A Low Power I-Cache Design with Tag-Comparison Reuse

Inoue, Koji
Dept. of Elec. Eng. and Computer Science Fukuoka University

Tanaka, Hidekazu
Dept. of Elec. Eng. and Computer Science Fukuoka University

Moshnyaga, Vasily G.
Dept. of Elec. Eng. and Computer Science Fukuoka University

Murakami, Kazuaki
Dept. of Informatics Kyushu University

出版情報：Proc. of the The International Symposium on System-On-Chip (SOC04), pp.61-67, 2004-11. IEEE
バージョン：
権利関係：

A Low-Power I-Cache Design with Tag-Comparison Reuse

Koji Inoue, Hidekazu Tanaka,Vasily G. Moshnyaga
Dept. of Elec. Eng. and Computer Science
Fukuoka University
8-19-1 Nanakuma, Jonan-ku,
Fukuoka 814-0180 JAPAN
{inoue, vasily}@tl.fukuoka-u.ac.jp

Kazuaki Murakami
Dept. of Informatics
Kyushu University
6-1 Kasuga-Koen, Kasuga,
Fukuoka 816-8580 JAPAN
murakami@i.kyushu-u.ac.jp

Abstract

This paper reports design and evaluation results of a low-energy I-cache architecture, called history-based tag-comparison (HBTC) cache. The HBTC cache attempts to re-use tag-comparison results to detect and eliminate unnecessary memory-array activations. We have performed cycle accurate simulations, and have designed an SRAM core based on a 0.18 μm CMOS technology. As a result, it has been observed that the HBTC approach can achieve 60% of energy reduction, with only 0.3% performance degradation, compared to a conventional cache. Furthermore, we have also evaluated the potential of the HBTC cache by combining with other low-energy techniques.

1. Introduction

On-chip caches are indispensable to design high-performance, low-energy SOCs. This is because confining memory references in on-chip eliminates not only the access latency to the off-chip memory but also the energy consumed for driving external I/O pins. However, with the increase in the cache size, its energy consumption has been becoming significant. Instruction caches (or I-caches) particularly affect total energy consumption due to their high access frequency. For instance, ARM920T microprocessor dissipates 25% of its total power in the I-cache [9].

In conventional set-associative caches, or SA caches, all the ways are activated in parallel at every cache look-up. This behavior dissipates large amount of energy unnecessarily because at most only one way can include the target instruction. In order to solve the energy issue, we have proposed a low-energy set-associative instruction cache called *history-based tag-comparison cache (HBTC cache)*[2]. The HBTC cache attempts to re-use previously generated tag-comparison results in order to activate selectively only the hit-way which includes the target instruction. In the previous work, we have done a primary evaluation by using an energy model assuming 0.8 μm old CMOS technology [4][5]. In this paper, the energy/performance efficiency of the HBTC cache is discussed based on the results from cycle-accurate simulations and a 0.18 μm CMOS custom design. Moreover, we compare the HBTC cache with previously proposed low-energy approaches, and evaluate its potential by combining with other techniques.

The rest of this paper is organized as follows: Section 2 presents briefly the organization and operation of the HBTC cache, and Section 3 evaluates its energy/performance efficiency. In Section 4, we conclude this paper.

2. The History-Based Tag-Comparison Cache

In conventional I-caches, only when a miss takes place, the state (or contents) of the cache is changed by evicting some instructions and filling missed ones. This means that once an instruction is loaded into the cache, it stays there at least until the next cache miss occurs. Now, consider in the case that an instruction is executed repeatedly due to temporal locality, e.g. loop executions. At the first reference of the instruction, tag checks have to be performed to identify the hit-way which includes the target instruction. However, at and after the second reference, if no cache miss has occurred, it is guaranteed that the target instruction still stays at the same location. In other words, the tag-comparison results generated at and after the second reference are exactly same as that of the first reference, if we do not have any cache misses. Unfortunately, since conventional caches do not take account of the access history, all the ways are always searched in parallel as shown in Figure 1(a), thereby dissipating large amount of energy unnecessarily. To solve the energy issue, the HBTC cache re-uses the tag-comparison result of the first reference for future accesses. When a tag-comparison result is re-used, the cache

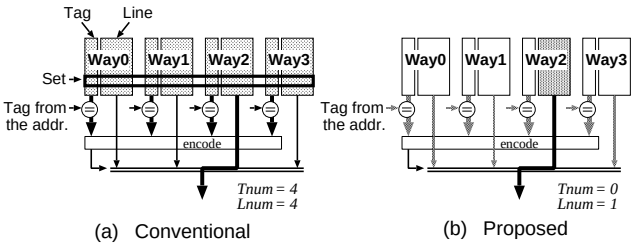

Figure 1. Conventional vs. HBTC cache

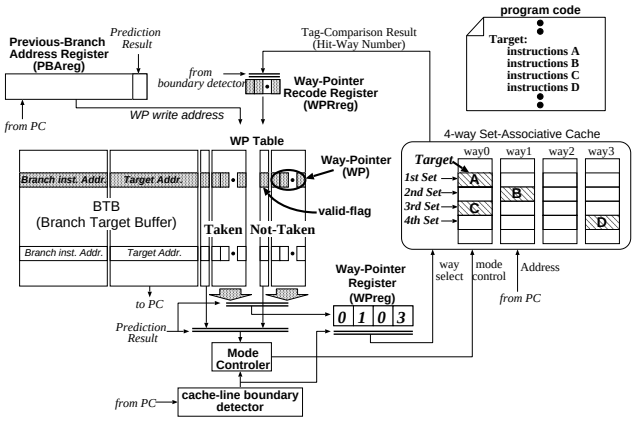

Figure 2. Organization of a 4-way HBTC cache

directly activates only the hit-way as depicted in Figure 1 (b), resulting in low-energy consumption.

Figure 2 shows the organization of a 4-way HBTC cache. The HBTC approach requires five additional components: Way-Pointer table (WP table), Way-Pointer Register (WPreg), Way-Pointer Record register (WPRreg), a mode controller, and Previous Branch-Address register (PBAreg). A conventional BTB is extended by adding the WP table. Each entry of the table corresponds to that of the BTB, and consists of M way pointers. A tag-comparison result (i.e. the hit-way number) is stored in a way pointer (WP), hence it can be implemented by $\log n$ bits where n is the cache associativity. For instance, in case of $M = 4$, each entry of the WP table can contain tag-comparison results for four cache lines which are accessed successively. A 1-bit valid-flag is used for determining whether the M of WPs are valid or not. The taken and not-taken WPs are used for the target and fall-through instructions of each corresponding branch, respectively.

The HBTC cache has three operation modes, **Nmode**, **Omode**, and **Tmode**. When the cache works in the Normal mode (Nmode), it behaves as a conventional I-cache, i.e. tag checks are performed and all the ways are activated in parallel. On the Omitting mode (Omode), the cache reuses tag-comparison results, thus only the hit-way is activated without performing tag checks. In the Tracing mode (Tmode), the cache works as the same as the Nmode, and also attempts to record the tag-comparison results generated by the I-cache.

On every BTB hit, the HBTC cache reads in parallel both the taken and the not-taken WPs associated with the BTB-hit entry, and selects one of them based on the branch prediction result. If the selected valid-flag is '1 ', the operation enters the Omode and the selected WPs are stored to the WPreg. Otherwise the Tmode is activated and both the branch-instruction address (PC) and the branch-prediction result (taken or not-taken) are stored to the PBAreg. In the Omode, the contents of the WPreg are provided to the I-cache as tag-comparison results. On the other hand, in the Tmode, the tag-comparison results generated by the I-cache are written to the WPRreg. When the next BTB hit occurs on the Tmode, the value of the WPRreg is moved into the entry of WP table pointed by the PBAreg, and the corresponding valid-flag is set to '1'. Whenever a cache miss takes place, all WPs recorded in the WP table are invalidated by resetting all the valid flags to '0' and operation transits to the Nmode. This is because instructions corresponding to valid WPs may be evicted from the cache. The invalidation is also performed when a BTB replacement occurs in order to guarantee correct operation. To simplify the mode control, in addition, we assume that the cache operates in the Nmode whenever a branch-target address is provided by a return address stack (RAS), or a branch misprediction is detected.

Here, we give an example of HBTC cache's operation. Assume that a branch instruction denoted as *Bi* is executed repeatedly, and has already been registered in the BTB without any effective WPs (i.e. no tag-comparison results in the WP table). Furthermore, we assume that all branches are predicted as 'taken' and no prediction miss occurs. The HBTC cache works as follows:

1. At the first execution of *Bi*, the HBTC cache reads the corresponding WPs from the BTB. Since there is no valid WPs in this case, the cache moves into the Tmode. Therefore, the PC and prediction result of the current branch are saved into the PBAreg. Then the following instructions are accessed with tag checks, so no energy reduction can be achieved. The tag-comparison results for following instruction accesses are temporally recorded into the WPRreg.

2. Another branch *Bj* is fetched and a BTB hit takes place. At that time, the tag-comparison results recorded in the WPRreg is stored into WP table's entry indexed by the PBAreg (PC and branch-prediction result of *Bi*), and the corresponding valid flag is set to '1'.

3. The branch *Bi* is executed again. On the BTB access, we see the valid WPs which are stored in the step 2. Thus, the WPs read from the BTB are moved into the WPreg and the cache transits to the Omode. Since tag-comparison results are provided from the WPreg to the instruction cache, the following instructions can be accessed by activating only the hit way.

3. Evaluation

In this section, we evaluate the energy/performance efficiency of the HBTC cache. A 16 KB 4-way SA cache with 32 byte line size is assumed. First, we investigate the detail of energy and time to be consumed for each HBTC operation based on a 0.18 μm CMOS design. Then, the results obtained from cycle-accurate simulations and that from the custom design are integrated for the evaluation.

We use an extended version of the SimpleScalar tool set [11]. Five SPEC95 integer benchmark programs with training input data [12], *124.m88ksim*, *126.gcc*, *129.compress*, *130.li*, and *132.ijpeg*, and four Mediabench programs[10], *adpcm*, *mpeg2*, *epic*, and *pegwit* are executed. For each of the Mediabench programs, an encoder and a decoder are executed separately.

In this evaluation, the following parameters are assumed: the number of direct-mapped branch-prediction-table entry is 512, predictor type is bimod, the number of BTB sets is 128, BTB associativity is 4, and RAS size is 8. In addition, to investigate the potential of the HBTC approach, we assume that each i-fetch is performed instruction by instruction (i.e. the fetch width is 1) with an in-order execution. We also use the assumption that the number of WPs implemented in a WP table entry is 4 based on the results presented in the previous work [2].

3.1. Performance/Energy Model

On the Omode, unlike conventional accesses, the HBTC cache activates only the hit-way. Therefore, the energy reduction achieved by the HBTC approach is proportional to the number of accesses in the Omode. The total energy consumed for the HBTC cache, E_{TOTAL}, can be expressed by the following equation

$$E_{TOTAL} = E_{CACHE} + E_{BTBEXT} + E_{LG} \quad (1)$$

where E_{CACHE} is the energy consumed by the I-cache core, E_{BTBEXT} is the energy overhead wasted for accessing the BTB extension, and E_{LG} is the energy dissipated by peripheral logics. In this model, we do not take the energy for accessing the BTB core into account, because it is originally consumed for branch prediction. Moreover, we assume that E_{LG} is trivial. E_{CACHE} can be presented by the following equations

$$E_{CACHE} = E_{DEC} + E_{TAG} + E_{LINE} \quad (2)$$
$$E_{TAG} = N_{tag} \times E_{tag} \quad (3)$$
$$E_{LINE} = N_{line} \times E_{line}. \quad (4)$$

E_{DEC} is the total energy consumed by the address decoder. E_{TAG} and E_{LINE} are the energy for reading tags and cache-lines, respectively. They can be approximated by multiplying the total number of tags (or lines) accessed during the whole program execution, N_{tag} (or N_{line}), and the average energy for reading a tag (or a line), E_{tag} (or E_{line}), from the SRAM array. On the other hand, we breakdown the energy for the BTB extension as follows:

$$E_{BTBEXT} = E_{WPrd} + E_{WPwt} + E_{WPinv} \quad (5)$$
$$E_{WPrd} = N_{wprd} \times E_{wprd} \quad (6)$$
$$E_{WPwt} = N_{wpwt} \times E_{wpwt} \quad (7)$$
$$E_{WPinv} = N_{wpi} \times E_{wpi}. \quad (8)$$

Here, E_{WPrd} and E_{WPwt} are the energy consumed for reading and writing WPs from and to the WP table, respectively. E_{WPrd} depends on the number of WP read accesses, N_{wprd}, and the average energy consumed per read operation, E_{wprd}. The HBTC cache needs to read the WP on every BTB access, thus N_{wprd} becomes the same as the number of BTB look-ups. E_{WPwt} is also presented by the number of WP write accesses, N_{wpwt}, and the average energy for a WP write operation, E_{wpwt}. E_{WPinv} is the energy dissipated for WP invalidations, and is given by the number of invalidations performed, N_{wpi}, and its average energy, E_{wpi}.

From the performance point of view, controlling the WP may degrade performance. Storing the contents of the WPRreg into the WP table may compete with a BTB access for branch-target prediction. In this case, the branch-prediction unit has to wait until the WP write operation is completed. In this evaluation, we assume that the BTB-conflict penalty is one cycle. Also, during the WP invalidation, accessing to the BTB may be prohibited, depends on the implementation. The design explained in Section 3.2 uses a dedicated control line to reset all valid flags, thus it does not conflict with BTB accesses for branch prediction. However, we use a pessimistic assumption that the processor is forced to stall fetching next instructions until WP invalidation is completed, and this penalty is one clock cycle. Note that the HBTC scheme does not affect cache hit rates and access time, thus other disadvantages for performance do not exist.

3.2. Design Results

We have designed a 4KB SRAM array, which organizes one of the ways of the assumed 16 KB four-way SA cache,

Table 1. Design Results of the 4KB SRAM array (Energy and Delay per operation)

I-Cache Core (1 way)					Extended BTB (1 way)			
Energy(pJ)			Delay(ns)		Energy(pJ)			Delay(ns)
E_{dec}	E_{tag}	E_{line}	Access	Cycle	E_{wpwt}	E_{wprd}	E_{wpi}	Invalid
69.2	68.1	872.1	3.0	3.9	88.1	61.3	65.0	0.6

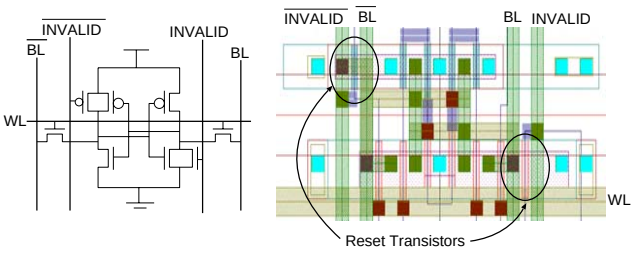

Figure 3. Layout of Resettable Memory Cell

with a 0.18 μm CMOS technology. The circuits have been optimized to meet 3.0 ns access time. Furthermore, we have also designed one column SRAM cells with reset function for the WP valid flags. Figure 3 shows the layout image of the valid flag memory cell. Two transistors are added to a conventional six transistor SRAM cell for the reset operation. Figure 4 is the photograph of the designed SRAM array. After the layout, we have measured energy consumption by performing Hspice simulations with extracted load capacitances. Table 1 reports design results, access time and energy consumption. For E_{tag}, E_{line}, E_{wpwt}, and E_{wprd}, we first obtained the energy for accessing 1-bit memory cell that includes sensing and pre-charging. Then we multiplied it with the total number of bits to be accessed.

In this design, each BTB entry includes four of two-bit WPs and a one-bit valid flag for both taken and not-taken paths as depicted in Figure 2. In addition, since we assume a 4-way set-associative BTB organization, the total number of memory cells accessed for WP read becomes 72 bits (18 bits $\times$ 4). Namely, all the ways in the assumed BTB need to be activated at every WP read operation. Similarly, the WP invalidation is performed on each way. However, E_{wprd} and E_{wpi} reported in Table 1 do not consider the associativity . Therefore, we use the value 245.2 pJ (61.3 pJ $\times$ 4) for E_{wprd} and 260.0 pJ (65.0 pJ $\times$ 4) for E_{wpi} in order to calculate the total energy consumption. Since WP write is performed only to the target way, E_{wpwt} in Table 1 can be used as it is. On the other hand, for the WP invalidation time, we see from the design results that it is much faster than the 4KB SRAM access time. Accordingly, we can complete the WP invalidation in one cycle.

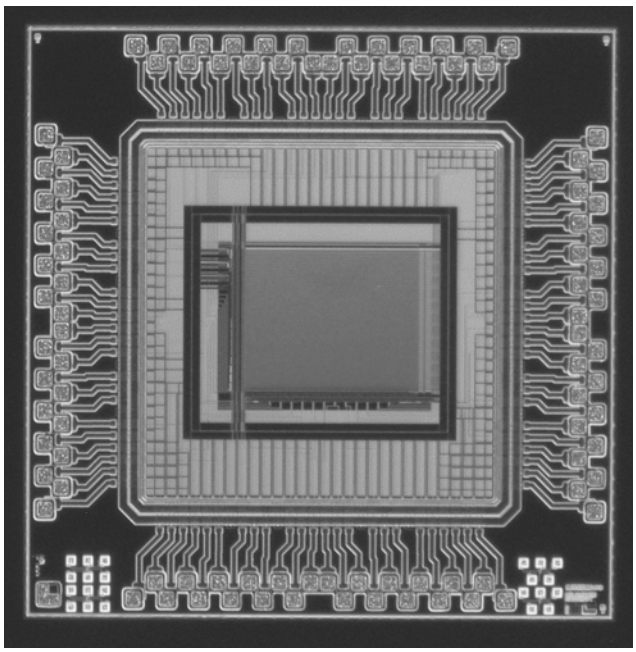

Figure 4. Photograph of the designed SRAM

3.3. Energy Consumption

In this section, we evaluate the energy efficiency of the HBTC cache by comparing with other low-power approaches. **Inter-Line Tag-comparison (ILT)** exploits sequential behavior of instruction references [6][9]. If two instructions *i* and *j* reside in the same cache line, and *j* is referenced immediately after *i*, then it is guaranteed that the hit-way of *j* is the same as that of *i*. Therefore, for *j*, we can activate directly only the hit way as well as the HBTC cache. Another low-power approach is to predict the hit-way before starting normal cache access[3][8]. If the prediction is correct, only the hit-way is activated. We use an MRU-base way-prediction proposed in [3], and refer it as to **PREDppc** which uses previous PC value for way-prediction table look-up. This scheme makes it possible to remove the way-prediction latency from the cache-critical paths.

Figure 5 shows the energy consumption of the HBTC, ILT, and PREDppc caches. Two hybrid models, HBTC with ILT or PREDppc, are also evaluated. All of the results are

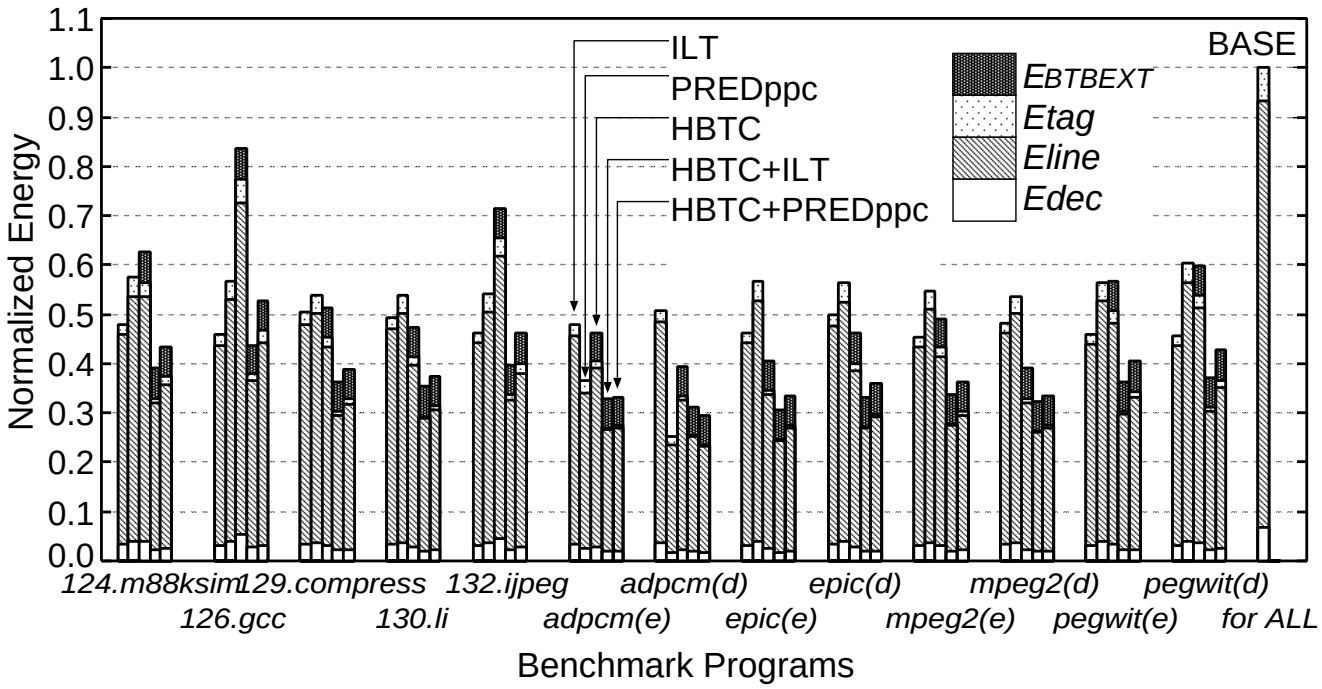

Figure 5. Energy Consumption

normalized to the energy of the conventional cache noted as **BASE**. In the best case, *adpcm(d)*, *epic(e)*, and *mpeg2(d)*, the HBTC cache reduces about 60% of energy consumption by re-using tag-comparison results. Furthermore, for four benchmarks, *130.li*, *epic(e)*, *epic(d)*, and *mpeg2(d)*, the cache produces better results than the other two low-power caches. In particular, the HBTC cache works very well for media applications. This is because the cache attempts to avoid unnecessary cache look-ups by exploiting iterative references of instruction, which is the main feature of media programs. For all but *adpcm(d)*, the hybrid model of the HBTC and ILT achieves the most significant energy reduction. This advantage comes from the fact that we can effectively exploit the benefits of the ILT for sequential accesses and that of the HBTC for non-sequential accesses. The combination with PREDppc also achieves low-energy consumption, the HBTC scheme degrades energy efficiency of the original PREDppc for some programs. For instance, in *adpcm(d)*, the energy dissipated by the extended BTB appears as energy overhead.

3.4. Reducing the Energy of Extended BTB

From the results presented in Section 3.3, we see that the energy consumed by the extended BTB (E_{BTBEXT}) is large. This overhead is mainly caused by reading WPs, because the HBTC cache requires reading WPs on every BTB access. Since the branch prediction is performed before the fetched instruction is decoded, the BTB is accessed on every clock cycle, increasing the number of WP read accesses, N_{wprd}. In order to compensate for the negative effect, we can consider the following three approaches: reducing the number of WPs to be implemented, reducing the number of BTB accesses, and reducing the energy consumed per WP read. To improve the energy efficiency of the HBTC architecture, we have evaluated the following extensions.

- **Taken:** This model employs WPs only for taken paths. Therefore, the number of WPs to be read at BTB accesses becomes half. However, the cache always needs to enter the Nmode when the branch is predicted as not-taken.
- **Not-Taken:** This is the same as **Taken** except that the cache employs WPs only for not-taken paths.
- **Pre-decoding:** In this model, the BTB is accessed only when branches (or jumps) are fetched. In other words, the BTB access is gated on non-branch executions. To implement this kind of gating approach, compiler supports [7] or a pre-decode hardware may be required .
- **BLP:** This model employs bit-line partitioning (BLP) to reduce load capacitances of each bit line [1]. We

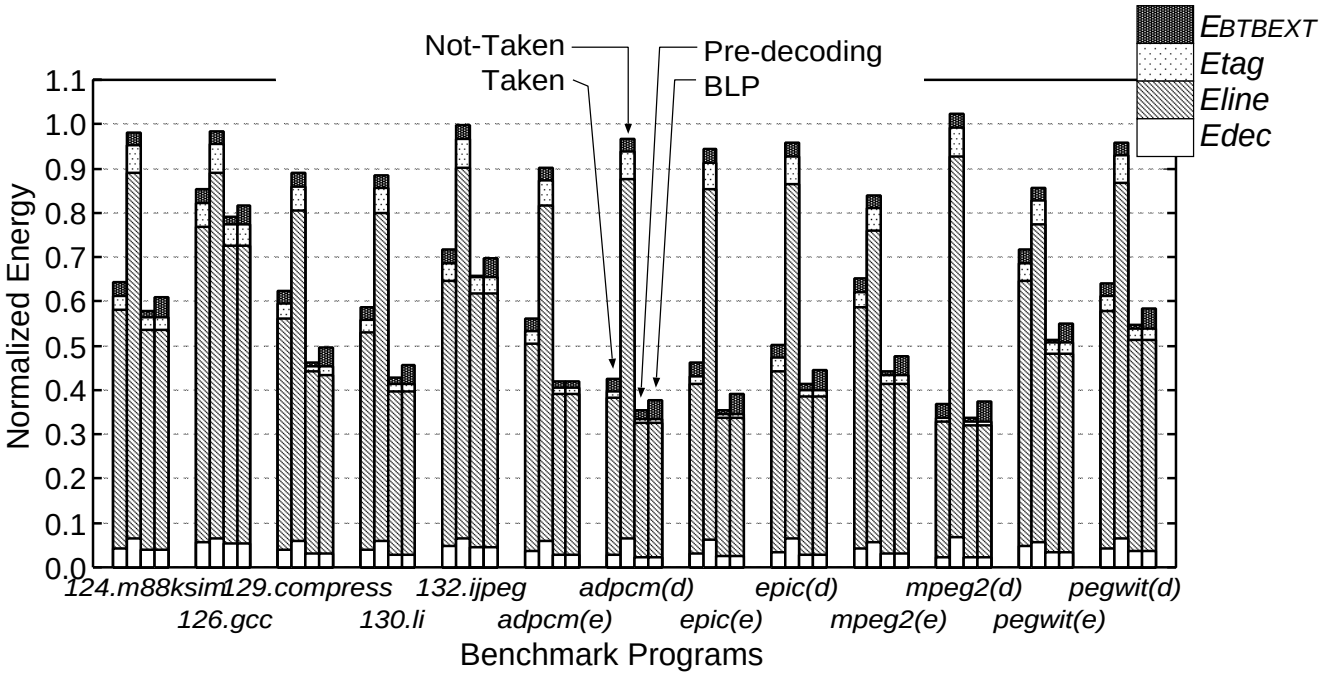

Figure 6. Energy Consumption with improved BTB Design

assume that a bit line is partitioned into four segments (32-bit $\times$ 4-segments).

Figure 6 shows evaluation results. All of the results are normalized to the energy consumed in the conventional organization. We see from the figure that **Not-taken** is inefficient. This is because generally a number of branches are predicted as taken, e.g. loops, thus this model loses opportunity for reusing tag-comparison results. Although **Taken** can produce higher energy-reduction rates, still the normal HBTC model including both taken and not-taken WPs is much efficient, as showed in Figure 5. On the other hand, **Pre-decoding** and **BLP** achieve more energy reduction. Especially, **Pre-decoding** gives the best performance for energy reduction. It reduces the energy consumed by the BTB core itself, too. Therefore, we conclude that the BTB gating is the most promising approach to improving the energy efficiency of the HBTC architecture.

3.5. Performance Overhead

Whenever the HBTC cache write the tag-comparison results into the WP table or a WP invalidation is performed, as explained in Section 3.1, the microprocessor takes one stall cycle. In this section, we evaluate the performance overhead caused by the HBTC approach. Figure 7 depicts the normalized execution time to the conventional cache.

We can see from the figure that the performance impact is quite trivial. In the worst case, *124.m88ksim* and *126.gcc*, the performance overhead is only 1.4%. For other programs, the performance is degraded by less than 0.7%. There are two reasons why the performance overhead is so small. The first reason is that the frequency of the WP write operation is not so high. The HBTC cache write the tag-comparison results to the WP table when a BTB hit occurs on the Tmode. At the first execution of instruction codes, the cache operates in the Tmode. However, after the program execution goes into the hot-spots, e.g. loop executions, the cache works in the Omode. Accordingly, the write operation of the WPs rarely takes place. The second reason is that the WP invalidation penalty can be effectively hidden

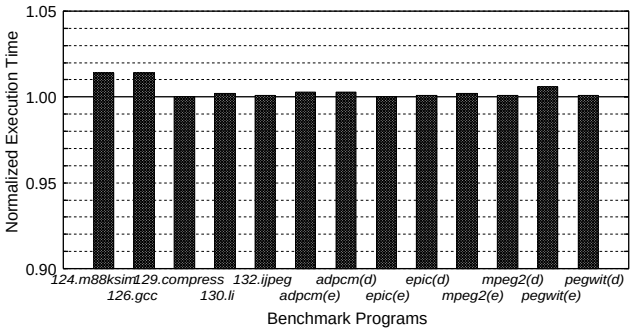

Figure 7. Execution Time

by cache-line replacements. There are two cases for the WP invalidation; a BTB replacement or a cache miss. We have found from the simulation results that more than 90% of invalidations are caused by cache misses for many benchmarks. The invalidation is performed in parallel with the cache-line replacement, thus invalidation penalty can completely be hidden if it is equal to or less than the cache-miss penalty. In this simulation, we have assumed that the cache-miss penalty is six cycles. From the design report in Table 1, we see that the WP invalidation time is much smaller than the 4KB SRAM access time. Therefore, it is possible to complete the invalidation process in one clock cycle. In addition, usually the cache-miss penalty is larger than one cycle. Consequently, we conclude that the performance overhead caused by the HBTC cache is negligible.

4. Conclusions

The on-chip cache is one of the most important components for high-performance, low-energy SOCs. In this paper, we have evaluated the detail of energy/performance efficiency of the HBTC architecture by performing cycle-accurate simulations and designing an SRAM core with a 0.18 μm CMOS technology.

In our simulation, it has been observed that the HBTC cache reduces cache-energy consumption by 60% from the conventional scheme. Moreover, we have reported that more energy reduction can be achieved by combining the HBTC scheme with other low-energy techniques. We have also considered some techniques to improve the energy efficiency of the HBTC architecture, and have found that the BTB access gating effectively compensates for the negative effect of the HBTC scheme. Our ongoing work is to finish the complete HBTC design with peripheral circuits.

Acknowledgments

This research was supported in part by the Grant-in-Aid for Creative Basic Research, 14GS0218 and for Encouragement of Young Scientists (A), 14702064. The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Hitachi Ltd. and Dai Nippon Printing Corporation.

References

[1] K. Ghose and M. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-segmentation," *Proc. of the 1999 Int. Symp. on Low Power Electronics and Design*, pp.70–75, Oct. 1999.

[2] K. Inoue, V. Moshnyaga, and K. Murakami, "A Low Energy Set-Associative I-Cache with Extended BTB," *Proc. of the Int. Conf. on Computer Design*, pp.187–192, Sep. 2002.

[3] K. Inoue, T. Ishihara, and K. Murakami, "Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, pp.273–275, Aug. 1999.

[4] M. B. Kamble and K. Ghose,"Analytical Energy Dissipation Models For Low Power Caches," *Proc. of the 1997 International Symposium on Low Power Electronics and Design*, pp.143–148, Aug. 1997.

[5] M. B. Kamble and K. Ghose,"Energy-Efficiency of VLSI Caches: A Comparative Study," *Proc. of the 10th International Conference on VLSI Design*, pp.261–267, Jan. 1997.

[6] R. Panwar and D. Rennels,"Reducing the frequency of tag compares for low power I-cache design," *Proc. of the 1995 International Symposium on Low Power Electronics and Design*, Aug. 1995.

[7] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. R. Stan, "Power Issues Related to Branch Prediction,"*Proc. of the 8th Int. Symp. on High-Performance Computer Architecture*, pp.233–244, Feb. 2002.

[8] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction annd Selective Direct-Mapping,"*Proc. of the 34th Int. Symp. on Microarchitecture*, pp.54–65, Dec. 2001.

[9] S. Segars, "Low Power Design Techniques for Microprocessors," *ISSCC Tutorial*, Feb. 2001.

[10] MediaBench, *URL: http://www.cs.ucla.edu/˜leec/mediabench/*.

[11] "SimpleScalar Simulation Tools for Microprocessor and System Evaluation," *URL:http://www.simplescalar.org/*.

[12] SPEC (Standard Performance Evaluation Corporation), *URL: http://www.specbench.org/osg/cpu95*.