An Optimal CAM-based Separated BTB for a Superscalar Processor

Lin Meng, Kosaku Fukuda, Takeshi Kumaki, and Takeshi Ogura

***Abstract*—Branch target buffer (BTB) is an important component for predicting branch target addresses to improve the performance of superscalar processor. However, BTB misprediction increases penalty by using deeper pipelines and larger windows in a current processor. Hence, increasing the accuracy of BTB prediction has become more important. This paper proposes a novel BTB that separates current BTB into conditional branch BTB (CBTB) and non-conditional branch BTB (NBTB). The CBTB uses the current BTB, and the NBTB is added on the current BTB. For optimization the separated BTB, we test NBTB by using two kinds of memory structures. One is static random access memory (SRAM) and the other is content addressable memory (CAM). For the replacement algorithms of CAM, we test a least recently used method and a rotation method. We equip our BTB on FPGA to measure the hardware size and use SimpleScalar to measure the performance. The experiment results show that proposed BTB improved IPC about 3.12% by adding an optimum of 128 entries to the current BTB with a CAM structure, and the optimal replacement algorithm is the rotation method.**

***Index Terms*—Branch target buffer, superscalar processor, FPGA.**

I. Introduction

Current superscalar processors use deeper pipelines and wider instruction issues to exploit instruction level parallelism. However, branch mispredictions incur a heavy penalty when they occur, such as wasting numerous cycles and power to recover from mispredictions. Hence, increasing the accuracy of branch predictions is more important for superscalar processors. The current branch predictor consists of a branch direction predictor using a pattern history table (PHT) and a branch target address predictor using a branch target buffer (BTB) [1]-[5].

Moreover, the BTB is widely used in embedded processors too [6], [7], and software CPU is widely used in FPGA as an embedded processor, like MicroBlaze [8] by xilinx, and Nios by Altera [9]. Thus, installing an effective BTB is important for FPGA software processors too.

The current BTB is shared by conditional branches and non-conditional branches. From our analysis, we found that many BTB misses were caused by conditional branches and non-conditional branches using the same BTB entry. Consequently, we separate the current BTB into a conditional branch BTB (CBTB) and a non-conditional branch BTB (NBTB) to reduce the number of BTB mispredictions. In our proposal, the CBTB uses the current BTB, and NBTB is an additional hardware. To optimize the hardware size and the performance, the structure of NBTB is tested by using an SRAM and CAM [10]-[13] because the replacement algorithms of CAM have a relationship with hardwire size and the searching hit rate of CAM. We test the replacing method by using a least recently used method (LRU) and a rotation method [14].

This paper describes the designs of the new BTB, measurements of the size of its hardware, the rate of enhancement to its instructions per clock (IPC), and the optimal size and the optimal replacement algorithm for the NBTB. Because software CPU is widely used in current processors, we use the FPGA to measure the hardware size now and will try to install our BTB into software CPU in the future.

Initial research leading to the work described in the paper was presented in [15]. However the initial research only evaluated the method on software. In this paper, we evaluated the proposal on hardware by using FPGA. We have used both software and hardware simulation results to discuss and achieve an optimal CAM-based Separated BTB. In addition, more graphs have been added to make our proposed method and explanations clearer.

The contributions of this paper are as follows:

1) A new BTB is proposed for reducing BTB misprediction, and the BTB is optimized.
2) The proposal is quantitatively evaluated using a SimpleScalar tool set [16] for IPC performance and FPGA for hardware size.
3) A replacement algorithm of CAM for optimizing the NBTB is quantitatively evaluated.

The rest of this paper is organized as follows. Sections II and III describe the current BTB and the proposal. Section IV presents an overview of NBTB. Sections V and VI explain software and hardware experiments. Section VII concludes this paper.

II. Branch Target Buffer

A. Structure of Simple BTB

BTB essentially involves a smaller cache to retain both the branch program counter (*PC*) and target address. Fig. 1 has a block diagram of the simple BTB, where *tag* keeps the $PC[n-1,\ i]$, and *target address* keeps the branch's result when the branch is taken or the branch is a non-conditional branch.

Manuscript received May 13, 2014; revised February 1, 2015.

Authors are with the Department of Electronic and Computer Engineering, Ritsumeikan University, Kusatsu, Shiga, Japan (e-mail: menglin@fc.ritsumei.ac.jp, {kumaki@fc, togura@se}.ritsumei.ac.jp).

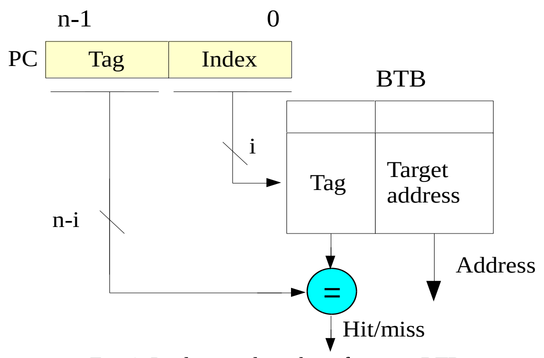


Fig. 1. Prediction algorithm of current BTB.

```
//IF Stage, Prediction
Algorithm (INS, BTB, PC) {
//INS:instruction; Cbranch: Conditional branch,
//Nbranch: Non-conditional branch
if(INS∈branch && INS != JR) then
  index = PC[i-1:0];
  access BTB:
  if (BTB[index].Tag == PC[n-1:i])  then
    BTB_Hit = true;      // BTB hit
  else
    BTB_Hit = false;    // BTB miss
  end if

  if (BTB_Hit == true) then
    if ( (INS ∈Nbranch) ||
        (INS ∈Cbranch && PHT prediction==T)) then
      Predict_PC = BTB[index].Target_Address;
    else
      Predict_PC = PC + 4;
    end if
  else
    if (INS ∈Cbranch && PHT prediction ==N) then
      Predict_PC = PC + 4;
    else
      Stall;
    end if
 end if
```

Fig. 2. Prediction algorithm of current BTB.

Prediction: Fig. 2 shows the prediction algorithm of current BTB. When branch instruction is fetched, the processor uses $PC[i-1, 0]$ as the index to search for the *tag* and *target address* in BTB. It then compares the BTB *tag* with $PC[n-1, n-i]$. When they are the same, this means a BTB hit, and the target address will be used as the next program counter when the branch is taken or it is a non-conditional branch. Otherwise, it means a BTB miss, and the processor will stall when the branch is taken or it is a non-conditional branch. Because JR instruction uses return address stack (RAS) [17], the JR does not use BTB.

Updating: Fig. 3 shows the updating algorithm of current BTB. When the branch is taken or a non-conditional branch, the branch information (*PC* and *target address*) will be updated if a BTB miss or misprediction occurs. The branch result will be updated into the target address of BTB when updating is done, and the $PC[n-1, i]$ will be updated into the tag of BTB. The updating entry of BTB is $PC[i-1, 0]$. Here, we know that the BTB entry size is $m * 2n$. The current BTB uses 4-set associative BTB, which maintains a 4-set simple BTB [18].

```
//WB Stage, BTB Updating
Algorithm (INS, BTB, PC) {
// Next PC is next instruction PC of branch
if((INS∈NBranch && INS !=JR) ||
    (INS∈CBranch  && Next_PC != PC+4)) then
  Update BTB:
    index = PC[i-1:0];
    BTB[index].Target_Address =Next_PC;
    BTB[index].Tag = PC[n-1:i]
end if
}
```

Fig. 3. Updating algorithm of current BTB.

B. Related Work

Several BTB structures have been proposed that, like the proposed BTB, use several kinds of BTB.

Bray and Flynn separated current BTB into several BTBs by the individual branch taken percentage. The processor needs to count the number of executed branch times and the branch taken percentage [4]. However, our proposal does not need to count the branch taken times and branch execution times or calculate the branch taken percentage.

BTB Access Filtering (BAF) uses a smaller filter buffer (FB) to reduce the BTB access times to decrease BTB access energy [5]. In this method, FB is accessed in parallel with a predictor, if the branch is predicted as NotTaken or if the branch is predicted as taken and the FB hits, so the BAF will not involve the BTB in the next cycle. Because the FB is very small, the energy of accessing current BTB can be saved. However, this method uses several cycles to access BTB when FB misses. Otherwise, our method accesses the BTB for only one cycle.

Extended BTB [2] keeps the misprediction times of every BTB entry to detect the misprediction bias branch and improve the accuracy of PHT prediction. The BTB of Saito and Yamana [3] changed the prediction and update method when a BTB miss happens. However, the conditional branch and non-conditional branch still conflict.

III. Separated BTB

A. Overview of Separated BTB

From the Prediction and Updating algorithm of current BTB, we find that the conditional branch and non-conditional branch may access the same entries. As we know, the different branches access the same entries, causing misprediction to happen.

By using this character, we separated BTB into two BTBs (CBTB and NBTB). Fig. 4 shows the block diagram of separated BTB. CBTB and NBTB are used to predict the target addresses of conditional and non-conditional branches, respectively. CBTB is organized as a SRAM that works the same way as the current BTB.

NBTB is the proposed hardware. For optimization, we will test it by using SRAM and CAM. When it uses SRAM, the action is the same as the current BTB. When it uses CAM, the NBTB has to keep the target addresses. Because there are not that many non-conditional branches, the NBTB can be small. (The NBTB in Fig. 4 is a CAM structure.)

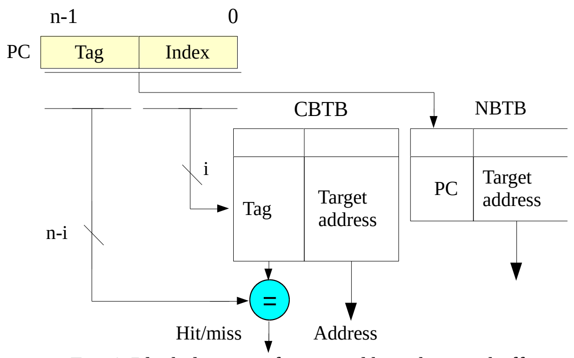


Fig. 4. Block diagram of separated branch target buffer.

This organization can reduce BTB conflict between conditional and non-conditional branches. When the NBTB is structured by CAM, although fewer entries into NBTB can save on hardware costs, the NBTB will quickly fill. Therefore, the size of the NBTB is an important factor in designing our BTB. Hence, CAM that maintains a PC must be replaced, and the replacement algorithm is also an important factor. We considered two algorithms: the LRU and a rotation method using a rotator pointer.

The actions for the proposed BTB can be explained as follows.

```
//IF Stage, Prediction
Algorithm (INS,BTB,PC){
//INS:instruction; Cbranch: Conditional branch,
//NCbranch: Non-conditional branch
if(INS∈ NCbranch && INS != JR) then
access NBTB
if (BTB_Hit == ture)
Predict_PC = BTB[index].Target_Address;
else
Stall;
  end if

else if(INS∈ Cbranch) then
access CBTB // using same algorithm of current BTB
end if
}
```

Fig. 5. Prediction algorithm of separated BTB.

```
//WB Stage, BTB Updating
Algorithm (INS,BTB,PC){
// Next PC is next instruction PC of branch
if ((INS∈NCBranch && INS !=JR) then
Update NBTB:
end if
else if (INS∈CBranch) then
  Update CBTB: // using same algorithm of current
BTB
end if
}
```

Fig. 6. Updating algorithm of separated BTB.

Prediction: Fig. 5 shows the prediction algorithm of separated BTB. When a conditional branch is fetched, the predictor uses CBTB to predict the branch target address as the current BTB. When a non-conditional branch is fetched, a PC will be searched for in NBTB. If a PC is found in NBTB, the target will be used as the next PC. Otherwise, the processor stalls the instruction.

Updating: Fig. 6 shows the updating algorithm of separated BTB. When the conditional branch commits, the predictor uses CBTB to update the branch result as the current BTB. When the non-conditional branch commits, the PC will be searched for in NBTB. If misprediction occurs, the target address will be updated into NBTB. If a BTB miss occurs, the target address and PC will be updated into NBTB. Otherwise, nothing is done.

B. CAM Replacement Method

LRU: this method keeps the used BTB entries and updates the new information into the entry that is not least recently used.

Rotation method: this method uses a rotator pointer to maintain the update entry.

Fig. 7 shows the two replacements methods. We find the rotation method is very simple since it just uses a rotator pointer.

```
//LRU
Replacement Algorithm (BTB,New_data){
When (BTB Accessed)
Recode the accessed entries;
When (update New_data)
Search the least recently used entry and update;
}

//Rotation method
//Rpointer: rotator pointer
Replacement Algorithm (BTB,New_data){
When (update New_data)
Update the new data into Rpointer;
Rpointer++;
}
```

Fig. 7. Replacement methods for CAM.

IV. Overview of NBTB

A. NBTB Mode

When the NBTB uses CAM, the branch address in NBTB is updated in a CAM, and the target address is updated in a SRAM.

Fig. 8 outlines the action flowchart for BTB. It includes various action modes, i.e., the commit, fetch, commit & fetch, and no operation (NOP) modes.

Fetch mode: This means that the non-conditional branch is fetched and the target address will be predicted. The fetch mode is indicated as action 1 in Fig.8. When the non-conditional branch is fetched, the src (search) signal will be set to search for the fetch instruction in CAM. If it is hit, the target address will be read from RAM; otherwise, 'Z' will be output.

Commit mode: This means that the branch and the target address will be updated into BTB when non-conditional misprediction occurs. The commit mode is indicated as action 2 in Fig. 8. The src (search) signal will be set to search for the commit instruction in CAM when in commit mode. If it is hit, the target address will be updated in RAM; otherwise, the branch address will be updated into CAM and the target address will be updated in RAM.

Commit & fetch mode: This means a non-conditional branch is fetched and the target address needs to be predicted. A non-conditional branch misprediction simultaneously occurs, and the information on branches will be updated. It will run actions 1 and 2 at the same time.

NOP mode: This means no non-conditional branch is fetched and no non-conditional branch misprediction occurs. Therefore, the NBTB does nothing in this mode.

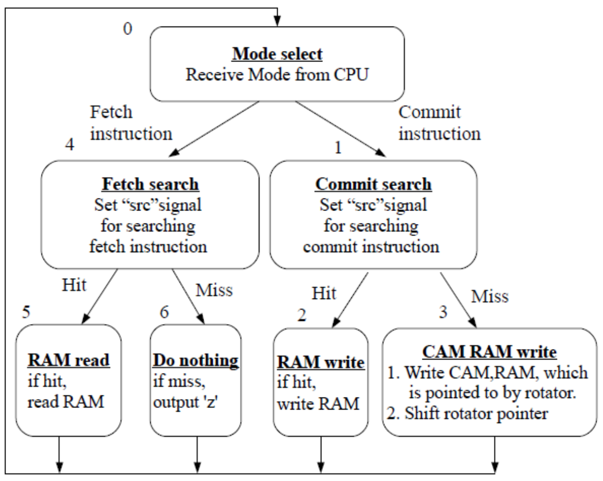


Fig. 8. Flowchart for NBTB.

B. *Hardware Action*

In this section, we describe the hardware action of NBTB. Here we show the NBTB is CAM and replacement algorithm is rotation.

1) Fetch action

Fig. 9 outlines the fetch action in NBTB. First, PC is searched in CAM, which retains the branch address. Here, we prepared a fetch hit flag to recode the hit information of fetch for all entries. If it is hit, the fetch hit flag of the entry is set to one; otherwise, it is set to zero. Then, the target address of NBTB is set to a selector and is selected as the final result when the entry of the hit fetch flag is one. If all the fetch hit flags are zero, high-impedance will be output. This means BTB is missed.

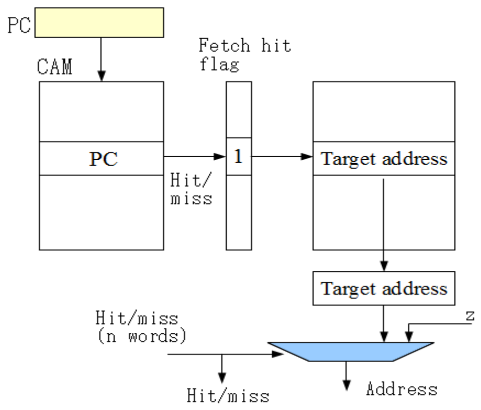


Fig. 9. Fetch action of NBTB.

2) Commit action

Fig. 10 outlines the commit action in NBTB. Here we use the rotation methd to replace the CAM. First, PC will be searched for in CAM, which retains the branch address. Here, we prepared a commit hit flag to recode the hit information of commit for all entries. If it is hit, the commit hit flag of the entry will be set to one, or else *it* will be set to zero.

Because NBTB is structured by CAM, it is full. Here, we prepared a rotator to point to the entry where we could store the new branch when CAM was full. The rotator is one bit, which is kept in all entries. When the entry for the rotator is one, the entry can be used to store the new branch.

When the branch information is updated into NBTB, NBTB uses a selector to select the entry. If the commit hit flag has one in the entry, the target address will be updated. The update entry is where the commit hit flag is one.

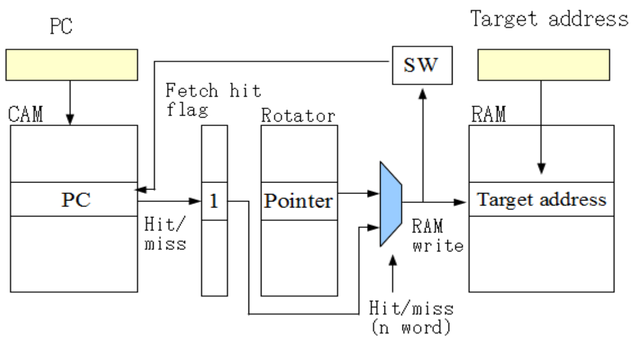


Fig. 10. Commit action of NBTB.

When there are no ones in the commit hit flag, this means there are no branches in NBTB. Therefore, the branch address and target address will be updated. The update entry is where the entry's rotator is one (rotator point). Then, the rotator point will be shifted.

V. IPC Evaluation

We evaluated our proposal on the SimpleScalar tool set to measure IPC. Table I lists the processor configuration parameters. The instruction set is the portable instruction set architecture (PISA: MIPS-like instruction set); the benchmarks include bzip, gcc, gzip, parser, twolf, vpr, and vortex from SPECint2000. We skipped the first 50 million instructions and executed the next 100 million.

TABLE I: Processor Configuration

Pipeline	5 stages: 1 fetch, 1 decode, 1 execute, 1 memory access, and 1 commit
Fetch Decode	4 instructions
Issue	Int: 4, fp: 2, mem: 2
Window	Dispatch queue: 256, Issue queue: 256
Branch Predictor	64K-entry combining predictor, 2K-entry 4-way associative BTB, 32-entry RAS
Memory	64KB, 4-way associative, 1-cycle instruction and data caches, 2MB, 8-way associative, 10-cycles L2

We know the memory structures are CAM and RAM, and the CAM replacement algorithms include LRU and the one using a rotator. In this section we will show the IPC performance uprate for the flowing cases.

1) The current BTB is compared with the separated BTB that uses an NBTB consisting of an SRAM.
2) The current BTB is compared with the separated BTB that uses an NBTB consisting of the CAM whose replacement algorithm is LRU.
3) The current BTB is compared with the separated BTB that uses an NBTB consisting of the CAM whose

replacement algorithm is the one using a rotator.

The experiment results show that the CAM with the replacement algorithm using the rotator is the best for the proposed BTB.

Tables II and III list the memory configurations of the NBTB (SRAM and CAM) on which we experimented.

TABLE II: MEMORY CONFIGURATION OF NBTB (SRAM)

Entries	Bit (Fixed length)	
(Variable depth)	RAM (PC)	RAM (Target address)
32 entries	25 bits	30 bits
64 entries	24bits	
128 entries	23bits	
256 entries	22bits	
512 entries	21bits	
1024 entries	20bits	
2048 entries	19bits	

TABLE III: MEMORY CONFIGURATION OF NBTB (CAM)

Entries	Bit (Fixed length)	
(Variable depth)	RAM (PC)	RAM (Target address)
32 entries	30 bits	30 bits
64 entries		
128 entries		
256 entries		
512 entries		
1024 entries		
2048 entries		

A. IPC Performance of NBTB (SRAM)

Since we know the current BTB consists of a SRAM, we measured the performance of the separated BTB whose NBTB is an SRAM. Fig. 11(a) shows the IPC performance uprate of the proposed BTB compared with the 2048-entry 4-set associate current BTB. The horizontal axis indicates the benchmarks from SPECint2000. The vertical axis indicates the rate of improvement in IPC for the current BTB, where the NBTB consists of an SRAM. The CBTB of the proposed BTB is a 2048-entry 4-way associate BTB. The NBTB size ranges from 32 to 2048 entries. The results show that the NBTB consisting of an SRAM has about 2.45% IPC performance improvement when the average size is 256 entries. It also improved more than 5% on average when there were more than 512 SRAM entries. The results revealed that more than 256 NBTB entries were necessary to reduce the number of BTB misses. They also indicated that our proposed BTB could not improve performance when there were more than 512 NBTB entries.

B. IPC Performance of NBTB (CAM with LRU Replacement)

Fig. 11(b) shows the IPC performance uprate of the proposed NBTB compared with the 2048-entry 4-set associate current BTB. The proposed NBTB of the proposed BTB ranges from 32 to 2048 entries. The replacement algorithm is LRU. Again, the CBTB is a 2048-entry 4-set associate BTB.

The experiment results indicated that the 32-entry NBTB decreased IPC performance and the 64-entry NBTB improved it only slightly. The reason for this is the conflict in non-conditional branches when NBTB was too small. However, when there were 128 CAM entries, our proposed BTB increased IPC by 3.36% on average. It also increased it by more than 5.63% on average when there were more than 128 CAM entries. The results revealed that more than 128 NBTB entries were necessary to reduce the number of BTB misses. They also indicated that our proposed BTB could not improve performance when there were more than 512 NBTB entries.

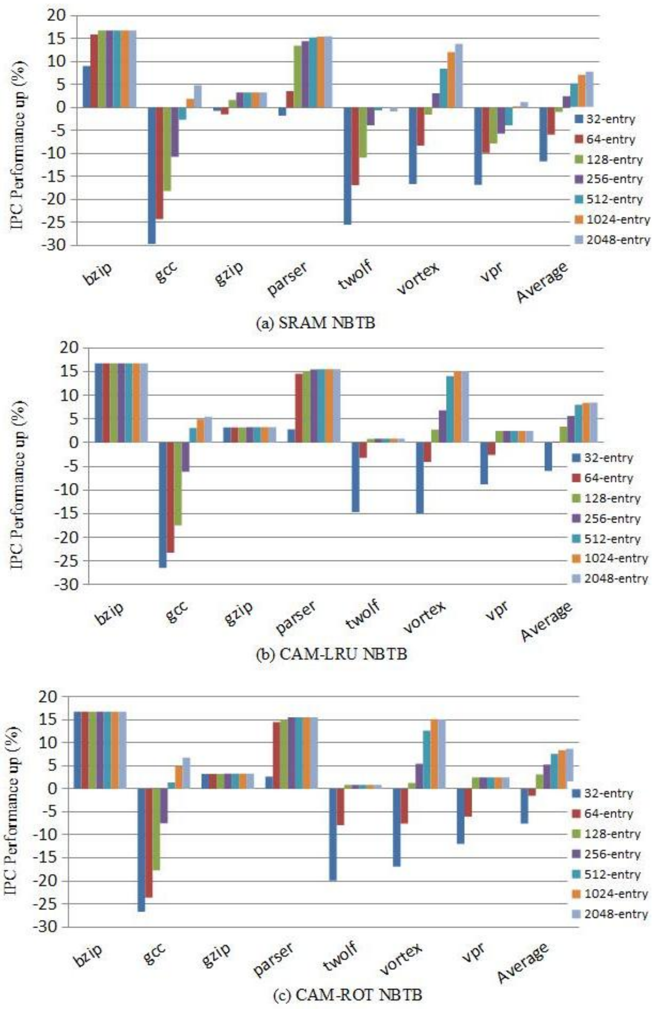


Fig. 11. Improved IPC performance uprate of NBTB: (a) SRAM, (b) CAM-LRU, and (c) CAM-ROT.

C. IPC Performance of NBTB (CAM with Rotator)

Fig. 11(c) shows the IPC performance uprate of the proposed NBTB compared with the 2048-entry 4-set associate current BTB. The proposed NBTB of the proposed BTB ranges from 32 to 2048 entries. The replacement algorithm uses a rotator. Again, the CBTB is a 2048-entry 4-set associate BTB.

The experiment results indicated that the 32-entry NBTB decreased IPC performance and the 64-entry NBTB improved it only slightly. The reason for this is the conflict in non-conditional branches when NBTB was too small. However, when there were 128 CAM entries, our proposed BTB increased IPC by 3.12% on average. It also increased it by more than 5.23% on average when there were more than 128 CAM entries. The results revealed that more than 128 NBTB entries were necessary to reduce the number of BTB misses. They also indicated that our proposed BTB could not improve performance when there were more than 512 NBTB entries.

From the experiment results, we find an NBTB constructed by an SRAM has good performance improvement when the entry has 512 entries. When CAM is used, the proposed BTB shows good performance improvement when the entry has 128 entries. We therefore conclude that using CAM for our BTB is an effective method.

For the CAM replacement algorithm, we found that the two kinds of replacement algorithms (LRU and rotator) yielded similar IPC improvement with the same entry size in additional experiments. We know that LRU needs a double-entry matrix to retain renewal information on CAM. However, the rotator needs only to retain a pointer to renew CAM. Therefore, since the rotator is more suitable than LRU with respect to hardware, we conclude that a 128-entry CAM with a rotator is optimal for a separated BTB.

D. *IPC Performance by Changing the BTB Size*

In the last subsection, we found that the 128 entry CAM with a rotator is optimal for a separated BTB. We also changed the current BTB to compare it with the 128-entry CAM BTB whose replacement algorithm uses a rotator. Fig. 12 shows the IPC performance uprate of the proposed NBTB compared with the current BTB. The CBTB of the proposed BTB is a 2048-entry 4-set associate BTB. The current BTB has 2048, 4096, or 8196 entries, all of which are 4-set associative. Since we find the IPC performance uprate is about 3% for the three kinds of current BTBs, we conclude that a 128-entry CAM with a rotator is optimal for a separated BTB.

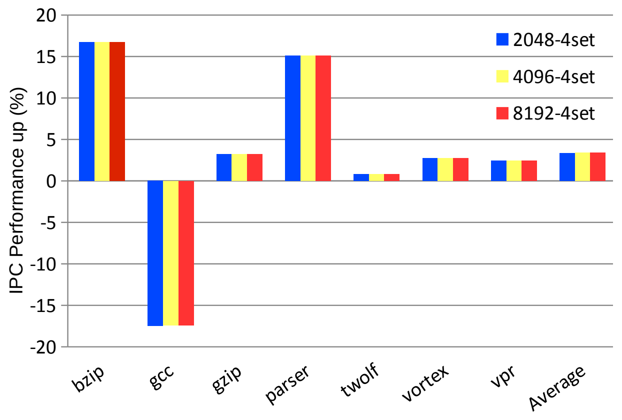


Fig. 12. Improved IPC performance uprate.

VI. Hardware Evaluation

We used Xilinx FPGA tools to measure the size of additional hardware in the current stage.

A. *Hardware Verification*

Table IV summarizes the hardware size for the proposed BTB, which was measured with an FPGA. The hardware design tools were those in the ISE Design Suite 13.1 Project Navigator of Xilinx. The target device was the Virtex6 xc6vlx240t-1ff1156.

The number of CBTB entries, 2048, was equal to the number of CBTB entries. The NBTB entries comprised the NBTB entries, which ranged from 32 to 2048 in the experiment. In the experiment, we measured only a 1-way CBTB and the 4-way associate BTB size was four times that of the 1-way CBTB.

We found that doubling the number of entries doubled the NBTB hardware size requirements. The NBTB was larger than the CBTB in terms of hardware size for the same number of entries. Because the NBTB used CAM, CAM needed more hardware for searches than SRAM (CBTB).

TABLE IV: Memory Configuration of NBTB (CAM)

	Memory config. (Entries)	Bit (Fixed length)		
		Registers	LUTs	LUT-FF Pairs
NBTB	32	2022	3042	3424
	64	4041	7119	7126
	128	8074	13569	13571
	256	16148	28098	28016
	512	32270	54201	54212
	1024	64522	111203	112583
	2048	129048	217195	219846
CBTB (1-way)	2048	100354	103148	103148
CBTB (4-way)	2048	401416	412592	412592

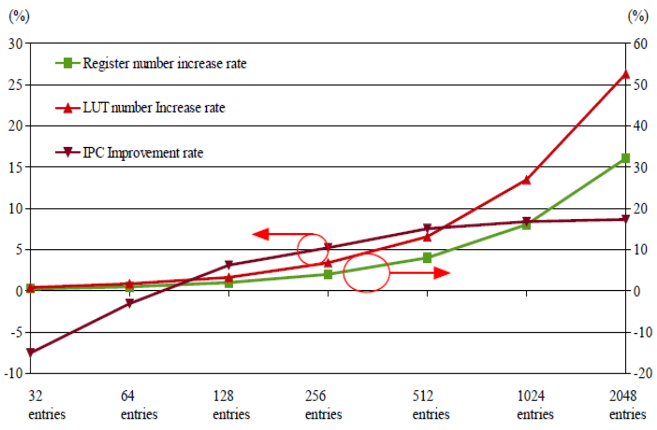


Fig. 13. Relationship between hardware increase rate and performance improvement rate.

B. *Discussion and Optimization*

Fig. 13 plots the relationship between the IPC improvement rate and the rates at which hardware was added, indicating the optimal size of our proposed BTB. The horizontal axis indicates the number of NBTB entries. The vertical axes indicate the IPC improvement rate and the rates at which lookup tables (LUTs) and registers were added. Because the rate at which LUT-FF pairs were added was similar to that for LUTs, we did not add LUT-FF information to the figure.

We used two kinds of standard axes to clarify the discussion. The left axis is for the IPC improvement rate, and the right one is for the rates at which LUTs and registers were added.

The results show that the IPC improvement rate and the rates at which hardware was added crossed at 64 and 512 entries. This makes it clear the optimal hardware sizes are between 64 and 512 entries.

We found that a 64-entry NBTB showed -1% performance improvement, a 256-entry NBTB needed about a 5.2% increase in hardware size to improve 5.4%, and a 128-entry NBTB needed 3.1% additional hardware to improve 2.6%. Consequently, the optimal number of entries was 128.

VII. CONCLUSION

It is important to improve the accuracy of predicting BTB in modern processors that exploit instruction level parallelism with deeper pipelines and wider instruction issues. Thus, this paper proposed a novel BTB by separating the current BTB into NBTB and CBTB. NBTB and CBTB were used for predicting non-conditional and conditional branches, respectively. This reduced the number of BTB misses that were caused by conflicts.

We used the SimpleScalar tool set in evaluations to measure IPC and used Xilinx FPGA tools to measure the size of additional hardware. The experimental results revealed that our proposed method could improve IPC when there were more than 64 NBTB entries. We found that NBTB with 128 entries was optimum by analyzing the experiments, which resulted in improving IPC by about 3.12%. About 2.6% more hardware than the current BTB also had to be added. However, the added NBTBs (CAM with 128 entries + SRAM with 128 entries) are insignificant in superscalar processors.

Although our proposed BTB reduced the average number of mispredictions to below that of conventional predictors, the number of mispredictions increased with some benchmarks. These need to be detected and decreased in future work.

REFERENCES

[1] C. H. Perleberg and A. J. Smith, "Branch target buffer design and optimization," *IEEE Transactions on Computers*, vol. 42, no. 4, 1993.

[2] L. Meng, K. Yamazaki, and S. Oyanagi, "A novel branch predictor using local history for miss-prediction bias," in *Proc. the 2012 International Conference on Computer Design (CDES'12)*, Jul. 2012, pp. 77-83.

[3] F. Saito and H. Yamana, "The branch predictor referring a BTB entry existence," *Transaction of IPSJ*, vol. 45, no. 7, pp. 71-79, Otc. 2004

[4] B. K. Bray and M. J. Flynn, "Strategies for branch target buffers," Technical Report, no. CSL-TR-91-480, 1991.

[5] S. Wang, J. Hu, and S. G. Ziavras, "BTB access filtering: A low energy and high performance design," in *Proc. Symposium on VLSI*, 2008, pp. 81-86.

[6] Y. J. Chang, "An energy-efficient BTB lookup scheme for embedded processors," *IEEE Transactions on Circuits and System-II: EXPRESS BRIEFS*, vol. 53, no. 9, pp. 817-821, 2006.

[7] S. Kim, E. Jo, and H. Kim, "Low power branch predictor for embedded processors," in *Proc. 10th IEEE International Conference on Computer and Information Technology*, 2010, pp. 107-114.

[8] XILINX. [Online]. Available: http://japan.xilinx.com/tools/microblaze.htm

[9] ALTERA. [Online]. Available: http://www.altera.co.jp/devices/processor/nios2/ni2-index.html

[10] M. Meribout, T. Ogura, and M. Nakanishi, "On using CAM concept for parametric curve extraction," *IEEE Trans. Image Processing*, vol. 9, no. 12, 2000.

[11] T. Ikenaga and T. Ogura, "CAM^2: A Highly-parallel Two-dimensional Cellular Automaton Architecture," *IEEE Trans. Comput.*, vol. 47, no. 7, 1998.

[12] Y. Ishikawa, J. Uchida, Y. Miyaoka, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A CAM processor optimizing method with area constraints," IEICE Technical Report, vol. 103, no. 705, pp. 13-18, 2004.

[13] T. Kumaki, K. Iwai, and T. Kurokawa, "A flexible multi-port content addressable memory," *IEICE Transactions on Information and Systems,* vol. J87-D-1, no. 1, pp. 12-21, 2004.

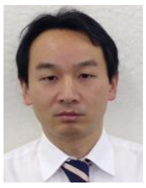
[14] C. C. Kavar and S. S. Paramar, "Performance analysis of LRU page replacement algorithm with reference to different data structure," *International Journal of Engineering Research and Application,* vol. 3, no. 1, pp. 2070-2076, 2013.

[15] K. Fududa, L. Meng, T, Kumaki, and T. Ogura, "A CAM-based separated BTB for a superscalar processor," in *Proc. 2013 First International Symposium on Computing and Networking*, 2013, pp. 385-388.

[16] D. Burger and T. M. Austin, "The simple scalar tool set version 2.0," Technical Report, University of Wisconsin-Madison Computer Sciences Dept. July 1997.

[17] D. Ye and D. Kaeli, "A reliable return address stack: microarchitectural features to defeat stack smashing," in *Proc. ACM SIGARCH Computer Architecture News-Special Issue: Workshop on Architectural Support for Security and Anti-virus (WASSA),* vol. 33, no. 1, 2005, pp. 73-80.

[18] A. Hilton and A. Roth, "Ginger: Control independence using tag rewriting," in *Proc. 35th Int'l Symposium on Computer Architecture,* May 2007, pp. 436-447.



Lin Meng received a Ph.D. degree in computer science from Ritsumeikan University, Shiga, Japan, in 2012. From 2011 to 2013, he was an associate researcher in the Department of VLSI System Design, Ritsumeikan University. Since 2013 he has been an assistant professor at the Department of Electronic and Computer Engineering, College of Science and Engineering, Ritsumeikan University. His research interests include computer architecture of high performance computing and image processing. Dr. Meng is a member of the IEICE and the IPSJ.

Kosaku Fukuda received his B.S. degree in the Department of VLSI System Design, Ritsumeikan University in 2012. He is a master student at the Department of Electronic and Computer Engineering, Ritsumeikan University. His research interests are memory design and VLSI design.

Takeshi Kumaki received his B.S. degree from the Department of Mathematics, Faculty of Science and completed the first half of the M.E. program in information mathematics from NDA, Kanagawa, Japan in 1998 and 2003, respectively, and Ph.D. degree in electric engineering from Hiroshima University, Hiroshima, Japan in 2006. From 2005 to 2009, he joined the RCNS and RNBS, Hiroshima Univ., Japan. From 2010 to 2012, he was a lecture in the Department of VLSI System Design, Ritsumeikan University. Since 2013, he has been a lecture in the Department of Electronic and Computer Engineering, Ritsumeikan University. Dr. Kumaki is a member of the IEEE, IEICE.

Takeshi Ogura received the B.S., M.S., and Ph.D. degrees in electrical engineering from Osaka University, Osaka, Japan, in 1976, 1978, and 1991, respectively. In 1978, he joined Nippon Telegraph and Telephone Corp. (NTT). In NTT, he engaged in the research and development of CAM LSIs and image encoding LSIs, and their applications. Since 2004 he has been a professor in the Department of VLSI System Design, Ritsumeikan University, Shiga, Japan. Dr. Ogura is a member of the IEEE, the IEICE, and the IPSJ.