the tag and the index (the instruction address of the branch instruction for the TPC can be reconstituted by combining the above two), and the removed TPC is stored in the victim BTB.

A PC is used to reference the victim BTB if an instruction cache error occurs. Entries in the victim BTB, in which the TPCs for the branch instructions present in the cache line for the PC are stored, are searched for, and as many relevant TPCs as possible are returned to the BTB. "As many as possible" here means "as long as there are empty BTB entries for the TPCs." As described above, as a reference, multiple instructions in the same cache line must be searched for with one PC. Therefore, all TPCs for the instructions present in one cache line need to be stored in the same set. As a result, the "$\log_2$(*the number of victim BTB sets*)" bits after the exclusion of the line offset to achieve the above is the index for the victim BTB.

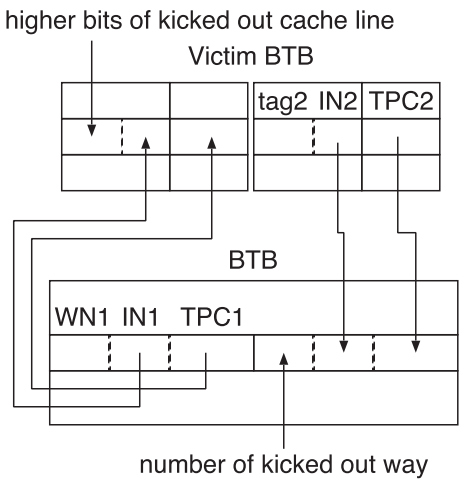

Fig. 4 Victim BTB.

The TPC needs to be returned from the victim BTB to the BTB if an L1 instruction cache error occurs. In this case, the L2 cache is referenced to obtain a new cache line, causing long latency. The operation to return the TPC from the victim BTB to the BTB is performed when the L2 cache is being referenced, so the latency of this operation is hidden.

6. Evaluation

6.1 Evaluation Environment

To confirm the effectiveness of our proposal, we focused on the following evaluation items: the overall energy consumption of the conventional and the proposed BTBs and rate of reduction in instructions per cycle (IPC) of the processor with a conventional BTB or our proposed BTB. To check the overhead of our proposal, we also evaluated the total area needed for implementing the proposed BTB.

We revised the Simcore [13], which is a software-based processor simulator, to evaluate the IPC of a superscalar processor and the number of accesses to the conventional and proposed BTBs and I-cache, and also used CACTI [14] and the Synopsys Design Compiler to evaluate the delay, the area, and the energy, which includes both the dynamic energy per access and the static energy per cycle due to leakage current, of the conventional and proposed BTBs and the I-cache.

We explained how to evaluate the delay, the area, and energy in detail. We did not evaluate the whole processor, but instead evaluated the following components.

- BTB which was evaluated by CACTI
- I-cache which was evaluated by CACTI
- victim BTB which was evaluated by CACTI
- "Others" which were evaluated by CACTI and DC [*1]

The BTB, the I-cache, and the victim BTB have already been described. The reason that the I-cache is evaluated is mentioned in Section 5.3. "Others" is the general term for all the other circuits other than the BTB and victim BTB in the proposed BTB. For example, in Fig. 2, comparators, AND gates, and so on under the BTB and the I-cache are categorized as "Others."

The BTB, I-cache, and victim BTB are buffers. The gate level structures of buffers have already been defined and are able to be evaluated by CACTI. We therefore evaluated the delay, the area, and their energy using only CACTI.

On the other hand, "Others" are not buffers, and the gate level structures of "Others" are unknown. It was impossible to evaluate them by CACTI. We therefore evaluated the delay, the area, and energy of "Others" as follows: Firstly, we wrote only "Others" in Verilog HDL. Secondly, we performed logic synthesis on them using Synopsys Design Compiler to generate gate level structures. Thirdly, we revised CACTI for the gate structures. Lastly, we evaluated the delay, the area, and energy of "Others" using CACTI.

In the processor simulation, we employed Alpha instruction set architecture. The SPEC benchmark is often used for evaluating the BTB. For example, in Section 3, which describes related studies, all studies except for Ref. [10] used the SPEC benchmark for the evaluation. We therefore also chose the SPEC benchmark and used bzip2, gobmk, h264ref, hmmer, mcf, omnetpp, perlbench, and sjeng from SPECint2006 as benchmark programs. To skip the initialization phase of each benchmark program, 2 billion instructions were skipped before 100 million instructions are run for the evaluation. **Table 1** shows the processor configuration used for measurement. As shown in Table 1, the number of instruction cache ports for tags and the data are 2 and 1, respectively. The reason for this instruction cache configuration is described in Section 5.3. For our evaluation of delay, area, and energy consumption, CACTI and the Synopsys Design Compiler used the same 32 nm-process technologies and the same 0.9 V supply voltage. The number of I/O ports of an evaluated table (for example, a BTB) was set to the value needed to prevent competition among ports.

[*1] 'DC' is short for 'Design Compiler'.

Table 1 Processor configurations.

Fetch Width	4 instruction/cycle
Decode Width	4 instruction/cycle
Issue Width	4 instruction/cycle
Commit Width	4 instruction/cycle
Instruction Window	64 entry Register Update Unit, 64 entry Load/Store Queue
Functional Unit	4 iALU, 2 iMULT/DIV, 3 fpALU, 2 fpMULT/DIV/SQRT
L1 I-cache	32 kB [*2], 2 way, 32 byte cache line, 1 cycle hit latency, 2 port for tag, 1 port for data
L1 D-cache	64 kB, 4 way, 64 byte cache line, 2 port, 1 cycle hit latency
L2 cache	256 kB, 4 way, 64 byte cache line, 10 cycle hit latency
Main memory	8 byte/cycle band width, 100 cycle latency
Branch predictor	gshare (11 bit history, 8k entry PHT), 32 entry Return Address Stack

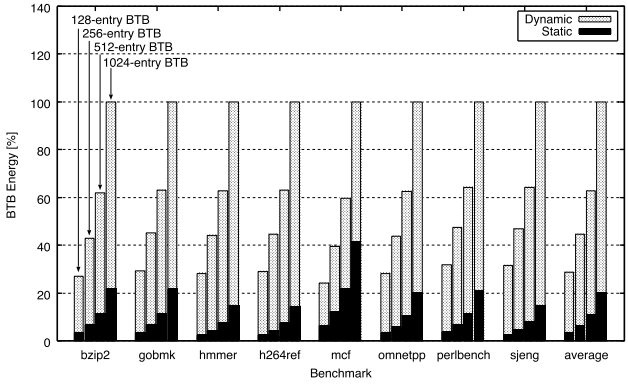

Fig. 5 Overall dynamic and static energy of conventional BTB.

6.2 The Conventional BTB

Figure 5 shows the overall energy consumption including both dynamic and static of the conventional BTB. **Table 2** shows the IPC degradation rate of the conventional BTB. The number of entries of the BTB ranged from 128 to 1,024. The number of ways of the BTB was fixed at 2.

In Fig. 5, the vertical axis indicates the overall energy consumption of the conventional BTB with each number of entries, normalized to that of the 1,024-entry conventional BTB. The hor-

[*2] Only in Section 6.3.3, we varied the I-cache capacity from 16 kB to 64 kB.

Table 2 IPC degradation rate of conventional BTB.

Benchmark	Num. of BTB Entries			
	128	256	512	1,024
bzip2	0.00%	0.00%	0.00%	0.00%
gobmk	4.73%	3.29%	1.35%	0.00%
hmmer	0.00%	0.00%	0.00%	0.00%
h264ref	1.82%	0.72%	0.10%	0.00%
mcf	0.00%	0.00%	0.00%	0.00%
omnetpp	2.20%	1.10%	0.34%	0.00%
perlbench	14.31%	9.16%	3.66%	0.00%
sjeng	8.59%	4.92%	1.81%	0.00%
average	4.08%	2.45%	0.92%	0.00%

izontal axis shows the benchmarks. Each group of four bars represents the normalized overall energy consumption for the 128-, 256-, 512-, and 1,024-entry BTBs. Each bar has two parts: "dynamic" and "static," which respectively indicate the dynamic energy and the static energy. Table 2 shows the IPC degradation rate of the conventional BTB with each number of entries compared with that for the 1,024-entry conventional BTB.

Figure 5 confirms that both of overall dynamic and static energy consumption grow as entry size increases. Table 2 shows that the IPC degradation rate gradually approaches 0% as the entry size increases. When the number of entries of the BTB is increased from 512 to 1,024, the change in IPC degradation rate is less than 1%. In other words, it approaches the lower limit. This makes the 2-way and 1,024-entry conventional BTB more suitable if higher performance is required, although it consumes more energy overall.

Based on the above evaluation results, we used the 2-way and 1,024-entry conventional BTB as our benchmark for the following evaluation of the overall energy consumption and the IPC reduction rate of our proposed BTB unless stated otherwise. In this configuration, the ratio of dynamic to static is about 4 to 1 on average.

6.3 Proposed BTB

6.3.1 Influence of BTB Size

Figures 6, **7**, and **Table 3** show the overall dynamic energy consumption, the overall static energy consumption, and the IPC degradation rate, respectively, of the proposed BTB. We varied the number of entries of the BTB from 1,024 to 4,096. It should be noted that the number of BTB sets is always equal to the number of I-cache sets in the proposed BTB. Therefore, the number of the BTB sets is fixed at 512 and the number of ways of the BTB changes from 2 to 8 as the number of the BTB entries increases. On the other hand, the number of ways and entries of the victim BTB are fixed at 4 and 128, respectively.

In Fig. 6, the vertical axis indicates the overall dynamic energy consumption of the proposed BTB, normalized to that of the 1,024-entry conventional BTB. The horizontal axis shows the benchmarks. Each group of three bars indicates the normalized overall dynamic energy consumption for the 1,024-, 2,048-, and 4,096-entry BTB from left to right, respectively. Each bar has four parts: "others," "victim," "i-cache," and "btb." "btb" indicates the dynamic energy consumed by the BTB. "i-cache" indicates the increments in I-cache dynamic energy due to the increase in the I-cache ports and accesses. "victim" indicates the dynamic energy consumed by the victim BTB. "others" indicates

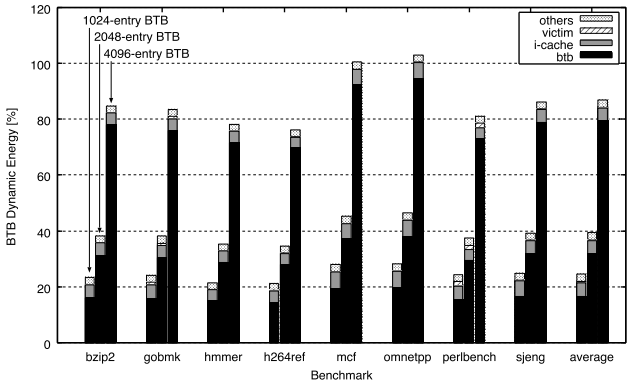

Fig. 6 Overall dynamic energy of proposed BTB (4-way and 128-entry victim BTB).

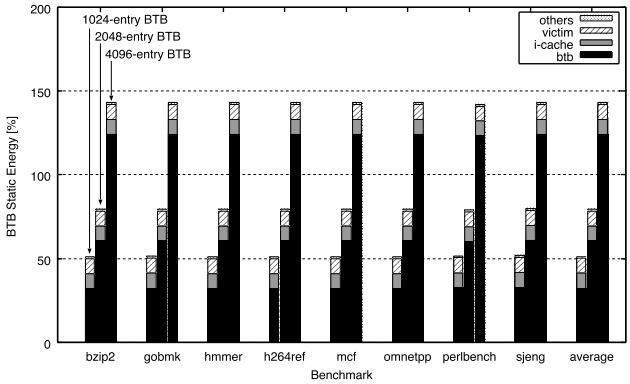

Fig. 7 Overall static energy of proposed BTB (4-way and 128-entry victim BTB).

Table 3 IPC degradation rate of proposed BTB (4-way and 128-entry victim BTB).

Benchmark	Num. of BTB Entries		
	1,024	2,048	4,096
bzip2	0.00%	0.00%	0.00%
gobmk	0.59%	0.00%	0.00%
hmmer	0.00%	0.00%	0.00%
h264ref	0.19%	−0.05%	−0.05%
mcf	0.00%	0.00%	0.00%
omnetpp	0.21%	0.00%	0.00%
perlbench	1.13%	−0.70%	−0.87%
sjeng	1.58%	0.23%	−0.06%
average	0.46%	−0.06%	−0.12%

the dynamic energy consumed by the other circuits which do not belong to the BTB, the I-cache, or the victim BTB. For example, in Fig. 2, the "other" circuits are comparators, AND gates, and so on under the BTB and the I-cache. The sum of "others," "victim," and "i-cache" is the additional dynamic energy consumption which is required only in the processor with the proposed BTB.

Figure 7 is the same as Fig. 6, except that the vertical axis indicates the overall static energy consumption of the proposed BTB, normalized to that of the 1,024-entry conventional BTB. Table 3 shows the IPC degradation rate of the proposed BTB for each number of BTB entries compared with the 1,024-entry conventional BTB.

Figure 6 confirms that the overall dynamic energy consumption of the proposed BTB is significantly reduced when the number of BTB entries is equal to or fewer than 2,048. When the number of BTB entries increases, the additional dynamic energy (the sum of "others", "victim", and "i-cache" denoted in the figure) changes

very little. This is because the number of BTB entries is not directly related to I-cache accesses or victim BTB accesses. On the other hand, the BTB dynamic energy (denoted "btb" in the figure) can be reduced using our proposed mechanism, but increases almost proportionally to the number of entries, since the number of BTB entries increases the BTB dynamic energy requirement per access. As a result, although the overall dynamic energy consumption is low, at 60.6% on average if the number of BTB entries is 1,024, the effect of the overall dynamic energy reduction is significantly decreased when the number of the BTB entries reaches 4,096.

The overall trend shown in Fig. 7 is similar to that shown in Fig. 6. However, there are differences, as shown below. A) The static energy depends little on benchmarks because the proposed mechanism has zero or negligible influence on the IPC, as explained in the next paragraph. B) The reduction rate in the static energy is lower than that in the dynamic energy. Therefore, if the BTB entry size is equal to or less than 2,048, the overall static energy can be reduced; however, otherwise, the overall static energy significantly exceeds 100%. The reason that the static energy reduction is possible in the 1,024- and 2,048-entry BTBs is that the tag field is removed from the proposed BTB and the static energy reduction due to the elimination of the tag field is greater than the static energy increase due to the other additional components, "i-cache," "victim," and "others," in Fig. 7.

Meanwhile, Table 3 confirms that the proposed BTB can maintain almost the same IPC regardless of the number of BTB entries. When the number of the BTB entries is 1,024, in some benchmarks, the IPC is slightly degraded, although the average IPC degradation rate is less than 1%. The slight degradation of the IPC is caused by competition between the branches associated with the same entry of the BTB. On the other hand, if the BTB size is 2,048 entries or bigger, there is no IPC degradation in the results due to the alleviation of BTB competition.

Figures 6, 7, and Table 3 indicate that if some IPC degradation is acceptable, the 1,024-entry BTB is most appropriate, because the overall energy consumption including both dynamic and static can be reduced by as much as 70.1% on average. Otherwise, if lower energy is required without IPC degradation, the 2,048-entry BTB, which can reduce the overall energy consumption by 52.5%, is the most suitable. It should be noted that the overall energy reduction rate is not equal to the arithmetic mean of the dynamic and static energy reduction rates shown in Figs. 6 and 7, because the ratio of dynamic to static is not one to one as shown in Fig. 5.

For the parameters used in this subsection, we evaluated the total area of all the components of the proposed mechanism and the increment in the I-cache due to the proposed mechanism. The evaluation results show that the total area is 0.040 mm^2, 0.059 mm^2, and 0.100 mm^2 when the number of the BTB entries is 1,024, 2,048, and 4,096, respectively.

6.3.2 Influence of Victim BTB Size

Figures 8, **9**, and **Table 4** show the overall dynamic energy consumption, the overall static energy consumption, and the IPC degradation rate, respectively, of the proposed BTB. In this subsection, we vary the number of entries of the victim BTB from 64 to 256. The number of ways of the victim BTB is fixed at 4, and the number of ways and entries of the BTB are fixed at 4 and 2,048, respectively.

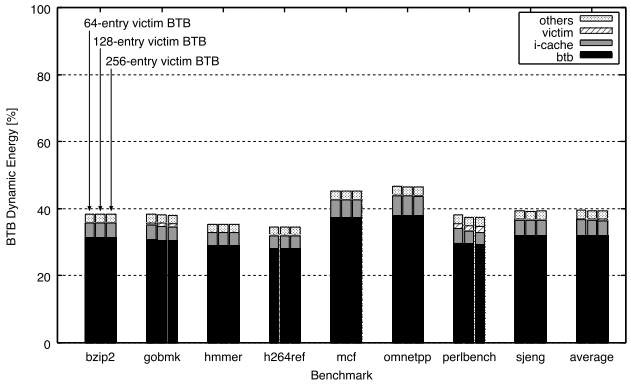

Fig. 8 Overall dynamic energy of proposed BTB (4-way and 2,048-entry BTB).

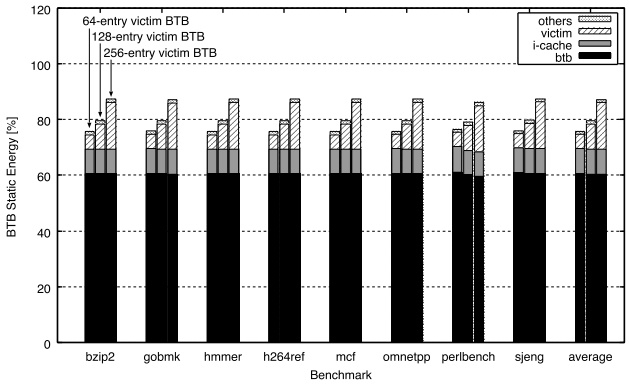

Fig. 9 Overall static energy of proposed BTB (4-way and 2,048-entry BTB).

Table 4 IPC degradation rate of proposed BTB (4-way and 2,048-entry BTB).

Benchmark	Num. of victim BTB Entries		
	64	128	256
bzip2	0.00%	0.00%	0.00%
gobmk	0.34%	0.00%	−0.25%
hmmer	0.00%	0.00%	0.00%
h264ref	−0.05%	−0.05%	−0.05%
mcf	0.00%	0.00%	0.00%
omnetpp	0.14%	0.00%	0.00%
perlbench	1.13%	−0.70%	−1.48%
sjeng	0.45%	0.23%	0.17%
average	0.25%	−0.06%	−0.20%

There is a great deal of similarity between Fig. 6 and Fig. 8. The differences are as follows. In Fig. 8, each group of three bars indicates the normalized overall dynamic energy consumption for the 64-, 128-, and 256-entry victim BTB from left to right, respectively. Figure 9 is the same as Fig. 8, except that the vertical axis indicates the overall static energy consumption of the proposed BTB, normalized to that of the 1,024-entry conventional BTB. Table 4 shows the IPC degradation rate of the proposed BTB for each number of victim BTB entries compared with the 1,024-entry conventional BTB.

Figure 8 confirms that the overall dynamic energy consumption of the proposed BTB changes little in relation to the number of victim BTB entries. The reason is as follows. The dynamic energy per access of the victim BTB significantly increases as its

size increases. However, the number of accesses to the victim BTB is very small, because the victim BTB is accessed when an I-cache error occurs, so any change in victim BTB dynamic energy consumption will have a negligible influence on the overall dynamic energy consumption requirements of the proposed BTB.

Figure 9 confirms that the overall static energy consumption of the proposed BTB can still be reduced, although the static energy of the victim BTB steadily increases as the number of victim BTB entries increases. The reason is that the static energy reduction due to the elimination of the tag field is large enough to be above the static energy of the additional components.

Table 4 confirms that our proposed BTB sustains almost the same number of IPC if the number of the victim BTB entries changes. If the number of victim BTB entries is 64, the IPC is slightly degraded by up to 1.13%, or by an average of 0.25%. However, if the victim BTB size is equal to or larger than 128 entries, there is no degradation in IPC in all benchmarks except for sjeng.

Based on the results of Figs. 8, 9, and Table 4, the 128-entry victim BTB is enough to significantly reduce the overall energy consumption without any IPC degradation on average.

The total area required for implementing the proposed BTB in the cases of 64-, 128-, and 256-entry victim BTB is 0.058 mm^2, 0.059 mm^2, and 0.062 mm^2, respectively. As in the previous subsection, the total area includes the area of the additional circuits in addition to that of the BTB.

6.3.3 Influence of I-cache Capacity

The number of sets of the proposed BTB is always equivalent to the sets of the I-cache. In other words, the configuration of the proposed BTB depends partially on that of the I-cache. The I-cache capacity might therefore have an influence on the overall energy consumption and the IPC degradation of the proposed BTB. To confirm the influence of I-cache capacity, in this section, we varied the I-cache capacity from 16 kB to 64 kB while maintaining the ways at 2 and the line size at 32 bytes. The number of BTB sets therefore changes from 256 to 1,024 as the I-cache capacity increases. For each I-cache capacity, we also determined the appropriate parameters for the conventional and proposed BTBs in the same ways as for Sections 6.2, 6.3.1, and 6.3.2. The determined parameters are shown in **Table 5**.

Figures 10, **11**, and **Table 6** show the overall dynamic energy consumption, the overall static energy consumption, and the IPC degradation rate of the proposed BTB when the I-cache capacity is varied. Figure 10 is quite similar to Figs. 6 and 8. The difference is as follows. In Fig. 10, each group of three bars indicates, from left to right, the normalized overall dynamic energy of the conventional BTBs in the cases of 16-kB, 32-kB, and 64-kB I-caches. Figure 11 is the same as Fig. 10, except that the vertical axis indicates the overall static energy consumption of the proposed BTB, normalized to that of the 1,024-entry conventional BTB. Table 6 shows the IPC degradation rate of the proposed BTB in the cases of the 16-kB, 32-kB, and 64-kB I-caches. The configuration of the conventional BTB, which is used as a basis for the calculations of the normalized dynamic energy, the normalized static energy, and the IPC degradation rate, in each I-cache capacity is shown in Table 5, as already noted.

Figures 10, 11, and Table 6 confirm that the proposed BTB significantly reduces the overall energy consumption while retaining almost the same IPC as the conventional BTB, even when the I-cache capacity is changed. Only when the I-cache capacity is 16 kB does the IPC slightly decrease on average. The reason for this IPC degradation is that the small size of the BTB sets causes competition between branches during BTB entry.

Table 5 confirms that the sum of the BTB entry size and the

Table 5 Configurations of conventional and proposed BTBs for each I-cache capacity.

I-cache Capacity	Proposed BTB		Conventional BTB
	BTB	victim	
16 kB	4 way, 1,024 entry	4 way, 128 entry	2 way, 1,024 entry
32 kB	4 way, 2,048 entry	4 way, 128 entry	2 way, 1,024 entry
64 kB	4 way, 4,096 entry	4 way, 128 entry	2 way, 2,048 entry

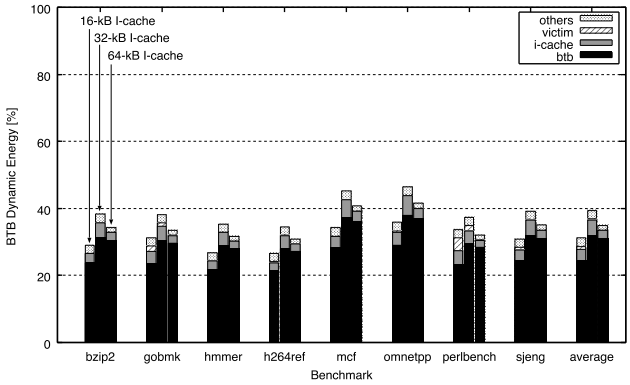

Fig. 10 Influence of I-cache capacity overall on dynamic energy of proposed BTB.

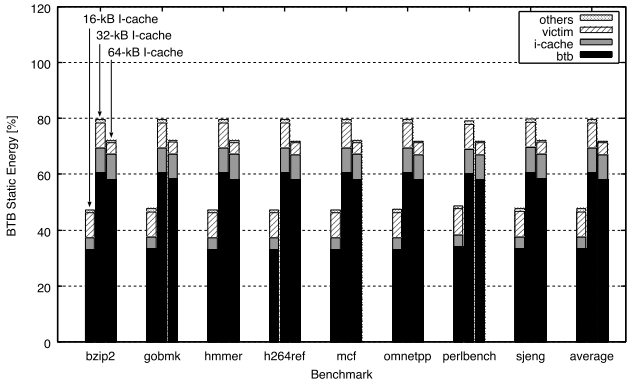

Fig. 11 Influence of I-cache capacity on overall static energy of proposed BTB.

Table 6 Influence of I-cache capacity on IPC degradation rate of proposed BTB.

Benchmark	I-Cache Capacity		
	16 kB	32 kB	64 kB
bzip2	0.00%	0.00%	0.00%
gobmk	0.94%	0.00%	0.07%
hmmer	0.05%	0.00%	0.00%
h264ref	0.00%	−0.05%	−0.05%
mcf	0.00%	0.00%	0.00%
omnetpp	0.29%	0.00%	−0.20%
perlbench	3.06%	−0.70%	−0.19%
sjeng	1.17%	0.23%	0.11%
average	0.69%	−0.06%	−0.03%

victim BTB entry size in the proposed BTB is greater than the BTB entry size in the conventional BTB. However, since our proposed BTB eliminates the tag array and reduces both the number of BTB ports and the number of accesses to the BTB, the increase in the number of entries has no negative effect on overall energy consumption.

The total area required for implementing the proposed BTB in the cases of a 16-kB, 32-kB, and 64-kB I-cache is 0.038 mm^2, 0.059 mm^2, and 0.114 mm^2, respectively. The total area includes the area taken up by the additional circuits described in the previous subsections in addition to that of the BTB.

6.3.4 Influence on Delay

In the conventional BTB, the delay of the branch target prediction is determined exclusively by the BTB. On the other hand, in the proposed BTB, the delay depends on the BTB, the victim BTB, and "Others" as noted in Section 6.1.

To facilitate understanding, firstly, we focus solely on the conventional and proposed BTBs. We evaluated the delay of the 2-way and 1,024-entry conventional BTB and the proposed BTB with the 4-way and 128-entry victim BTB and the 4-way and 2,048-entry BTB. The evaluation results show that the delay of the conventional and proposed BTBs is 281.3 ps and 266.3 ps, respectively. These results confirm that the proposed BTB improves the delay of the branch target prediction. The reason is as follows. In a conventional BTB, the critical path is determined exclusively by the BTB. Meanwhile, in the proposed BTB, the critical path is determined by the BTB and "Others." The delay of "Others" is 35.9 ps, negatively affecting the critical path of the proposed BTB. However, the delay of the BTB is reduced to 230.4 ps by deletion of the tag field. The effect of this deletion more than compensates for the loss caused by "Others."

Secondly, we focus on the proposed BTB and the revised instruction cache for the proposed BTB. As shown in Fig. 2, the tag array of the instruction cache and part of "Others" are connected in series. There is a possibility that the delay of the path passing through the above two components is longer than the delay of the instruction cache or the delay of the proposed BTB. Therefore, in addition to the proposed BTB, we evaluated the delay of the data array and the tag array of the instruction cache and the relevant part of "Others" which are connected to the tag array in series. The evaluation results are as follows: The delay of the data array is 289.2 ps. The delay of the tag array is 230.3 ps. The delay of the relevant part of "Others" is 36.2 ps. The above results confirm that the delay of the path passing through the tag array and the relevant part of "Others" is 266.5 ps, almost equivalent to the delay of the proposed BTB, and shorter than the delay of the data array. Therefore, the delay of the instruction cache is not affected by the proposed BTB and the delay of the proposed BTB is minimally affected by the instruction cache.

6.4 Comparison with Related BTB

In this section, we compare the proposed BTB with two related mechanisms (RMs), which incur one or no delay cycles per branch prediction. We denote them RM1 and RM2. RM1 was proposed by Kahn et al. [2]. RM2 was proposed by Chung et al. [11]. Summaries of these have been given in Section 3.

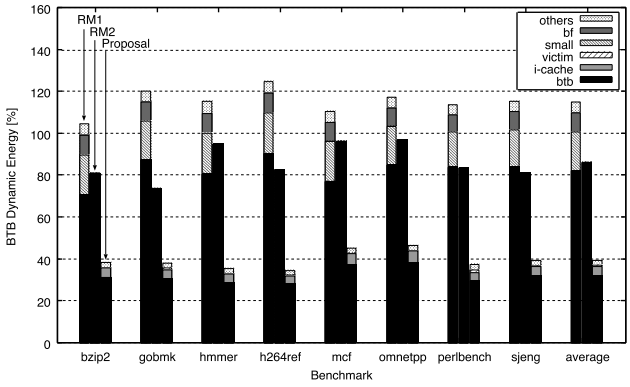

Fig. 12 Overall dynamic energy of related mechanisms.

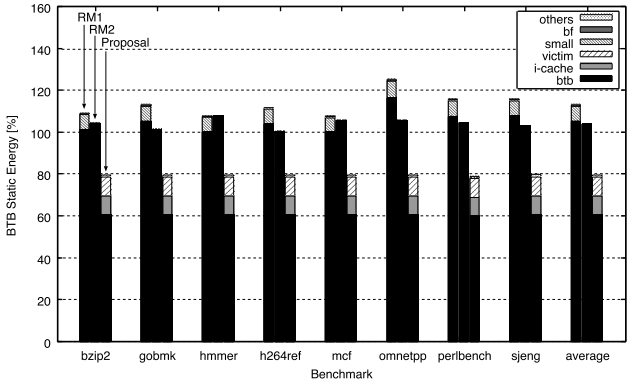

Fig. 13 Overall static energy of related mechanisms.

Table 7 IPC degradation rate of related mechanisms.

Benchmark	RM1	RM2	Proposal
bzip2	1.25%	3.92%	0.00%
gobmk	4.81%	1.35%	0.00%
hmmer	0.05%	7.06%	0.00%
h264ref	3.63%	0.19%	−0.05%
mcf	0.00%	5.06%	0.00%
omnetpp	13.96%	5.09%	0.00%
perlbench	6.98%	4.45%	−0.70%
sjeng	7.06%	2.77%	0.23%
average	4.82%	3.76%	−0.06%

Figures 12, **13**, and **Table 7** show the total dynamic energy demand, the total static energy demand, and the IPCs, respectively, of our proposed BTB, RM1, and RM2. A 32-kB I-cache is used. In the proposed BTB, we use a 4-way and 2,048-entry BTB and 4-way and a 128-entry victim BTB. In RM1 and RM2, each configuration of BTBs is the same as that of a conventional BTB. In RM1, the 2-way and 64-entry small BTB is used and the configuration of the Bloom filter is the same as that described in Ref. [11]. The 2-way and 1,024-entry conventional BTB is used as a basis for the calculations of the normalized dynamic energy, the normalized static energy, and the IPC degradation rate. In Fig. 12, each group of three bars indicates, from left to right, the normalized overall dynamic energy of RM1, RM2, and the proposed BTB. In the figure, there are six explanatory notes: "others," "bf," "small," "victim," "i-cache," and "btb." "btb" indicates the dynamic energy consumed by the BTB. The meanings of "victim" and "i-cache" are the same as those in Fig. 6–Fig. 11. "bf" and "small" indicate the dynamic energy consumed by the Bloom filter and the small BTB, respectively, in RM1. "others" indicates the dynamic energy consumed by the other circuits which are not

categorized as "bf," "small," "victim," "i-cache," or "btb," but are required for implementing the proposed BTB, RM1, or RM2. For example, "others" in the proposed BTB is already explained in Section 6.3.1. Table 7 shows the IPC degradation rate for each mechanism compared with the conventional BTB.

Figure 12 confirms that the proposed BTB reduces the overall dynamic energy consumption on average more than the others. In RM1, the overall dynamic energy is increased by an average of 14.9%. The reason for the increase is as follows. RM1 uses a Bloom filter to detect any instructions that are not registered in the BTB, regardless of branch or nonbranch. The instruction uses an instruction address to access some entries in the Bloom filter. If at least one of the accessed entries is zero, the instruction is found not to be registered in the BTB and is filtered out. However, in most cases, the Bloom filter does not work effectively because the lowest partition of the address used for accesses to the Bloom filter provides the most information. This problem is reported in Ref. [12]. As a result, many instructions reach the BTB without being filtered out. Another drawback is that the Bloom filter and the small BTB require dynamic energy. In RM2, the average dynamic energy reduction rate averages 13.9%. The reason that the rate is lower is as follows. RM2 accesses gshare to predict the branch direction of an instruction one cycle earlier than the instruction is fetched. If the predicted direction is Taken, the BTB is accessed. Otherwise, it is not accessed. If the fetched instruction is a branch, this mechanism works well. The accesses to the BTB from Not Taken branches are eliminated. However, if it is a nonbranch, this mechanism cannot accurately judge whether to stop BTB access, since gshare is a prediction mechanism for branches.

Figure 13 confirms that the proposed BTB reduces the overall static energy consumption on average more than "others." In RM1, the overall static energy is increased by an average of 15.0% for the following reasons. For the reduction of the dynamic BTB energy, RM1 prepares additional components that consume static energy: "small," "bf," and "others" in Fig. 13. RM1 also degrades the IPC in most cases, as mentioned in the next paragraph, increasing the time during which the leakage current is flowing. In RM2, the overall static energy increases by an average of 3.9% because the IPC is increased, similar to RM1. A more detailed explanation of IPC degradation is given in the next paragraph. To reduce the dynamic energy of the BTB, RM2 uses the output of the already existing branch direction predictor. Therefore, unlike RM1 or the proposed BTB, RM2 does not need to prepare any dedicated buffers that consume static energy. Although small-scale circuits such as multiplexers are required to implement RM2, their static energy (denoted "others") is too small to see in Fig. 13.

Meanwhile, Table 7 shows that the average IPC of the proposed BTB is larger than that of the others. RM1 degrades the IPC by 4.82% on average. If the small BTB access misses and the Bloom Filter hits, the fetched instruction accesses the BTB, delaying the branch prediction. This delay adversely affects the IPC. RM2 degrades the IPC by 3.76% on average. The reason is that the gshare used in RM2 is revised to assist dynamic energy reduction, but the revision has a negative impact on branch prediction.

The total areas required for implementing the proposed BTB, RM1, and RM2 are 0.059 mm^2, 0.182 mm^2, and 0.159 mm^2, respectively. As in the previous subsection, the total area includes the area of the additional circuits in addition to that of the BTB. The area of the proposed BTB is the smallest because the tag field is removed; the area of the removed tag field is greater than that of the other additional components. The area of RM1 is the largest because RM1 requires additional components irrespective of any area reduction of the BTB, unlike the proposed BTB. The area of RM2 is close to that of the conventional BTB because RM2 makes use of pre-existing components plus only a few extra circuits.

7. Conclusion

In this paper, we propose a method that reduces energy demand by branch target prediction by cutting the number of BTB accesses. In our proposed mechanism, we integrate the BTB tag and I-cache tag. Further, we predict the branch target in units of cache lines. We then predict the branch target address of a plurality of instructions present in the cache line using one BTB reference. As a consequence, we reduce the number of BTB accesses and cut the energy cost of branch target prediction without any performance loss.

To evaluate the proposed mechanism, we varied the number of entries of the BTB from 1,024 to 4,096. The results showed that, for BTB 2,048-entry, the energy consumption was reduced to 47.5% of that required by a conventional mechanism, without causing any decline in IPC.

Acknowledgments A part of this research has been supported by JSPS KAKENHI Grant Number 25330060 and 26330063. The authors wish to thank Hiroki Yamamoto for technical assistance and helpful discussions that he provided when he was a master student at Toyohashi University of Technology.

References

[1] Hennessy, J.L. and Patterson, D.A.: *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., 2nd edition (1996).
[2] Kahn, R. and Weiss, S.: Thrifty BTB: A Comprehensive Solution for Dynamic Power Reduction in Branch Target Buffers, *Microprocess. Microsyst.*, Vol.32, No.8, pp.425–436 (2008).
[3] Sadeghi, H., Sarbazi-Azad, H. and Zarandi, H.R.: Power-aware branch target prediction using a new BTB architecture, *2009 17th IFIP International Conference on Very Large Scale Integration* (*VLSI-SoC*), pp.53–58 (2009).
[4] McFarling, S.: Combining branch predictors, WRL Technical Note, TN-36, Digital Equipment Corporation (1993).
[5] Jiménez, D.A. and Lin, C.: Dynamic branch prediction with perceptrons, *Proc. 7th International Symposium on High-Performance Computer Architecture*, pp.197–206 (2001).
[6] Fagin, B. and Russell, K.: Partial resolution in branch target buffers, *Proc. 28th Annual ACM/IEEE International Symposium on Microarchitecture*, pp.193–198 (1995).
[7] Calder, B. and Grunwald, D.: Fast and accurate instruction fetch and branch prediction, *Proc. 21st Annual International Symposium on Computer Architecture*, pp.2–11 (1994).
[8] Driesen, K. and Hölzle, U.: The cascaded predictor: Economical and adaptive branch target prediction, *Proc. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pp.249–258 (1998).
[9] Parikh, D., Skadron, K., Zhang, Y., Barcella, M. and Stan, M.R.: Power issues related to branch prediction, *Proc. 8th International Symposium on High-Performance Computer Architecture*, pp.233–244 (2002).

[10] Petrov, P. and Orailoglu, A.: Low-power Branch Target Buffer for Application-Specific Embedded Processors, *Proc. Euromicro Symposium on Digital Systems Design*, pp.158–165 (2003).
[11] Chung, S.W. and Park, S.B.: A Low Power Branch Predictor to Selectively Access the BTB, *Advances in Computer Systems Architecture*, pp.374–384 (2004).
[12] Peir, J.K., Lai, S.C. and Lu, S.L.: Bloom filtering cache misses for accurate data speculation and prefetching, *Proc. 16th Annual ACM International Conference on Supercomputing*, pp.189–198 (2002).
[13] Kise, K., Honda, H. and Yuba, T.: SimAlpha Version 1.0: Simple and Readable Alpha Processor Simulator, *LNCS*, Vol.2823, pp.122–136 (2003).
[14] Muralimanohar, N., Balasubramonian, R. and Jouppi, N.P.: *CACTI 6.0: A tool to model large caches*, HP Laboratories (2009).

Ryotaro Kobayashi received his B.E., M.E., and D.E. degrees from Nagoya University in 1995, 1997, and 2001, respectively. He had been a research assistant in Nagoya University from 2000 to 2008. He is currently a lecturer at Toyohashi University of Technology. His research interests include computer architecture, parallel processing, and network security.

Kaoru Saito is currently a Bachelor Course student of Toyohashi University of Technology. His research interests include computer architecture.

Hajime Shimada was born in 1976 and received his B.E., M.E. and D.E. degrees from Nagoya University, Japan in 1998, 2000 and 2004 respectively. He was an assistant professor in Kyoto University from 2005 to 2009. He wan an associate professor in NAIST from 2009 to 2013. He is now an associate professor in Nagoya University, Japan since 2013. He is currently focusing on computer architecture and network related researches with low power consumption and high dependability techniques. He is a member of IEEE, IPSJ, and IEICE.